CS294: Deep Reinforcement Learning [1]

Lecture Notes: Introduction to Policy Gradient [2]

Fall 2017[3]

[1] Course Instructors: Sergey Levine

[2] Author: InnerPeace

[3] Un-official Lecture Notes

Keyphrases: Markov Decision Process. Reinforcement Learning Algorithms.

This set of notes first introduces the definition of Markov Decision Process (MDP). Then it dives into the definition of RL problems and goals, also giving the high-level overview of the anatomy of RL which is the high-level structure shared among all the algorithms we will learn. Lastly we give a brief overview of RL algorithm types.

1 *Reward Function*

In *Imitation Learning*, we learned how to train the policy with demonstrations. But how can we achieve the result without using example data? Following the mechanism of solving typical DL problem, we need to find a way to define the task, and find out what the object function is, then we may apply SGD trick to learn the policy. In RL, we usually take the rewards $r(\mathbf{s}, \mathbf{a})$ [4] of all time-steps as the object function, which tells us which states and actions are preferred. In the case of driving, one has high reward when driving safely on the road with desire speed or low reward for being a state where one collided with another car. The crucial thing is that the reward function does not tell you how to act right now, it represents the desirability of the outcome. The the job of RL is trying to take a preferred action so that the future outcome will be more desirable which is referred as *delayed reward problem*.

[4] Some may also use the notation $c(\mathbf{x}, \mathbf{u})$ as the cost function, where $r(\mathbf{s}, \mathbf{a}) = -c(\mathbf{x}, \mathbf{u})$

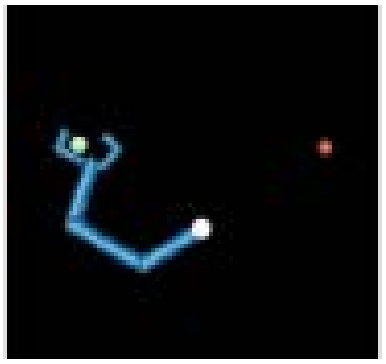


Figure 1: simple benchmark simulation task

What the reward function should be? As the case of a simple benchmark simulation task where the robotic arm trying to grab a little ball and move to a specific spot (figure 1), the reward function is straightforward in theory.

$$r(\mathbf{s}, \mathbf{a}) = \begin{cases} 1 & \text{if object at target} \\ 0 & \text{otherwise} \end{cases}$$

However, with the kinds of RL algorithms we have for now, reward function like this is very difficult to work with. The reason is that until you move the ball to the target, you don't know what you are supposed to be doing. So in practice, we usually end up with a much more complex reward function doing the seemingly simple thing, *e.g.* minimizing the distance between the gripper and object, the distance between object and target, also the size of actions in case of taking gigantic torques on the joints.

$$
\begin{aligned}
r(\mathbf{s},\mathbf{a}) = &-w_1||p_{gripper}(\mathbf{s}) - p_{object}(\mathbf{s})||^2 + \\
&-w_2||p_{object}(\mathbf{s}) - p_{target}(\mathbf{s})||^2 + \\
&-w_3||\mathbf{a}||^2
\end{aligned}
$$

The reward function reconstructing for practical use sometimes is referred as *reward shaping* [5]. For imitation learning, the reward function is the log probability of action given state. The problem is that we often do not know what the reward is in real life scenario, *e.g.* pouring water into a cup, and we will learn some methods to mitigate this problem, *i.e.*, learning the reward with Inverse Reinforcement Learning.

[5] Refer Ng's thesis for more details

$$r(\mathbf{s},\mathbf{a}) = \log p(\mathbf{a} = \pi^\star(\mathbf{s})|\mathbf{s})$$

A note about terminology of "R" word. In computer science, RL is often referred as the *problem* statement where the goal is $\max \sum_{t=1}^{T} E[r(\mathbf{s}_t,\mathbf{a}_t)]$ where $\mathbf{s}_{t+1} \sim p(\mathbf{s}_{t+1}|\mathbf{s}_t,\mathbf{a}_t)$. However, in optimal control or dynamic programming, RL represents the *method* which does not use the *model* $\max \sum_{t=1}^{T} E[r(\mathbf{s}_t,\mathbf{a}_t)]$.

