Reinforcement Learning

Reinforcement learning is learning what to do—how to map situations to actions—to maximize a numerical reward signal

Introduction: Solving interactive problems with Reinforcement Learning

In reinforcement learning, the goal is to develop a system (agent) that improves its performance based on interactions with the environment. Since the information about the current state of the environment typically also includes a so-called reward signal, we can think of reinforcement learning as a field related to supervised learning. However, in reinforcement learning this feedback is not the correct ground truth label or value, but a measure of how well the action was measured by a reward function. Through its interaction with the environment, an agent can then use reinforcement learning to learn a series of actions that maximizes this reward via an exploratory trial-and-error approach or deliberative planning.

Characteristics of Reinforcement Learning:

- There is no supervisor.
  So, there is no one to tell you the best possible action. You just get a reward for each action.
- Sequential decision making
- Time plays a crucial role in Reinforcement problems
  Reinforcement Learning pays much attention to sequential data unlike other paradigms where you receive random inputs. Here the input at the next step will always be dependent on input at previous state.
- Feedback is always delayed, not instantaneous
  You may not get a reward at each step. A reward may be given only after the completion of the entire task. Also, say you get a reward at a step only to find out that you made a huge blunder in a future step.
- Agent's actions determine the subsequent data it receives
  The agents action effects its next input. Say, you have the choice of going either left or right. After you take the action, the input at the next time step will be different if you chose right rather than left.

Popular **Example**:

Chess game. Here, the agent decides upon a series of moves depending on the state of the board (the environment), and the reward can be defined as win or lose at the end of the game. The outcome of each move can be thought of as a different state of the environment. To explore the chess example further, let's think of visiting certain locations on the chess board as being associated with a positive event—for instance, removing an opponent's chess piece from the board or threatening the queen. Other positions, however, are associated with a negative event, such as losing a chess piece to the opponent in the following turn. Now, not every turn results in the removal of a chess piece, and reinforcement learning is concerned with learning the series of steps by maximizing a reward based on immediate and delayed feedback.

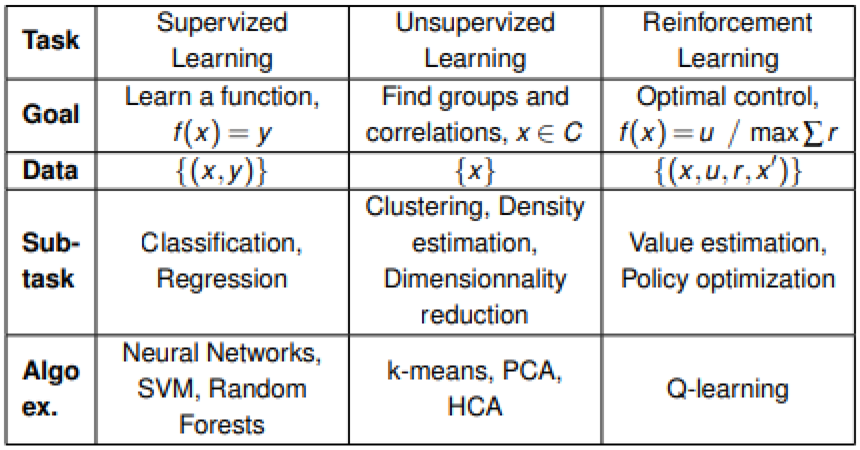
Two distinguishing features of Reinforcement Learning:

- Trial and error Search
- Delayed reward

Reinforcement Learning problem:

In Dynamic systems theory, this can be defined as the optimal control of incompletely-known Markov decision processes.

