Reinforcement Learning. Main concept. Value, Policy and Model based RL

Course project

Applied Artificial Intelligence

Petar Bechev

Aleksi Stoyanov

Mihail Nenchev

Table of Contents

[Reinforcement Learning – Main Concept 1](#_Toc134460989)

[Policy based Reinforcement Learning. How does the agent make decisions? 3](#_Toc134460990)

[Deterministic policies 3](#_Toc134460991)

[Stochastic policies 3](#_Toc134460992)

[Softmax parametrization 3](#_Toc134460993)

[Gaussian parametrization 4](#_Toc134460994)

[REINFORCE algorithm 5](#_Toc134460995)

[Value based reinforcement learning 5](#_Toc134460996)

[Q-Learning algorithm 6](#_Toc134460997)

[Model based reinforcement learning 10](#_Toc134460998)

[Models 10](#_Toc134460999)

[Dyna-Q algorithm 11](#_Toc134461000)

[Sources 13](#_Toc134461001)

# Reinforcement Learning – Main Concept

Reinforcement Learning (RL) is a type of machine learning that involves an agent learning to make decisions through interactions with an environment. In RL, the agent receives feedback in the form of rewards or penalties based on its actions, which guides its decision-making process. The goal of the agent is to learn a policy that maps states to actions, maximizing its cumulative reward over time.

To understand the main concepts of Reinforcement Learning, it's helpful to consider the components of an RL problem:

1. Environment

The environment is the world in which the agent operates. It is defined by a set of states, actions, and rewards. The agent takes actions in the environment and receives rewards or penalties based on those actions.

2. State

The state represents the current situation of the agent in the environment. It is a snapshot of the environment at a particular time. The agent selects actions based on the current state.

3. Action

An action is a decision made by the agent in response to the current state. Actions can change the state of the environment and result in rewards or penalties.

4. Reward

The reward is a signal from the environment that tells the agent how well it is performing. The agent's goal is to maximize the cumulative reward over time.

5. Policy

A policy is a mapping from states to actions. It tells the agent what action to take in each state to maximize its reward.

6. Value

The value represents the expected cumulative reward the agent will receive if it starts in a particular state or takes a particular action. It is an estimate of how good a state or action is.

With these components in mind, we can see that the main concept of Reinforcement Learning is for the agent to learn a policy that maximizes its cumulative reward over time. The agent does this by learning to estimate the value of each state or state-action pair. It then uses this estimate to select the action with the highest value in each state.

There are different types of RL algorithms that can be used to learn a policy. Value-Based RL algorithms, such as Q-Learning, learn the value of each state or state-action pair and use this to select the action with the highest value. Policy-Based RL algorithms, such as the REINFORCE algorithm, learn the policy directly, without learning the value function. Model-Based RL algorithms, such as Dyna-Q, learn a model of the environment and use this to plan the agent's actions.

Overall, Reinforcement Learning is a powerful technique for learning to make decisions in complex environments. The choice of RL method depends on the specifics of the problem at hand, including the size and complexity of the state and action spaces, the availability of a good model of the environment, and the desired level of sample efficiency.

# Policy based Reinforcement Learning. How does the agent make decisions?

In Reinforcement Learning, the agent makes decisions by following a policy. A policy is a mapping from states to actions, and it tells the agent what action to take in each state to maximize its cumulative reward over time.

There are two main types of policies in Reinforcement Learning: deterministic policies and stochastic policies.

## Deterministic policies

A deterministic policy maps each state to a single action. For example, if the agent is in certain state, a deterministic policy might specify that it should always take single specific action. Such policies can be useful when the agent needs to take precise actions, and there is little uncertainty in the environment. However, they can be limiting in environments with multiple optimal actions or with noisy rewards.

## Stochastic policies

In contrast, a stochastic policy maps each state to a probability distribution over actions. For example, if the agent is in state *s1*, a stochastic policy might specify that it should take action *a1* with probability 0.8 and action a2 with probability 0.2. Stochastic policies can be useful in environments where there is uncertainty or randomness, or when multiple actions are equally good.

### Softmax parametrization

Stochastic policies can be parameterized in different ways. One common parameterization is the *softmax* policy, where the probability of selecting each action is proportional to the exponentiated value of its expected reward. In other words – *softmax* transforms a vector of numbers into a vector of relative “probabilities”.

A picture containing text, screenshot, font, line

Description automatically generated

### Gaussian parametrization

Another common parameterization is the Gaussian policy, where the probability of selecting each action is modeled as a Gaussian distribution with mean and variance parameters. Gaussian policy is used in the case of continuous action space, for example when driving a car and you steer the wheels or press on the gas pedal, these are continuous actions because these are not few actions that you do since you you can (in theory) decide the rotation degree or the flow amount of gas.

