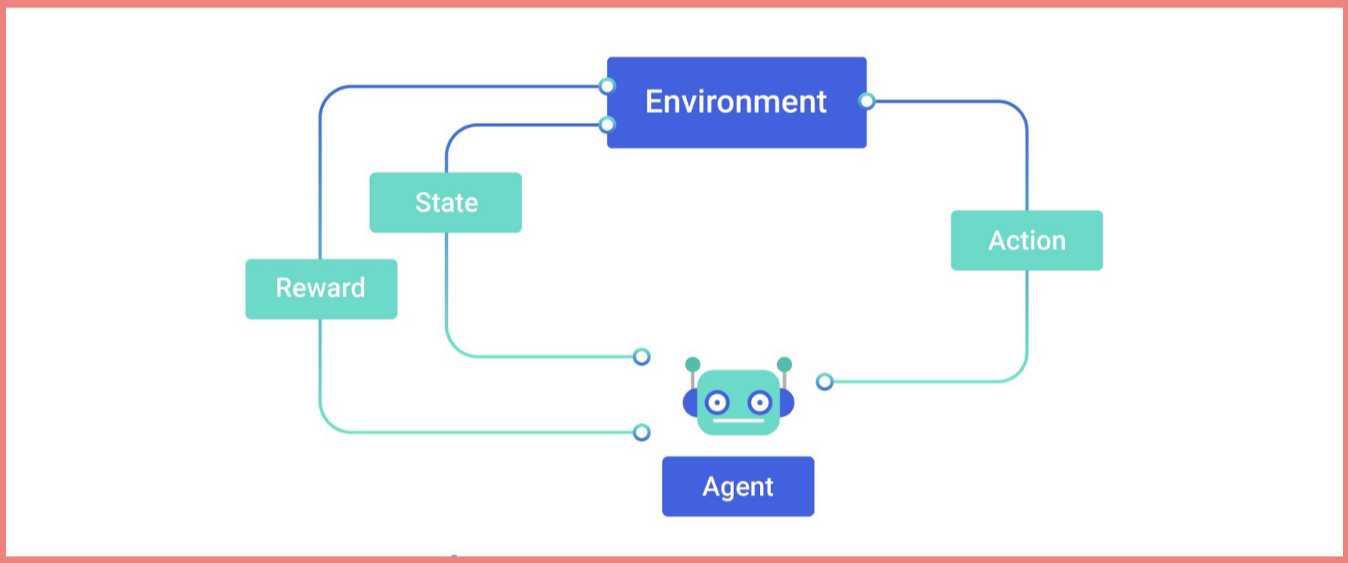
REINFORCEMENT LEARNING

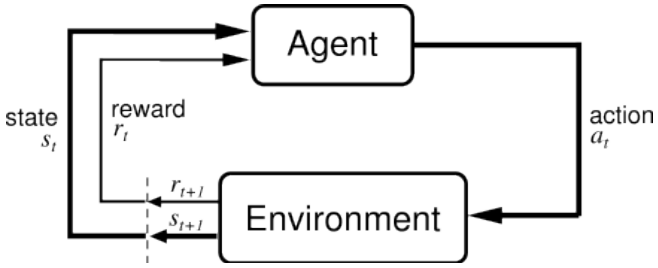


By Jonas Hellevang

Prof. ANTONIO RODRIGUEZ ARMAS

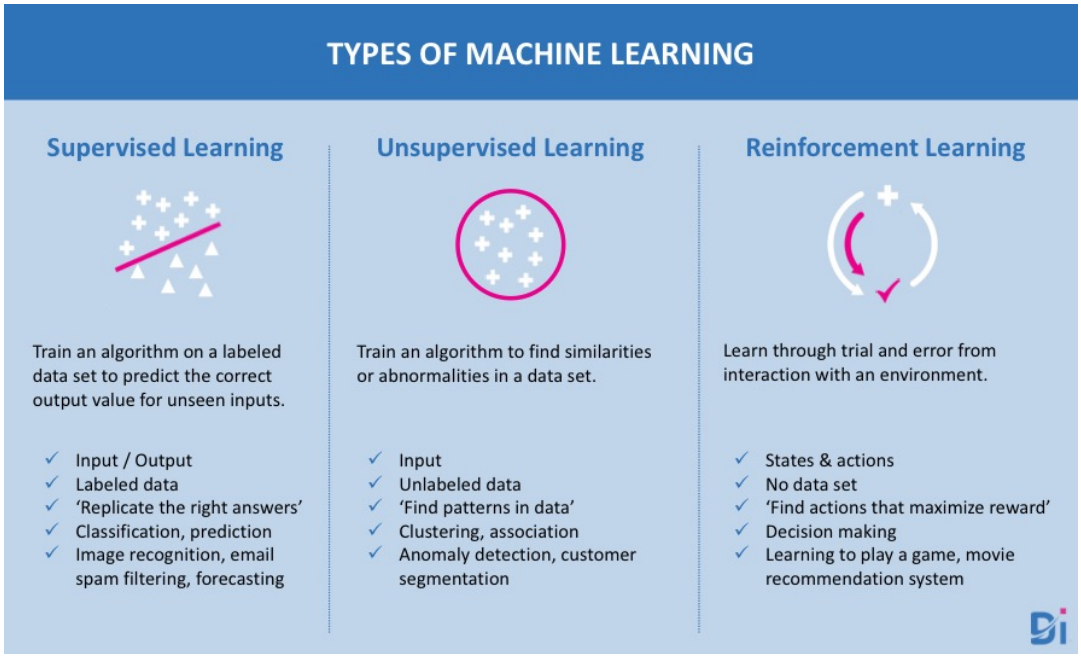
1. What is Reinforcement Learning and how is it different from other methods like Supervised or Unsupervised Learning?

There are many things fascinating about reinforcement learning, but what really distinguish it from other methods is the "delayed reward" and "trial and error". The reinforcement learning system has many elements to make the whole machinery work: a problem, an environment, an agent, a policy, a reward, states, actions and a value state. An agent, which can be a player in a chess game or a robot lawnmower your lawn, is acting in an environment. Let's take the example of the robot lawn mower. The robot is trying to find the recharge station and will for this get a reward when he does this. The environment is the lawn, the agent is of course the lawn mower, states can be battery level and location compared to the recharge station, it has limited actions to left, right, straight and backwards. Operating like this, the agent will learn what is the optimal route and how to get the highest reward possible.



Reinforcement learning has been especially important the last 5 years with companies such as Tesla, Amazon, Facebook and Google in the lead. Although it is great to know what companies use this type of machine learning technique, it is important to know that it has been around for a long time, but only in the recent years it has gotten more focus and big advancements. As in the example with the lawn mower, reinforcement learning uses live data to find the best strategy to achieve a goal, for example to best land on the moon, or race a car, like we have seen in class.

The easiest and most efficient way to compare supervised- and unsupervised learning with reinforcement learning is through this illustration:



2. What are Markov Decision Process (MDP) and Bellman's equation?