How is this Learning different from Supervised and Unsupervised Learning:

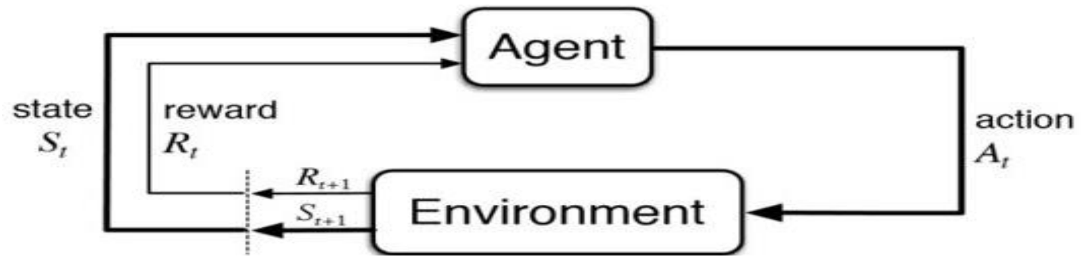


Task	Supervized Learning	Unsupervized Learning	Reinforcement Learning
Goal	Learn a function, $f(x) = y$	Find groups and correlations, $x \in C$	Optimal control, $f(x) = u$ / $\max \sum r$
Data	$\{(x, y)\}$	$\{x\}$	$\{(x, u, r, x')\}$
Sub-task	Classification, Regression	Clustering, Density estimation, Dimensionnality reduction	Value estimation, Policy optimization
Algo ex.	Neural Networks, SVM, Random Forests	k-means, PCA, HCA	Q-learning

Supervised learning is learning from a training set of labeled examples provided by a knowledgeable external supervisor. Each example is a description of a situation together with a specification—the label—of the correct action the system should take to that situation, which is often to identify a category to which the situation belongs. The object of this kind of learning is for the system to extrapolate, or generalize, its responses so that it acts correctly in situations not present in the training set. This is an important kind of learning, but alone it is not adequate for learning from interaction. In **interactive problems** it is often impractical to obtain examples of desired behavior that are both correct and representative of all the situations in which the agent has to act. In uncharted territory—where one would expect learning to be most beneficial—an agent must be able to learn from its own experience.

Unsupervised learning is typically about finding structure hidden in collections of unlabeled data. The terms supervised learning and unsupervised learning would seem to exhaustively classify machine learning paradigms, but they do not. Although one might be tempted to think of reinforcement learning as a kind of unsupervised learning because it does not rely on examples of correct behavior, reinforcement learning is trying to maximize a reward signal instead of trying to find hidden structure. Uncovering structure in an agent's experience can certainly be useful in reinforcement learning, but by itself does not address the reinforcement learning problem of maximizing a reward signal. We therefore consider reinforcement learning to be a third machine learning paradigm, alongside supervised learning and unsupervised learning and perhaps other paradigms.

Reinforcement Learning:



Reinforcement Learning is based on the reward hypothesis. **Reward** is a scalar feedback signal which indicates how well agent is doing at step t. On each time step, the environment sends to the reinforcement learning agent a single number called the reward. The agent's sole objective is to maximize the total reward it receives over the long run. The reward signal thus defines what are the good and bad events for the agent. The agent's job is to maximize the cumulative reward. A learning agent must be able to sense the state of its environment to some extent and must be able to take actions that affect the state. In general, reward signals may be stochastic functions of the state of the environment and the actions taken.

Major components of RL agent:

- Policy: A policy defines the learning agent's way of behaving at a given time. In other words, it is a mapping from perceived states of the environment to actions to be taken when in those states
- Value function: It specifies the value of a state that is the total amount of reward. The value of a state is the total amount of reward an agent can expect to accumulate over the future, starting from that state
- Model: This mimics the behavior of the environment. It helps you to make inferences to be made and determine how the environment will behave. Models are used for planning, by which we mean any way of deciding on a course of action by considering possible future situations before they are experienced.

Reinforcement Learning Algorithms:

- Value-Based:
  In a value-based Reinforcement Learning method, you should try to maximize a value function V(s). In this method, the agent is expecting a long-term return of the current states under policy π.
- Policy-based:
  In a policy-based RL method, you try to come up with such a policy that the action performed in every state helps you to gain maximum reward in the future.
  Two types of policy-based methods are:

Deterministic: For any state, the same action is produced by the policy π.
Stochastic: Every action has a certain probability, which is determined by the following equation. Stochastic Policy: n{a\s) = P\A, = a\S, =S]

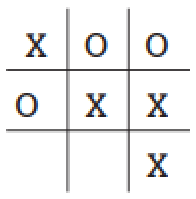
- Model-Based:
  In this Reinforcement Learning method, you need to create a virtual model for each environment. The agent learns to perform in that specific environment.