A picture containing text, font, screenshot, algebra

Description automatically generated

The choice between deterministic and stochastic policies depends on the specifics of the problem at hand. In general, deterministic policies can be useful in simple, low-dimensional environments with little uncertainty. Stochastic policies can be useful in more complex, high-dimensional environments where there is uncertainty or multiple optimal actions. The choice of policy can also depend on the agent's computational resources, as stochastic policies can require more computation to evaluate than deterministic policies.

To determine the optimal policy, the agent needs to explore the environment and collect information about the rewards it receives for different actions in different states. This is typically done using a trial-and-error approach, where the agent takes actions in the environment and observes the rewards it receives.

## Popular Algorithms

Policy-based reinforcement learning algorithms learn a policy directly, without explicitly estimating the value of each state or state-action pair. Here are some examples of policy-based reinforcement learning algorithms:

1. REINFORCE:
   * REINFORCE is a policy-based reinforcement learning algorithm that learns a stochastic policy by updating the policy parameters in the direction of the gradient of the expected cumulative reward.
2. Proximal Policy Optimization (PPO):
   * PPO is a policy-based reinforcement learning algorithm that learns a stochastic policy by maximizing a clipped surrogate objective function that approximates the expected cumulative reward.
3. Trust Region Policy Optimization (TRPO):
   * TRPO is a policy-based reinforcement learning algorithm that learns a stochastic policy by optimizing a surrogate objective function subject to a constraint on the maximum change in the policy parameters.
4. Deterministic Policy Gradient (DPG):
   * DPG is a policy-based reinforcement learning algorithm that learns a deterministic policy by computing the gradient of the expected cumulative reward with respect to the policy parameters.
5. Actor-Critic:
   * Actor-Critic is a family of policy-based reinforcement learning algorithms that combine a policy network (the actor) with a value function network (the critic). The critic estimates the value of the current state, and the actor updates the policy parameters to increase the expected cumulative reward based on the estimated value.

## REINFORCE algorithm

REINFORCE is **a Monte Carlo variant of a policy gradient algorithm in reinforcement learning**. The agent collects samples of an episode using its current policy, and uses it to update the policy parameter . Since one full trajectory must be completed to construct a sample space, it is updated as an off-policy algorithm.

In this approach, the agent learns a probability distribution over actions for each state, and selects actions by sampling from this distribution. The agent collects a trajectory τ of one episode using its current policy, and uses it to update the policy parameter. Since one full trajectory must be completed to construct a sample space, REINFORCE is updated in an off-policy way.

Here is the pseudo code for REINFORCE :

A picture containing text, font, screenshot, white

Description automatically generated

So, the flow of the algorithm is:

1. Perform a trajectory roll-out using the current policy
2. Store log probabilities (of policy) and reward values at each step
3. Calculate discounted cumulative future reward at each step
4. Compute policy gradient and update policy parameter
5. Repeat 1–4

# Value based reinforcement learning

Value-based reinforcement learning algorithms learn the value of each state or state-action pair, and use these values to make decisions.

Some of the most popular algorithms are :

1. Q-Learning:
   * Q-Learning is a widely used value-based reinforcement learning algorithm. It learns the value of each state-action pair by updating a Q-table based on the rewards it receives for different actions in different states. The Q-table is a matrix that stores the expected value of each state-action pair.
2. Deep Q-Networks (DQN):
   * DQN is an extension of Q-learning that uses deep neural networks to approximate the Q-values instead of a Q-table. DQN has been successfully applied to a variety of tasks, including playing Atari games and controlling robots.
3. SARSA:
   * SARSA is another value-based reinforcement learning algorithm. It learns the value of each state-action pair by updating a table of state-action values based on the rewards it receives for different actions in different states, and the next action selected by the policy. The name SARSA comes from the fact that the algorithm updates the value of the current state-action pair based on the state, action, reward, next state, and next action.
4. Expected SARSA:
   * Expected SARSA is a variant of SARSA that estimates the expected value of the next state-action pair using a softmax function over the Q-values of the next state.
5. TD-Learning:
   * TD-Learning is a value-based reinforcement learning algorithm that uses temporal difference learning to estimate the value of each state or state-action pair. It updates the value of each state or state-action pair based on the difference between the predicted value and the actual reward received.

These are just a few examples of value-based reinforcement learning algorithms. Each algorithm has its own strengths and weaknesses, and the choice of algorithm depends on the specific problem at hand.

## Q-Learning algorithm

One common algorithm for learning the optimal policy is the Q-Learning algorithm. In Q-Learning, the agent learns the expected reward, or value, of each state-action pair. It does this by updating a Q-table based on the rewards it receives for different actions in different states. The Q-table is a matrix that stores the expected value of each state-action pair.