Before we go into the topic of MDP, we need to have a clear picture of what Markov Chains are. Simply explained it is a representation where you can measure probabilities in a series, which means that you are moving through different steps. This is usually represented in a transition matrix. Chess is a good example of this, as when you make one move, you now have a defined number of probabilities to end up in a position k steps ahead. MDP simply adds an agent and a decision making process to Markov Chains. The below top three steps are already components within the Markov Chain, and number 4 and 5 illustrates the set of possible actions and reward function in the Markov Decision Process:

- Markov Chain components:
  1. Set of possible States: $S = \{s_0, s_1, \ldots, s_m\}$
  2. Initial distribution (the likelihood of starting at a particular state)
  3. Transition matrix: $\begin{bmatrix} \Pr(s_0|s_0) & \Pr(s_1|s_0) & \cdots & \Pr(s_m|s_0) \\ \vdots & \vdots & \ddots & \vdots \\ \Pr(s_0|s_m) & \Pr(s_1|s_m) & \vdots & \Pr(s_m|s_m) \end{bmatrix}$
- New components for MDP:
  4. Set of possible Actions: $A = \{a_0, a_1, \ldots, a_m\}$
  5. Reward Function: $R(s)$

As you can see in number 3 above, the transition matrix of a Markov Chain is illustrated. However, this changes as we introduce the new factors in an updated transition matrix, where the probability of every combination is included:

$$T(s', r|s, a) = \begin{bmatrix} \Pr(s_0, r_0|s_0, a_0) & \Pr(s_1, r_0|s_0, a_0) & \cdots & \Pr(s_m, r_n|s_0, a_0) \\ \Pr(s_0, r_0|s_1, a_0) & \Pr(s_1, r_0|s_1, a_0) & \cdots & \Pr(s_m, r_n|s_1, a_0) \\ \Pr(s_0, r_0|s_0, a_1) & \Pr(s_1, r_0|s_0, a_1) & \cdots & \Pr(s_m, r_n|s_0, a_1) \\ \Pr(s_0, r_0|s_1, a_1) & \Pr(s_1, r_0|s_1, a_1) & \cdots & \Pr(s_m, r_n|s_1, a_1) \\ \vdots & \vdots & \ddots & \vdots \\ \Pr(s_0, r_0|s_{m-1}, a_{i-1}) & \Pr(s_1, r_0|s_{m-1}, a_{i-1}) & \vdots & \Pr(s_m, r_n|s_{m-1}, a_{i-1}) \\ \vdots & \vdots & \ddots & \vdots \\ \Pr(s_0, r_0|s_m, a_i) & \Pr(s_1, r_0|s_m, a_i) & \cdots & \Pr(s_m, r_n|s_m, a_i) \end{bmatrix}$$

