MIS4311
Machine Learning Applications

Fall 2025

Lecture #12

Reinforcement Learning

Reinforcement learning fills the gap between supervised learning, where the algorithm is trained on the correct answers given in the target data, **and unsupervised learning**, where the algorithm can only exploit similarities in the data to cluster it.

The middle ground is where **information is provided about whether or not the answer is correct**, but not how to improve it.

The reinforcement **learner has to try out different strategies** and **see which work best.**

Search is a fundamental part of any reinforcement learner: the algorithm **searches over the state space** of possible inputs and outputs in order **to try to maximize a reward.**

Reinforcement Learning

Reinforcement learning is usually described in terms of **the interaction between some agent and its environment.**

The agent is the thing that **is learning**, and the environment is where it is learning, and what it is learning about.

RL is about learning the optimal behavior in an environment **to obtain maximum reward.**

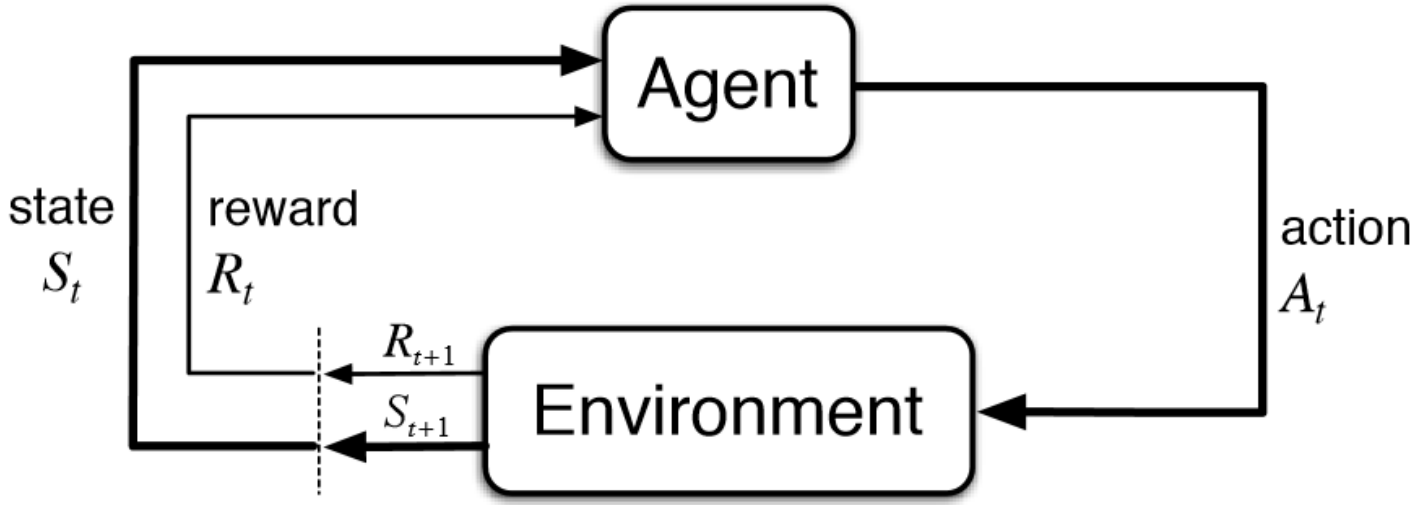
This optimal behavior is learned through interactions with the environment and observations of how it responds, similar to children exploring the world around them and learning the actions that help them achieve a goal.

In the **absence of a supervisor**, the **learner must independently discover** the sequence of actions that maximize the reward.

Reinforcement Learning

As compared to unsupervised learning, reinforcement learning is different in terms of goals. While the goal in unsupervised learning is to find similarities and differences between data points, in the case of reinforcement learning **the goal is to find a suitable action model** that would maximize the **total cumulative reward** of the agent.

The figure below illustrates the **action-reward feedback loop** of a generic RL model.



Reinforcement Learning

The main elements of an RL system are:

1. **Environment —** Physical world in which the agent operates
2. **State —** Current situation of the agent
3. **Reward —** Feedback from the environment
4. **Policy —** Method to map agent's state to actions
5. **Value —** Future reward that an agent would receive by taking an action in a particular state

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.**

Reinforcement Learning

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 Markov Decision Processes (MDP).
MDP provides a mathematical framework for **modeling decision making** in situations where outcomes are partly **random** and partly under the control of a decision maker. MDPs are useful for studying optimization problems solved via **dynamic programming**.