2 *Markov Decision Process*

The state, action, reward and transition probability together define the *Markov Decision Process (MDP)*.

2.1 *Markov Chain*

Markov Chain is the formalism for reasoning about a particular kind of stochastic process. It's defined by state space $\mathcal{S}$ and transition operator $\mathcal{T}$ as $\mathcal{M} = \{\mathcal{S},\mathcal{T}\}$. For states $s \in \mathcal{S}$, it could be discrete or continuous, *i.e.*, a categorical variable or a vector of continuous numbers. $\mathcal{T}$ defines the probability of transferring to a new state give current state, $p(s_{t+1}|s_t)$. Let's assume that the state is discrete, and let $\mu_{t,i} = p(s_t = i)$, *i.e.*, the probability of being state i at timestep t, so $\vec{\mu}_t$ is a vector of probabilities. Let $\mathcal{T}_{i,j} = p(s_{t+1} = i|s_t = j)$, then $\vec{\mu}_{t+1} = \mathcal{T}\vec{\mu}_t$ where $\mathcal{T}$ is just a linear operator. If the state is continuous, then the $\mathcal{T}$ becomes infinitely large linear operator. The transition process demonstrates the independent property where $\vec{\mu_{t+1}}$ only depend on $\vec{\mu_t}$. And the second order or third order Markov chain means that it depends on the current and previous one, or the one before that.

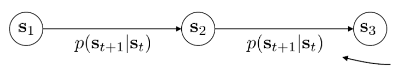

$$\begin{tikzcd} \mathbf{s}_1 \arrow[r, "p(\mathbf{s}_{t+1}|\mathbf{s}_t)"'] & \mathbf{s}_2 \arrow[r, "p(\mathbf{s}_{t+1}|\mathbf{s}_t)"'] & \mathbf{s}_3 \end{tikzcd}$$

Figure 2: Markov Chain

2.2 *Markov Decision Process*

MDP is an extension of Markov Chains to decision making setting. MDP is defined as $\mathcal{M} = \{\mathcal{S}, \mathcal{A}, \mathcal{T}, r\}$, the state space is same as Markov Chain, and the action space $\mathcal{A}$ is alike where $a \in \mathcal{A}$ could be discrete or continuous. The transition operator $\mathcal{T}$ becomes a tensor (3 dimension) which describes the probability of entering a particular state given current state and action.

$$\begin{aligned}
\mu_{t,j} &= p(s_t = j) \\
\xi_{t,k} &= p(a_t = k) \\
\mathcal{T}_{i,j,k} &= p(s_{t+1} = i | s_t = j, a_t = k) \\
\mu_{t+1,i} &= \sum_{j,k} \mathcal{T}_{i,j,k} \mu_{t,j} \xi_{t,k}
\end{aligned}$$

And lastly, the reward function $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ which maps state and action to real value numbers, and $r(s_t, a_t)$ is the reward of state and action pair at a particular timestep t.

Another generalization of Markov Chain is *Partially Observed Markov Decision Process (POMDP)* which is defined as $\mathcal{M} = \{\mathcal{S}, \mathcal{A}, \mathcal{O}, \mathcal{T}, \mathcal{E}, r\}$. We alluded observation in Imitation learning, and $\mathcal{O}$ represents the observation space, observations $o \in \mathcal{O}$ could also be discrete or continuous. $\mathcal{E}$ means the emission probability $p(o_t, s_t)$ tells how likely a certain observation o to occur given state s at timestep t.

Markov Decision Process:
(Adapted from cs231n of stanford.) At time step $t = 0$, environment samples initial state $s_0 \sim p(s_0)$.