Bellman Equation, however, is a way to connect immediate rewards with future rewards. The future reward is known as the return, and it is equal the total sum of all future discounted rewards. To make this a reality, it uses the discounted factor, because a future reward may have higher uncertainty, does not provide any immediate benefits, don't have to worry about infinite loops and provides mathematical convenience. In general, a high reward right now will benefit more than an uncertain reward *k* steps into the future.

3. What is a Dynamic Programming (DP) method and how is it different from MDP methods?

DP is one of many methods to use algorithms for finding the optimal policy for a model. The only downside is really that you need to have a fully complete and accurate model of the environment (on-policy), which in contrast to an off-policy method would not care and use trial and error. MDP uses "magical" utility (state values), but with DP we solve the problem by iterating through *n* numbers of bellman equations in a process called value iteration algorithm. When we iterate through this process, we use random values that are usually zero, calculate the utility of a bellman equation based on the model before, and assign this to the new state. This continues and we need a stop criterion, where in reinforcement learning there is no measures of accuracy, we can here see when there are no more utility changes, we stop the model, something we call an equilibrium.

4. What is a Multi-armed Bandit problem, and what are example applications of it in real-life?

A multi-armed bandit problem is a typical reinforcement learning problem where you allow your agent to choose from different actions, trying to achieve the maximum reward. A typical example for this is the slot machine you find at casinos, where each individual slot machine has their own reward associated with it for each pull on the handle. The job of a multi-armed bandit is to find the biggest reward from *n* pulls in the handle of *k* slot machines. Another typical application is content recommendation at Netflix. If you saw a movie, you will be recommended movies based on that one. Other examples are stock market where you want to maximize your return, find the best treatment for a disease while minimizing the loss, advertising based on the visitor itself and not some static factor always showing the same commercial to every page-visitor.

5. Explain 2 algorithms for solving a Multi-Armed Bandit problem

There are many algorithms to explore, but a quick Google search seems to favor ϵ-greedy and Upper-Confidence-Bound, so let's explore these a little further:

ϵ-greedy

ϵ-greedy is an algorithm for balancing exploration and exploitation, where we explore 10 % of the time, while we exploit 90 % of the time. What does this mean in the example of slot machines? The greedy accident action selection choses the slot with the maximum value function estimated by the agent. ϵ-greedy then has a probability of epsilon not to choose the maximum utility slot machine and choose another slot machine depressed with a uniform probability or the uniform distribution. However, explained in the most comprehensive way: ϵ-greedy randomly selects another slot machine when it decides not to take the greedy action.

Upper-Confidence-Bound (UCB)

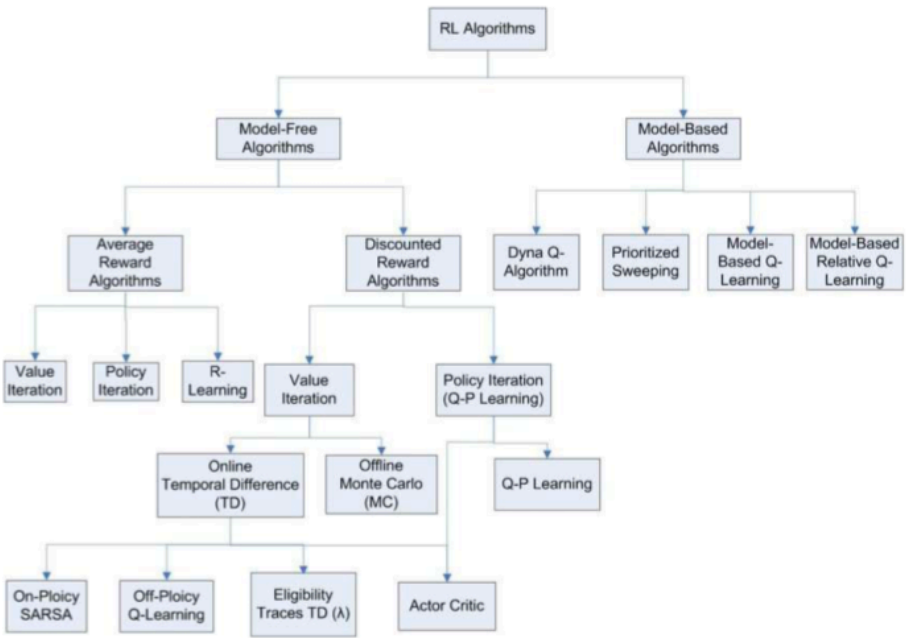
Upper-Confidence-Bound is yet another multi armed bandit technique for balancing exploration and exploitation. Actually, it is a family of algorithms. In comparison to the ϵ-greedy algorithm, UCBs favors exploration of actions with the potential of giving them a high reward. UCBs solves the problem with the greedy algorithm where you don't really know if it is the best action to choose, so there is a lot of uncertainty about the non-explored actions. The algorithm is for this reason the most widely used, because the less we know about an arm of actions, the more important it is to explore it.