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**.

Reinforcement Learning

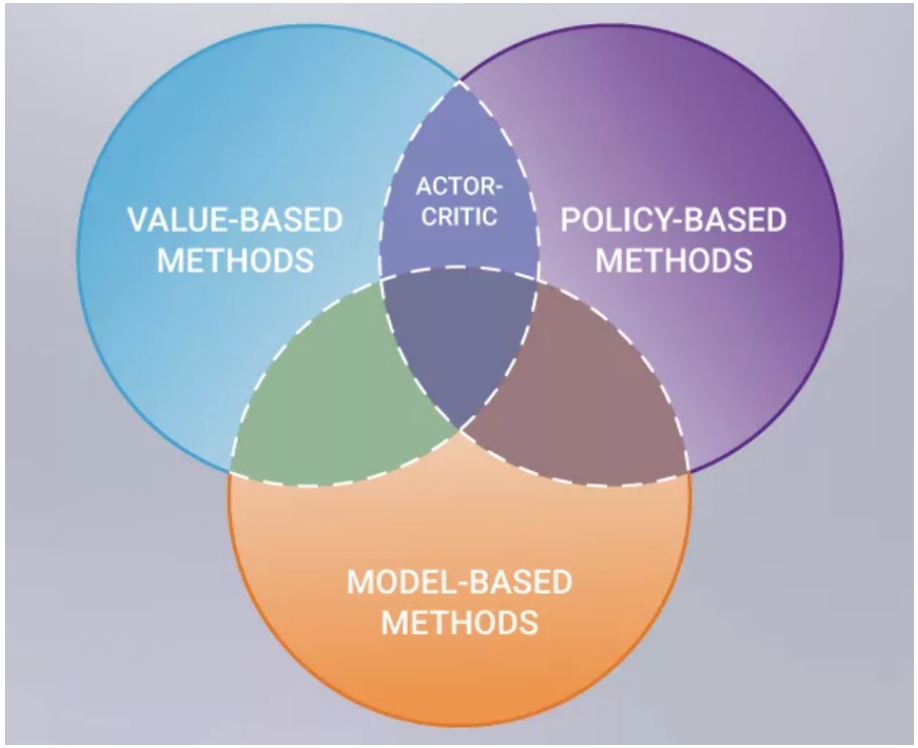
Value-based algorithms consider optimal policy to be a direct result of estimating the value function of every state accurately. Some popular value-based algorithms are **State–action–reward–state–action (SARSA) and Q-learning.**

Policy-based algorithms, 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.
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.

Reinforcement Learning

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.



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 (model-free) is a regression problem, the latter (model-based) 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.

Reinforcement Learning

Q-learner:

As the name implies, the learning target of a Q-learner is Q-values of state-action pairs. In other words, we are using Q-learner to learn $Q(s, a)$.

For any MDP, we always have

$$v(s) = r + \gamma v(s')$$

we explained the relation between values of states and values of state-action pairs, where γ represents a discount factor on future rewards. It determines how much the system values future results relative to the current state. *r* is the immediate reward after transition from *s* to *s'* with action *a*

$$v(s) = \max_a \left\{ r(s) + \gamma \left[\sum_{s'} p(s'|s,a) v(s') \right] \right\} = \max_a Q(s,a)$$

Reinforcement Learning

Assuming at a certain step of a game, we are at state *s*, we choose an action *a*, and we arrive at next state *s′*. For this particulate step, we experienced the state-action pair (*s*, *a*) and the results of this experience can be assessed by

$$Q(s,a) = r(s) + \gamma v(s') = r(s) + \gamma \max_{a'} Q(s', a')$$

Using the concept of updating and step size we introduced, we know that this assessed value is the *Target* learned in this step. In other words, from this step, we estimate that $Q(s, a)$ should be a value *somewhere* near this.

Reinforcement Learning

$$\text{NewEstimate} = \text{OldEstimate} + \text{StepSize}\,[\text{Target} - \text{OldEstimate}]$$

- *Old Estimate*: Q-value of that state-action pair before this step
$Q(s, a)[i-1]$

- *New Estimate*: Q-value of that state-action pair after this step
$Q(s, a)[i]$

- *Target*: $r(s) + \gamma \max_a Q(s', a)$

- *Step Size*: an arbitrary constant value α, also called *learning rate*

Now, we have the updating rule of Q-learning:

$$Q(s,a)_i = Q(s,a)_{i-1} + \alpha \left[r(s) + \gamma \max_{a'} Q(s',a')_{i-1} - Q(s,a)_{i-1} \right]$$

Reinforcement Learning

A Sample Application of Q-learning:

Assume we have four states in a game, from s_1 to s_4, with s_4 as the ending state. Only one action a is available for each state. For the purpose of demonstration, this game has no stochastic. Whenever we play this game, we receive the following results:

$$[s_1, a, r = -2, s_2, a, r = 4, s_3, a, r = 1, s_4 \text{ (game ends)}]$$

Let's use a discounted factor γ of 0.5.