Once the agent has learned the values of each state-action pair, it can use this information to select the optimal action in each state. This can be done using an epsilon-greedy strategy, where the agent selects the action with the highest value with probability *1-epsilon*, and selects a random action with probability *epsilon*.

#### Numerical example

Suppose we have 5 rooms in a building connected by doors as shown in the figure below.  We’ll number each room from 0 through 4.  The outside of the building can be thought of as one big room (5).  Notice that doors 1 and 4 lead into the building from room 5 (outside).

A picture containing diagram, sketch, plan, technical drawing

Description automatically generated

We can represent the rooms on a graph, each room as a node, and each door as a link. We can also assign rewards to the graph representation. For this example, we’d like to put an agent in any room, and from that room, go outside the building - this will be our target room. In other words, the goal room is number 5. To set this room as a goal, we’ll associate a reward value to each door (i.e. link between nodes). The doors that lead immediately to the goal have an instant reward of 100.  Other doors not directly connected to the target room have zero rewards. Because doors are two-way ( 0 leads to 4, and 4 leads back to 0 ), two arrows are assigned to each room. Each arrow contains an instant reward value, as shown below:

A picture containing diagram, line, circle, font

Description automatically generated

In Q-learning, the goal is to reach the state with the highest reward, so that if the agent arrives at the goal, it will remain there forever. This type of goal is called an “absorbing goal”. We’ll call each room, including outside, a “state”, and the agent’s movement from one room to another will be an “action”.  In our diagram, a “state” is depicted as a node, while “action” is represented by the arrows.

Suppose the agent is in state 2.  From state 2, it can go to state 3 because state 2 is connected to 3.  From state 2, however, the agent cannot directly go to state 1 because there is no direct door connecting rooms 1 and 2 (thus, no arrows).  From state 3, it can go either to state 1 or 4 or back to 2 (look at all the arrows about state 3).  If the agent is in state 4, then the three possible actions are to go to states 0, 5, or 3.  If the agent is in state 1, it can go either to state 5 or 3.  From state 0, it can only go back to state 4.

We can put the state diagram and the instant reward values into the following reward table, “matrix R”.

A picture containing text, font, screenshot, diagram

Description automatically generated

The rows of matrix Q represent the current state of the agent, and the columns represent the possible actions leading to the next state (the links between the nodes).

The transition rule of Q learning is a very simple formula:

*Q(state, action) = R(state, action) + Gamma \* Max[Q(next state, all actions)]*

According to this formula, a value assigned to a specific element of matrix Q is equal to the sum of the corresponding value in matrix R and the learning parameter Gamma, multiplied by the maximum value of Q for all possible actions in the next state.

The Q-Learning algorithm goes as follows:

*1. Set the gamma parameter and environment rewards in matrix R.*

*2. Initialize matrix Q to zero.*

*3. For each episode:*

*Select a random initial state.*

*Do While the goal state hasn’t been reached.*

* *Select one among all possible actions for the current state.*
* *Using this possible action, consider going to the next state.*
* *Get maximum Q value for this next state based on all possible actions.*
* *Compute: Q(state, action) = R(state, action) + Gamma \* Max[Q(next state, all actions)]*
* *Set the next state as the current state.*

*End Do*

*End For*

#### Q-Learning Example By Hand

To understand how the Q-learning algorithm works, we’ll go through a few episodes step by step. The rest of the steps are illustrated in the source code examples.

We’ll start by setting the value of the learning parameter Gamma = 0.8, and the initial state as Room 1.

Initialize matrix Q as a zero matrix:

A picture containing text, screenshot, font, number

Description automatically generated

Look at the second row (state 1) of matrix R.  There are two possible actions for the current state 1: go to state 3, or go to state 5. By random selection, we select to go to 5 as our action.

A picture containing text, font, screenshot, diagram

Description automatically generated

Now let’s imagine what would happen if our agent were in state 5.  Look at the sixth row of the reward matrix R (i.e. state 5).  It has 3 possible actions: go to states 1, 4, or 5.

*Q****(state, action) = R(state, action) + Gamma \* Max[Q(next state, all actions)]  
Q(1, 5) = R(1, 5) + 0.8 \* Max[Q(5, 1), Q(5, 4), Q(5, 5)] = 100 + 0.8 \* 0 = 100***

Since matrix Q is still initialized to zero, Q(5, 1), Q(5, 4), Q(5, 5), are all zero.  The result of this computation for Q(1, 5) is 100 because of the instant reward from R(5, 1).