6. What is the difference between Model-based and Model-free RL methods? Describe a method for each

In model-based reinforcement learning algorithms the environment is known to the agent, while in the model-free reinforcement learning algorithms the environment is not known to the agent. Model-based algorithms utilize reward functions to progressively improve the model, on the other hand model-free algorithms disregard the normal reward functions and estimate their own value functions for an unknown Markov decision process, based on experience. Examples of model-free algorithms are Monte-Carlo Learning, Temporal-Difference Learning, and Q-Learning. Examples of model-based algorithms are Model-Based Q-Learning, Prioritized Sweeping and Dyna Q-Algorithm.

Q-learning: A key idea behind Q-learning is that an agent learns about how good actions are under perfect behavior without having to behave perfectly. An agent tries different actions in different states and stores the results in a Q-table. The iterating process makes recorded values in the table more precise.

Dyna Q-Algorithm: Dyna-Q algorithm combines model learning and direct reinforcement learning, where it integrates planning in one-step tabular Q-planning, and learning in one-step tabular Q-learning. It can combine both real and simulated experience in a way that it makes good models for financial trading.



7. What is the difference between On-policy and Off-policy RL methods? Describe a method for each

I touched based on this in question 3, so I will repeat some of the information I wrote there as well as introducing new information: In on-policy methods you need to have a fully complete and accurate model of the environment, which in contrast to an off-policy method would not care and use trial and error. In other words, on-policy learning is the most stable method, where you learn as you go. This method is also less sample efficient than off-policy learning. Off-policy reinforcement learning methods on the other hand is less stable, where we learn by observing. An example of on-policy reinforcement learning is SARSA, and an example for off-policy reinforcement learning is Meta Q-learning.

8. What are the advantages and disadvantages of Monte Carlo (MC) methods vs Temporal Difference (TD) methods?

Temporal Difference Advantages:

- Can learn before the final outcome: Can learn online after every step, as shown in illustration below.
- Can learn without the final outcome: Can learn from incomplete sequences, and works in continuing (non-terminating) environments
- Usually more efficient than Monte Carlo

Monte Carlo Advantages:

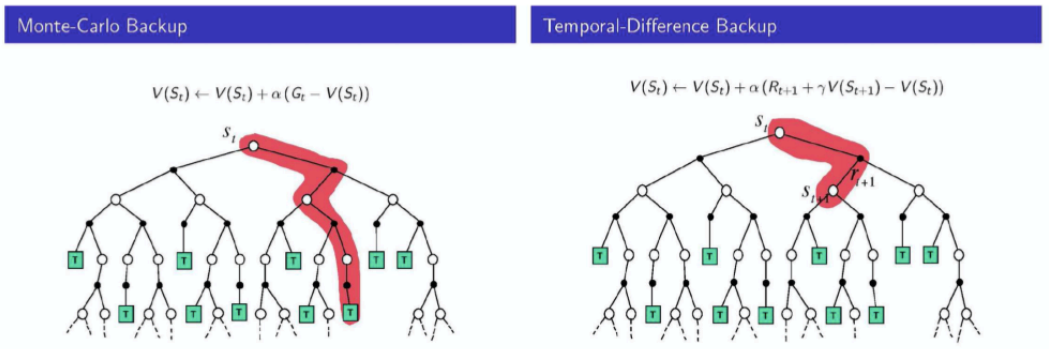
- Has high variance and zero bias which makes it very stable on paper
- It has good convergence properties
- Not very sensitive to the initial value set
- It is very simple and easy to understand