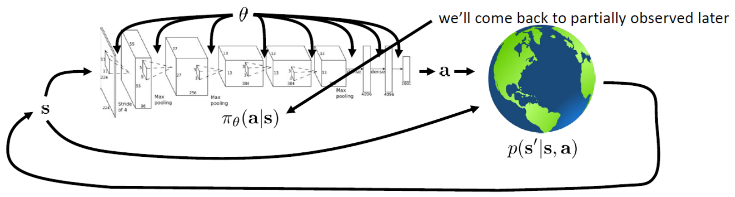
Then for $t = 0$ until done:

- Agent selects action a_t
- Environment gets reward $r(s_t, a_t)$
- Environment samples next state $s_{t+1} \sim p(s_{t+1} | s_t, a_t)$
- Agent receives reward r_t and next state s_{t+1}

3 *The Goal*

In RL problem setting, we have a initial state $\mathbf{s}_1$ and the parametrized policy $\pi_\theta(\mathbf{a}|\mathbf{s})$ [6] output the action $\mathbf{a}_1$, then with transition function $p(\mathbf{s}'|\mathbf{s}, \mathbf{a})$ which we usually do not know, we enter a new state $\mathbf{s}'$ and keep circling following the mechanism.

[6] For now we focus on the fully observed case.



θ
we'll come back to partially observed later
a
s
$\pi_\theta(\mathbf{a}|\mathbf{s})$
$p(\mathbf{s}'|\mathbf{s}, \mathbf{a})$

Figure 3: Illustration of RL problem setting

This process gives rise to the distribution over state-action sequences for finite horizon task. The distribution is given by the probability of the starting point $\mathbf{s}_1$ which is often determined by nature, times the product of all the timesteps of probability of taking an action times the probability of transitioning to the next state. We will denote the joint probability over T states and actions as $p_\theta(\tau)$ where

τ is the trajectory of states and actions, and the probability depends on θ.

$$\underbrace{p_\theta(\mathbf{s}_1, \mathbf{a}_1, \cdots, \mathbf{s}_T, \mathbf{a}_T)}_{\pi_\theta(\tau)} = p(\mathbf{s}_1) \prod_{t=1}^{T} \left(\pi_\theta(\mathbf{a}_t | \mathbf{s}_t) p(\mathbf{s}_{t+1} | \mathbf{s}_t, \mathbf{a}_t) \right)$$

Then the objective of RL problem setting could be finding the parameter θ which maximize the expectation of the overall rewards w.r.t trajectories drawn from probability distribution $p_\theta(\tau)$. And the policy gives rise to trajectory distributions.

$$\theta^\star = \arg\max_\theta E_{\tau \sim p_\theta(\tau)} \Big[\sum_t r(\mathbf{s}_t, \mathbf{a}_t) \Big]$$

The transition probability and policy at timestep t could be seen as *Markov Chain* on $(\mathbf{s}, \mathbf{a})$, where

$$p((\mathbf{s}_{t+1}, \mathbf{a}_{t+1}) | (\mathbf{s}_t, \mathbf{a}_t)) = p(\mathbf{s}_{t+1} | \mathbf{s}_t, \mathbf{a}_t) \pi_\theta(\mathbf{a}_{a+1} | \mathbf{s}_{t+1})$$

or

$$\begin{pmatrix} \mathbf{s}_{t+1} \\ \mathbf{a}_{t+1} \end{pmatrix} = \mathcal{T} \begin{pmatrix} \mathbf{s}_t \\ \mathbf{a}_t \end{pmatrix}$$

The probability *state-action* of the next timestep is given by $\mathcal{T}$ times the probability of that at current timestep. $\mathcal{T}$ is the *state-action* transition operator. We can turn MDP into the Augmented Markov Chain with knowing the policy, which is helpful to define the infinite horizon task. With turning into Markov Chain, we can rewrite the objective as.

$$\begin{aligned} \theta^\star &= \arg\max_\theta E_{\tau \sim p_\theta(\tau)} \Big[\sum_t r(\mathbf{s}_t, \mathbf{a}_t) \Big] \\ &= \arg\max_\theta \sum_{t=1}^{T} E_{(\mathbf{s}_t, \mathbf{a}_t) \sim p_\theta(\mathbf{s}_t, \mathbf{a}_t)} \left[r(\mathbf{s}_t, \mathbf{a}_t) \right] \end{aligned}$$

We take the sum outside of the expectation cause the expectation is linear, and change to calculate the expectation over reward of timestep t, and sum over them all afterwards. The $(\mathbf{s}_t, \mathbf{a}_t)$ is drawn from the state-action marginal distribution $p_\theta(\mathbf{s}_t, \mathbf{a}_t)$.