Exploration and Exploitation:

Main challenge in reinforcement learning is to find the trade-off between exploration and exploitation

Reinforcement learning is like trial and error learning. The agent should discover a good policy from its experiences of the environment without losing too much reward along the way. Exploration finds more information about the environment. Exploitation exploits known information to maximize reward. It is usually important to explore and exploit.

Understanding Reinforcement Learning with an Example: Tic-Tac-Toe



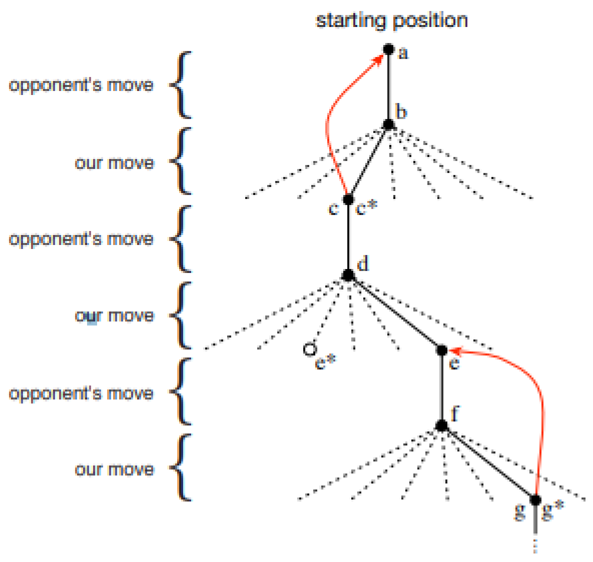
Tic-Tac-Toe is a two-player game where the two players take turns playing on a three-by-three board. One player plays Xs and the other Os until one player wins by placing three marks in a row, horizontally, vertically, or diagonally, as the X player has in the game shown to the right. If the board fills up with neither player getting three in a row, then the game is a draw. Because a skilled player can play so as never to lose, let us assume that we are playing against an imperfect player, one whose play is sometimes incorrect and allows us to win. let us consider draws and losses to be equally bad for us. How might we construct a player that will find the imperfections in its opponent's play and learn to maximize its chances of winning?

Although this is a simple problem, it cannot readily be solved in a satisfactory way through classical techniques. For example, the classical "minimax" solution from game theory is not correct here because it assumes a particular way of playing by the opponent. For example, a minimax player would never reach a game state from which it could lose, even if in fact it always won from that state because of incorrect play by the opponent. Classical optimization methods for sequential decision problems, such as dynamic programming, can compute an optimal solution for any opponent, but require as input a complete specification of that opponent, including the probabilities with which the opponent makes each move in each board state. Let us assume that this information is not available a priori for this problem, as it is not for the vast majority of problems of practical interest. On the other hand, such information can be estimated from experience, in this case by playing many games against the opponent. About the best one can do 1.5. An Extended Example: Tic-Tac-Toe 9 on this problem is first to learn a model of the opponent's behavior, up to some level of confidence, and then apply dynamic programming to compute an optimal solution given the approximate opponent model.

An evolutionary method applied to this problem would directly search the space of possible policies for one with a high probability of winning against the opponent. Here, a policy is a rule that tells the player what move to make for every state of the game; every possible configuration of Xs and Os on the three-by-three board. For each policy considered, an estimate of its winning probability would be obtained by playing some number of games against the opponent. This evaluation would then direct which policy or policies were considered next. A typical evolutionary method would hill-climb in policy space, successively generating and evaluating policies to obtain incremental improvements. Or, perhaps, a genetic-style algorithm could be used that would maintain and evaluate a population of policies. Literally hundreds of different optimization methods could be applied.