Because only one action a is available for each state in this simple example, we have

$$Q(s, a) = r(s) + \gamma \max_{a'} Q(s', a') = r(s) + \gamma Q(s', a)$$

Reinforcement Learning

Since we experienced the series of states [s_1, s_2, s_3, s_4] in every game, we have

$$Q(s_1, a) = r(s_1) + \gamma Q(s_2, a)$$
$$Q(s_2, a) = r(s_2) + \gamma Q(s_3, a)$$
$$Q(s_3, a) = r(s_3) + \gamma Q(s_4, a)$$

Now let's see if a Q-learner can achieve these same Q-values. For now, we initialize all the Q-values as zero at beginning. We will discuss how initialization affects Q-learner later in this story.

	(s_1, a)	(s_2, a)	(s_3, a)	(s_4, a)
Q_0	0	0	0	0

We play our first game, and get the results of each state-action pair:

	(s_1, a)	(s_2, a)	(s_3, a)	(s_4, a)
Q_0	0	0	0	0
result of step	$s' = s_2, r = -2$	$s' = s_3, r = 4$	$s' = s_4, r = 1$	

Reinforcement Learning

Now we can apply our updating rule of Q-learning. For now, we use a step-size or learning rate of $\alpha = 0.3$.

	(s_1, a)	(s_2, a)	(s_3, a)	(s_4, a)
$Q(s,a)_0$	0	0	0	0
result of step	$s' = s_2, r = -2$	$s' = s_3, r = 4$	$s' = s_4, r = 1$	
$r(s)$	-2	4	1	
$Q(s',a)_0$	0	0	0	
$r(s) + \gamma Q(s',a)_0$	-2	4	1	
$\alpha[r(s) + \gamma Q(s',a)_0 - Q(s,a)_0]$	-0.6	1.2	0.3	
$Q(s,a)_1 = Q(s,a)_0 + \alpha[r(s) + \gamma Q(s',a)_0 - Q(s,a)_0]$	-0.6	1.2	0.3	

After the first game, our Q-values are updated from all zeros to [−0.6, 1.2, 0.3]. Now let's play the game again.

	(s_1, a)	(s_2, a)	(s_3, a)	(s_4, a)
$Q(s,a)_1$	-0.6	1.2	0.3	0
result of step	$s' = s_2, r = -2$	$s' = s_3, r = 4$	$s' = s_4, r = 1$	
$r(s)$	-2	4	1	
$Q(s',a)_1$	1.0	0.3	0	
$r(s) + \gamma Q(s',a)_1$	-1.4	4.15	1	
$\alpha[r(s) + \gamma Q(s',a)_1 - Q(s,a)_1]$	-0.24	0.885	0.21	
$Q(s,a)_2 = Q(s,a)_1 + \alpha[r(s) + \gamma Q(s',a)_1 - Q(s,a)_1]$	-0.84	2.085	0.51	

Reinforcement Learning

Then we play the game for the third time:

	(s_1, a)	(s_2, a)	(s_3, a)	(s_4, a)
$Q(s,a)_2$	-0.84	2.085	0.51	0
result of step	$s' = s_2, r = -2$	$s' = s_3, r = 4$	$s' = s_4, r = 1$	
$r(s)$	-2	4	1	
$Q(s',a)_2$	2.085	0.51	0	
$r(s) + \gamma Q(s',a)_2$	-0.9575	4.255	1	
$\alpha[r(s) + \gamma Q(s',a)_2 - Q(s,a)_2]$	-0.03525	0.651	0.147	
$Q(s,a)_3 = Q(s,a)_2 + \alpha[r(s) + \gamma Q(s',a)_2 - Q(s,a)_2]$	-0.87525	2.736	0.657	

Then we repeat this game for another time… until we converge a right solution.

Let's apply this procedure on python to clarify this learning algorithm.

Next Week

- Image Processing

Thank you for your participation ☺