Since the *state-action* transition operator is linear, we could obtain the probability for timestep $t + k$.

$$\begin{pmatrix} \mathbf{s}_{t+k} \\ \mathbf{a}_{t+k} \end{pmatrix} = \mathcal{T}^k \begin{pmatrix} \mathbf{s}_t \\ \mathbf{a}_t \end{pmatrix}$$

When timesteps T goes to infinite, does $p(\mathbf{s}_t, \mathbf{a}_t)$ converge to a *stationary* distribution? Firstly, let's assume the stationary distribution μ exists, which stays the same before and after the transition.

$$\mu = \mathcal{T} \mu \quad \rightarrow \quad (\mathcal{T} - \mathbf{I}) \mu = 0$$

It's true when μ is eigenvector of $\mathcal{T}$ with eigenvalue 1. And if $\mathcal{T}$ is a stochastic matrix under some regularity conditions, then μ does exist. Which means the marginal distribution will finally converge the stationary distribution $\mu = p_\theta(\mathbf{s}, \mathbf{a})$. Then for the infinite case, we divide the sum with overall time-steps T since the sum over rewards of infinite time-steps also goes to infinite. At the same time, the distribution over state-action will be dominated by stationary distribution when T becomes infinite, *e.g.* after 5 million steps, it converge to stationary distribution, and since you have infinite long life time, the first 5 million steps won't matter anymore.

$$\begin{aligned}\theta^\star &= \arg\max_\theta \frac{1}{T}\sum_{t=1}^{T} E_{(\mathbf{s}_t,\mathbf{a}_t)\sim p_\theta(\mathbf{s}_t,\mathbf{a}_t)}\left[r(\mathbf{s}_t, \mathbf{a}_t)\right] \\ &= \arg\max_\theta E_{(\mathbf{s},\mathbf{a})\sim p_\theta(\mathbf{s},\mathbf{a})}\left[r(\mathbf{s}, \mathbf{a})\right]\end{aligned}$$

The objective reduce to just the expectation under stationary distribution.

In RL, we almost always care about *expectations*. In some cases, the reward is discrete, *e.g.* the task showed in figure 1 where the reward could be 1 when completing the task or 0 otherwise. With *expectation* over large amounts of trials, we can introduce the probability of success or failure which will smooth the objective so that we can applied gradient based methods to learn the policy.

4 *Anatomy*

In general, all the algorithms consist of three parts: samples generating, model fitting or return estimating, policy improving.

Samples generating is the process of sampling the trajectories with running the current policy. And it could be expensive for the real time experiments, or trivial in simulator which could be thousands times faster.

Model fitting or return estimating is computing delayed rewards from current time-step $\hat{Q} = \sum_{t'=t}^{T} \gamma^{t'-t} r_{t'}$, which is trivial and fast, for policy gradient algorithm, fitting $Q_\phi(\mathbf{s}, \mathbf{a})$ which is expensive but non-trivial to parallelize for actor-critic and Q-learning algorithm and estimating $p(\mathbf{s}'|\mathbf{s}, \mathbf{a})$ for model-based algorithms.

Policy improving could be gradient updating for PG, computing $\pi(\mathbf{s}) = \arg\max Q_\phi(\mathbf{s}, \mathbf{a})$ which is trivial for Q-learning or optimizing $\pi_\theta(\mathbf{a}|\mathbf{s})$ in model-based algorithms which is more expensive.

In order to work with stochastic systems, we use the conditional expectations where the objective is to maximize the overall rewards for the policy. We can decouple the state-action distribution to state distribution and action distribution respectively.