Here is how the tic-tac-toe problem would be approached with a method making use of a value function. First, we would set up a table of numbers, one for each possible state of the game. Each number will be the latest estimate of the probability of our winning from that state. We treat this estimate as the state's value, and the whole table is the learned value function. State A has higher value than state B, or is considered "better" than state B, if the current estimate of the probability of our winning from A is higher than it is from B. Assuming we always play Xs, then for all states with three Xs in a row the probability of winning is 1, because we have already won. Similarly, for all states with three Os in a row, or that are filled up, the correct probability is 0, as we cannot win from them. We set the initial values of all the other states to 0.5, representing a guess that we have a 50% chance of winning.

We then play many games against the opponent. To select our moves, we examine the states that would result from each of our possible moves (one for each blank space on the board) and look up their current values in the table. Most of the time we move greedily, selecting the move that leads to the state with greatest value, that is, with the highest estimated probability of winning. Occasionally, however, we select randomly from among the other moves instead. These are called exploratory moves because they cause us to experience states that we might otherwise never see. This can be diagrammatically explained as below:



While we are playing, we change the values of the states in which we find ourselves during the game. We attempt to make them more accurate estimates of the probabilities of winning. To do this, we “back up” the value of the state after each greedy move to the state before the move, as suggested by the arrows in Figure. the current value of the earlier state is updated to be closer to the value of the later state. This can be done by moving the earlier state’s value a fraction of the way toward the value of the later state. If we let St denote the state before the greedy move, and St+1 the state after the move, then the update to the estimated value of St, denoted V (St), can be written as

$$V(S_t) \leftarrow V(S_t) + \alpha\Big[V(S_{t+1}) - V(S_t)\Big].$$

This simple example illustrates some of the key features of reinforcement learning methods. First, there is the emphasis on learning while interacting with an environment, in this case with an opponent player. Second, there is a clear goal, and correct behavior requires planning or foresight that takes into account delayed effects of one’s choices. For example, the simple reinforcement learning player would learn to set up multi-move traps for a shortsighted opponent. It is a striking feature of the reinforcement learning solution that it can achieve the effects of planning and lookahead without using a model of the opponent and without conducting an explicit search over possible sequences of future states and actions.

Two important learning models in Reinforcement Learning:

- Markov Decision Process:
  Markov property states that “The future is independent of the past given the present”

  The mathematical approach for mapping a solution in reinforcement Learning is reconsidered as a Markov Decision Process or (MDP).
- Q-Learning:
  Q learning is a value-based method of supplying information to inform which action an agent should take.

Applications of Reinforcement Learning:

Traffic light control
Robotics
Web System Configuration
Chemistry
Personalized Recommendations
Bidding and advertising
Games: Atari games
Deep learning

Conclusion:

Reinforcement learning is a computational approach to understanding and automating goal-directed learning and decision making. It is distinguished from other computational approaches by its emphasis on learning by an agent from direct interaction with its environment, without requiring exemplary supervision or complete models of the environment.

Reinforcement learning uses the formal framework of Markov decision processes to define the interaction between a learning agent and its environment in terms of states, actions, and rewards. This framework is intended to be a simple way of representing essential features of the artificial intelligence problem. These features include a sense of cause and effect, a sense of uncertainty and nondeterminism, and the existence of explicit goals.

The concepts of value and value function are key to most of the reinforcement learning methods that we consider in this book. We take the position that value functions are important for efficient search in the space of policies. The use of value functions distinguishes reinforcement learning methods from evolutionary methods that search directly in policy space guided by evaluations of entire policies.

References:

Source 1: http://incompleteideas.net/book/RLbook2018.pdf

Source 2: https://www.youtube.com/watch?v=2pWv7GOvuf0&t=3284s

Source 3: https://towardsdatascience.com/applications-of-reinforcement-learning-in-real-world-1a94955bcd12

Source 4: The very basics of Reinforcement Learning - Becoming Human: Artificial Intelligence Magazine