Temporal Difference Disadvantages:

- Has low variance and some bias, which make it sometimes not converge mathematically.
- It is more sensitive to the initial value set

Monte Carlo Disadvantages:

- Can only learn from complete sequences
- Only works for episodic (terminating) environments
- You must wait until the end of an episode before the return is known



9. What is SARSA and how is it different from Q-learning?

SARSA (State-Action-Reward-State-Action) resembles Q-learning in many ways, but the key difference is an on-policy algorithm, where the agent has a fully complete overview of the environment. Q-learning has no constraints over the next action and is not greedy, where in contrast to SARSA it fully well knows based on its current policy, as well as it is greedy. In SARSA, the agent selects one action from the poll of options in the policy and moves one step forward. The agent then observes the reward, the new state and the associated action, before it updates the action-value function using the update rule.

10. Describe the following three RL algorithms , and how these work in high level:

Principal Policy Optimization (PPO)

PPO is a policy gradient method which learns directly from what the agent experiences in the environment. It only uses the collective experience once, hence it is less sample efficient. When training the agent, it runs x amounts of epochs of gradient decent over a sample of mini batches, forcing the agent to obtain high rewards. PPO introduced a new objective function called *Clipped surrogate objective function* that will constraint the policy change in a small range using a clip. This is introduced to avoid errors that often occurs; too high variance in the training or too slow training process.

Meta-Q-learning (MQL)

Meta Q-learning is an off-policy reinforcement learning algorithm used for meta reinforcement learning. It is still very little information to be found about the topic online, but by reading the paper by Chaudhari, Pratik, Smola & Soatto (2020) I was able to gather these three takeaways: (1) It simplifies the meta-training objective as a multi-task optimization across partially related tasks. (2) It uses plain Q-learning algorithms enriched with a contextual variable that can be competitive with SOTA meta-RL alternatives. (3) It leverages information in the meta-training replay buffer which can improve policy adaptation on new tasks.

Asynchronous Advantage Actor Critic (A3C)

A3C uses a neural network to get two outputs, a policy net and a value net. The value net has a single number as an output that values the current state for the value function, while the policy net has a softmax output of probabilities for taking each action. The predicted values is subtracted from the actual returns of the actions to obtain the advantage and scale the policy given.

References

Sutton, Richard S., and Andrew G. Barto. 2018. Reinforcement Learning. 2nd ed. Cambridge, MA.

2020. Image. https://schoolofdisruption.com/wp-content/uploads/2018/07/Supervised-Unsupervised-Reinforcement-learning.jpg.

"Asynchronous Methods For Deep Reinforcement Learning". 2016. https://arxiv.org/pdf/1602.01783.pdf.

"Introduction To Various Reinforcement Learning Algorithms. Part I (Q-Learning, SARSA, DQN, DDPG)". 2020. Medium. https://towardsdatascience.com/introduction-to-various-reinforcement-learning-algorithms-i-q-learning-sarsa-dqn-ddpg-72a5e0cb6287.

"Multi Armed Bandit Problem & Its Implementation In Python". 2020. Analytics Vidhya. https://www.analyticsvidhya.com/blog/2018/09/reinforcement-multi-armed-bandit-scratch-python/.

"Multi-Armed Bandits: UCB Algorithm". 2020. Medium. https://towardsdatascience.com/multi-armed-bandits-ucb-algorithm-fa7861417d8c.

"Online Planning Agent: Dyna-Q Algorithm And Dyna Maze Example (Sutton And Barto 2016)". 2020. Medium. https://medium.com/@ranko.mosic/online-planning-agent-dyna-q-algorithm-and-dyna-maze-example-sutton-and-barto-2016-7ad84a6dc52b.

"Reinforcement Learning: Temporal-Difference, SARSA, Q-Learning & Expected SARSA On Python". 2020. Medium. https://towardsdatascience.com/reinforcement-learning-temporal-difference-sarsa-q-learning-expected-sarsa-on-python-9fecfda7467e.

"Simple Reinforcement Learning With Tensorflow Part 8: Asynchronous Actor-Critic Agents (A3C)". 2020. Medium. https://medium.com/emergent-future/simple-reinforcement-learning-with-tensorflow-part-8-asynchronous-actor-critic-agents-a3c-c88f72a5e9f2.

Smola, Alexander, Stefano Soatto, Pratik Chaudhari, and Rasool Fakoor. 2020. "M E TA - Q - LE A R N I N G". https://openreview.net/pdf?id=SJeD3CEFPH.