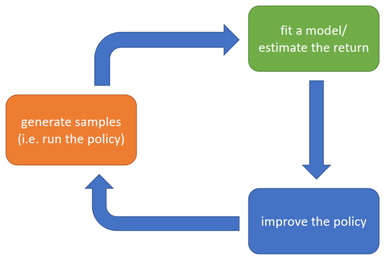


Figure 4: The anatomy of RL algorithms

$$\begin{aligned} R &= \sum_{t=1}^{T} E_{(\mathbf{s}_t,\mathbf{a}_t)\sim p_\theta(\mathbf{s}_t,\mathbf{a}_t)}\left[r(\mathbf{s}_t,\mathbf{a}_t)\right] \\ &= E_{\mathbf{s}_1\sim p(\mathbf{s}_1)}\left[E_{\mathbf{a}_1\sim\pi(\mathbf{a}_1|\mathbf{s}_1)}[Q(\mathbf{s}_1,\mathbf{a}_1)|\mathbf{s}_1]\right] \\ Q(\mathbf{s}_1,\mathbf{a}_1) &= r(\mathbf{s}_1,\mathbf{a}_1) + E_{\mathbf{s}_2\sim p(\mathbf{s}_2)}\left[E_{\mathbf{a}_2\sim\pi(\mathbf{a}_2|\mathbf{s}_2)}[r(\mathbf{s}_2,\mathbf{a}_2)+\cdots|\mathbf{s}_2]|\mathbf{s}_1,\mathbf{a}_1\right] \end{aligned}$$

Then it's easy to modify $\pi_\theta(\mathbf{a}_1|\mathbf{s}_1)$ if $Q(\mathbf{s}_1,\mathbf{a}_1)$ is known, *e.g.* set $\pi_{\theta'}(\mathbf{a}|\mathbf{s}) = 1$ if $\mathbf{a} = \arg\max_{\mathbf{a}} Q^{\pi_\theta}(\mathbf{s},\mathbf{a})$, the improved policy is at least as good as π_θ.

4.1 Q-function

We define the *Q-function* $Q^\pi(\mathbf{s}_t,\mathbf{a}_t)$ as the conditional expected cumulative reward from taking action $\mathbf{a}_t$ in state $\mathbf{s}_t$ and then following the policy, which specifies how good a state-action pair is.

$$Q^\pi(\mathbf{s}_t,\mathbf{a}_t) = \sum_{t'=t}^{T} E_{\pi_\theta}[r(\mathbf{s}_{t'},\mathbf{a}_{t'})|\mathbf{s}_t,\mathbf{a}_t]$$

4.2 Value function

Similarly, we specify how good a state is with *value function*, which is defined as the expected cumulative reward from state $\mathbf{s}_t$ conditioned on the policy π.

$$V^\pi(\mathbf{s}_t) = \sum_{t'=t}^{T} E_{\pi_\theta}[r(\mathbf{s}_{t'},\mathbf{a}_{t'})|\mathbf{s}_t]$$

Then relationship between Q-function and value function is

$$V^\pi(\mathbf{s}_t) = E_{\mathbf{a}_t\sim\pi(\mathbf{a}_t|\mathbf{s}_t)}Q^\pi(\mathbf{s}_t,\mathbf{a}_t)$$

And the RL objective could be rewrite as

$$E_{\mathbf{s}_1\sim p(\mathbf{s}_1)}[V^\pi(\mathbf{s}_1)]$$

These functions basically compress all the future rewards from the current state or action, and it helps a lot to improve the policy.