The next state, 5, now becomes the current state.  Because 5 is the goal state, we’ve finished one episode.  Our agent’s brain now contains an updated matrix Q as:

A picture containing text, screenshot, font, number

Description automatically generated

For the next episode, we start with a randomly chosen initial state and so on.

Overall, the key to making decisions in Reinforcement Learning is to learn a policy that maximizes the expected cumulative reward over time. The agent does this by exploring the environment, learning the values of each state-action pair, and selecting the optimal action in each state based on these values.

# Model based reinforcement learning

## Models

Some models produce a description of all possibilities and their probabilities; these we call distribution models. Other models produce just one of the possibilities, sampled according to the probabilities; these we call sample models.

Given a starting state and a policy, a sample model could produce an entire episode, and a distribution model could generate all possible episodes and their probabilities. In either case, we say the model is used to *simulate* the environment and produce *simulated experience*.

**Model -> Planning -> Policy**

Two basic ideas:

1. All state-space planning methods involve computing value functions as a key intermediate step toward improving the policy.
2. They compute value functions by updates or backup operations applied to simulated experience.

**Model -> Experience (Simulated) -> Values -> Policy**

Model-based reinforcement learning algorithms learn a model of the environment, such as the transition dynamics and rewards, and use this model to make decisions. Here are some examples of model-based reinforcement learning algorithms:

1. Dyna-Q:
   * Dyna-Q is a model-based reinforcement learning algorithm that learns the transition dynamics and rewards of the environment by updating a model based on observed transitions. It then uses this model to simulate future experiences and updates its value estimates based on the simulated experiences.
2. Model-based Interval Estimation Reinforcement Learning (MIERL):
   * MIERL is a model-based reinforcement learning algorithm that uses Bayesian inference to estimate the transition dynamics and rewards of the environment. It updates its estimates based on observed transitions and uses these estimates to select actions that maximize the expected cumulative reward.
3. Model Predictive Control (MPC):
   * MPC is a model-based reinforcement learning algorithm that uses a dynamic model of the environment to predict the future states and rewards of the system. It then solves an optimization problem to select the action that maximizes the expected cumulative reward over a finite time horizon.
4. Tree-based Model-based Reinforcement Learning (TB-MBRL):
   * TB-MBRL is a model-based reinforcement learning algorithm that learns a decision tree to represent the transition dynamics and rewards of the environment. It uses this tree to select the action that maximizes the expected cumulative reward.
5. Gaussian Process Model-based Reinforcement Learning (GP-MBRL):
   * GP-MBRL is a model-based reinforcement learning algorithm that uses Gaussian process regression to learn the transition dynamics and rewards of the environment. It uses the learned model to select the action that maximizes the expected cumulative reward.

## Dyna-Q algorithm

When planning is done online, while interacting with the environment, a number of interesting issues arise:

* New information gained from the interaction may change the model and thereby interact with planning.
* It may be desirable to customize the planning process in some way to the states or decisions currently under consideration, or expected in the near future.
* If decision making and model learning are both computation-intensive processes, then the available computational resources may need to be divided between them.

Dyna-Q is a simple architecture integrating the major functions needed in an online planning agent.

Within a planning agent, there are at least two roles for real experience:

1. it can be used to improve the model (to make it more accurately match the real environment). This is called model-learning.
2. It can be used to directly improve the value function and policy using different kinds of reinforcement learning methods. This is called direct reinforcement learning.

A picture containing diagram, circle, text, font

Description automatically generated

Dyna-Q includes all of the processes shown in the diagram above—planning, acting, model-learning, and direct RL—all occurring continually.

The model-learning method is also table-based and assumes the environment is deterministic. If the model is queried with a state–action pair that has been experienced before, it simply returns the last-observed next state and next reward as its prediction.  
The direct RL method is one-step tabular Q-learning.

The planning method is the random-sample one-step tabular Q-planning method.

A picture containing text, font, screenshot

Description automatically generated

Learning and planning are deeply integrated in the sense that they share almost all the same machinery, differing only in the source of their experience.

A picture containing text, screenshot, diagram, font

Description automatically generated

Tabular Dyna-Q algoritm is shown below:

A picture containing text, screenshot, font, number

Description automatically generated

# Sources

* *“Understand Q-Learning in Reinforcement Learning with a numerical example and Python implementation”* – Sefidian Academy by Amir Masoud Sefidian - Machine Learning Engineer
* *“Deriving Policy Gradients and Implementing REINFORCE”* - Chris Yoon
  + *Lecture slides from University of Toronto:* [*http://www.cs.toronto.edu/~tingwuwang/REINFORCE.pdf*](http://www.cs.toronto.edu/~tingwuwang/REINFORCE.pdf)
* *“Dyna-Q - Planning and Learning” -*  Notes on AI, Reinforcement Learning