4.3 Bellman equation

We define the optimal Q-function $Q^\star$ as the maximum expected cumulative reward achievable from a given state-action pair $(\mathbf{s},\mathbf{a})$.

$$Q^{\pi^\star}(\mathbf{s}_t,\mathbf{a}_t) = \max_\pi \sum_{t'=t}^{T} E_{\pi_\theta}[r(\mathbf{s}_{t'},\mathbf{a}_{t'})|\mathbf{s}_t=\mathbf{s},\mathbf{a}_t=\mathbf{a}]$$

$Q^{\pi^\star}$ satisfies the *Bellman equation*:

$$Q^{\pi^\star}(\mathbf{s},\mathbf{a}) = E_{\mathbf{s}'\sim p(\mathbf{s}'|\mathbf{s},\mathbf{a})}\left[r + \max_{\mathbf{a}'} Q^{\pi^\star}(\mathbf{s}',\mathbf{a}')|\mathbf{s},\mathbf{a}\right]$$

The intuition is that if the optimal state-action values for the next time-step $Q^{\pi^\star}(\mathbf{s}',\mathbf{a}')$ are known, then the optimal strategy is to take the action that maximizes the expected value of $r + Q^{\pi^\star}(\mathbf{s}',\mathbf{a}')$. And the optimal policy $\pi^\star$ corresponds to taking the best action in any state as specified by $Q^{\pi^\star}$. This is the principle of Q-learning, and we will learn more about it.

Another way to take advantage of Q-functions and value functions is computing gradient to increase probability of good action $\mathbf{a}$. Since $V^\pi(\mathbf{s}_t) = E[Q^\pi(\mathbf{s}_t,\mathbf{a}_t)]$, if $Q^\pi(\mathbf{s}_t,\mathbf{a}_t) > V^\pi(\mathbf{s}_t)$, then $\mathbf{a}$ is better than average. Then we can modify policy π to increase probability of $\mathbf{a}$.

5 *Algorithms*

5.1 *Policy Gradient*

Policy Gradient directly differentiate the objective of RL problem, and improve the policy with tricks like SGD, and optimizer like *adam* also works fine. Firstly, we evaluate returns $R_\tau = \sum_t r(\mathbf{s}_t,\mathbf{a}_t)$ and update parameters of policy network with gradients of returns with respect to parameters θ.

Examples

- REINFORCE
- Natural Policy Gradient
- Trust Region Policy Optimization

5.2 *Value-based algorithms*

This kind of algorithm has no explicit policy, instead estimating the value function or Q-function of the optimal policy. After sample generating, it tries to fit $V(\mathbf{s})$ or $Q(\mathbf{s},\mathbf{a})$ and set $\pi(\mathbf{s}) = \arg\max_a Q(\mathbf{s},\mathbf{a})$ to improve the policy.

Examples

- Q-learning, DQN
- Temporal difference learning
- Fitted value iteration

5.3 *Actor-critic algorithms*

Actor-critic could be seen as the combination of PG and value-based algorithms which estimate value function or Q-function of the current policy, and use it to import policy with gradient updating. Specifically, it will fit $V(\mathbf{s})$ or $Q(\mathbf{s}, \mathbf{a})$, then evaluate the returns with V or Q instead of the cumulative rewards, and finally update policy parameters with gradients. The typical example is *Asynchronous advantage actor critic (A3C)*.

5.4 *Model-based algorithms*

The model-based RL is about estimating the transition model which is used for planning with no explicit policy or to improve the policy with gradients. The model fitting phrase is learning $p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$, then a few options for the phrase of policy improving.

Use the model to plan

- Trajectory optimization or optimal control which is primarily used in continuous spaces, both of them are essentially backpropagation to optimize over actions (no explicit policy).
- Discrete planning in discrete action spaces, *e.g.* Monte Carlo Tree Search.

Backpropagation gradients into the policy

Use the model to learn value function

- Dynamic programming
- Generate simulated experience for model-free learner, *e.g. Dyna*.

  The typical examples is *Dyna* and *Guided Policy Search*.

5.5 *Trade-offs*

In general deep learning setting, *e.g.* supervised learning, where we usually use some flavor of gradient descent and while we may argue about whether SGD with momentum or Adam is better, they are all essentially similar to each other. However, in RL, different algorithms have different trade-offs and there is no algorithm performs best in all cases. So what are the trade-offs?

Sample efficiency: it refers to how many samples we need to generate to learn a good policy. We usually divide RL algorithms into two parts: *on* policy or *off* policy w.r.t the sampling procedure.

- Off-policy means that the algorithms are able to improve the policy without generating new samples from that policy.

- On-policy means we need to generate new samples *each time* the policy is changed, even a little bit. It's could be problematic for gradient based methods, cause each update changes the policy little bit and you end up with generating so many samples during training.

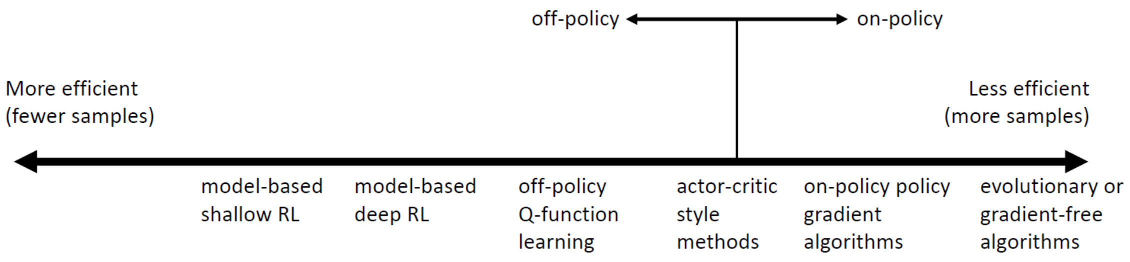

off-policy
on-policy
More efficient (fewer samples)
Less efficient (more samples)
model-based shallow RL
model-based deep RL
off-policy Q-function learning
actor-critic style methods
on-policy policy gradient algorithms
evolutionary or gradient-free algorithms

Figure 5: General Comparison of sample efficiency

The evolutionary or gradient-free algorithms take no use of the gradients of policy, treat the parameters as black-box or potentially non-differentiable and operates directly on the parameters. To be clear, less efficiency is not synonymous with bad and we may need this kinds of algorithms in some cases, *e.g.* you have fast simulators and with parallelism so that they could be the fastest in terms of the *wall clock time*. In contrast, on-policy policy gradient method is more efficient due to using the gradients. The actor-critic style methods could be off-policy or on-policy which are more efficient. The model-based algorithms perform better w.r.t sample efficiency, specifically, shallow RL is the most efficient by using different approximator like Bayesian function, but typically at the cost of introducing additional assumptions or restricted solvable classes of problems. To be clear that *wall clock time* is not the same as efficiency due to the tricks like working with simulators or parallelism, besides *wall clock time* contains the model fitting or neural network fitting process. Surprisingly, the comparison of efficiency w.r.t the wall clock time looks like the opposite of this figure 5.

Stability & ease of use: it refers to how difficult the algorithm is to choose the hyper-parameters, the learning rates or exploration parameters etc. to get it working. Convergence is the crucial question for learning algorithms, it's always gradient descent for supervised learning, for which convergence is heavily studied and almost guaranteed, but it's often not gradient descent for RL algorithms.

- Q-learning is an instance of fixed point iteration. Under certain conditions, it can converge to the optimal expected reward. Approximating expected rewards accurately is not the same as having a good policy. In other words, it's minimizing the error of prediction, not maximizing the expected reward. Sadly, many popular deep RL value fitting algorithms are not guaranteed to converge to anything in the nonlinear case.

- For model-based RL, the model is optimized for fitting the physics of the real world, which has complex implications on the performance of the corresponding police, instead of expected reward. Usually the model is trained to predict the future which will converge, but converge to a prediction model which is not equal to better policy.
- Policy gradient performs gradient descent or ascent on the true objective, but has other trade-offs.

Also, different algorithms hold different assumptions. some will assume stochastic dynamics and stochastic policies, while some would operate better on deterministic dynamics, and the same goes to continuous setting versus discrete cases, episodic setting versus infinite horizon. *e.g.* full observability assumption is generally assumed by value function fitting methods, for the partially observed case, it can mitigated by adding recurrence. Besides, the performance could be task specific, *e.g.* it's easier to represent the policy, value functions or transition model.