https://github.com/kengz/SLM-Lab

Chapter 1: INTRODUCTION: POLICY-BASED AND VALUE-BASED ALGORITHM

1/ Reinforcement Learning:

- Def: RL is concerned with solving sequential decision making problems.
- When solving question, we have objective and goals. We then take actions and get feedback from the world about how close we are to achieve the objective.
- RL problems can be expressed as a system consisting of an agent and environment.
    + An environment produces information which describes the state of the system. This is known as a state.
    + An agent interacts with the environment by observing the state and using information to select an action.
    + The environment accepts the action and transition it into the next state.
    + It then return the next state and a reward to the agent
    + A cycle of state - action - reward complete, we said that 1 time step have passed/
    + The cycle repeats until the env terminates.
- Policy = maps states to actions. An action will change an environment and affects what an agent observes and does next.
- RL problems have an object, **which is the sum of rewards received by an agent.**
- An agent's goal is to maximize the objectives by selecting the good actions.
- (s,a,r) = (state, action, rewards) => This tuple at time t is called experience

==========

2/ Reinforcement Learning as MDP:

- Transition function = MDP = math framework that models sequential decision making.
- To make environment transition function more practical, we turn it into a MDP by adding the assumption that transitionto the next state only depends on the previous state and action
- The Markov property implies that the current state and action at time step t contain sufficient info to full
- The state:
    + Is produced by an environment and observed by an agent.
    + The state that is used b transition function.
- MDP formula for RL:
    + S = set of state
    + A = set of action
    + P(st+1 | st, at) = state transition function of the environment
    + R(st, at, st+1) = the reward function of the environment.

Algorithm 1 MDP control loop

```
1: Given an env (environment) and an agent:
2: for episode = 0, . . . , MAX_EPISODE do
3:     state = env.reset()
4:     agent.reset()
5:     for t = 0, . . . , T do
6:         action = agent.act(state)
7:         state, reward = env.step(action)
8:         agent.update(action, state, reward)
9:         if env.done() then
10:            break
11:        end if
12:    end for
13: end for
```

==========

3/ Learnable functions in RL:

- Note: An agent can learn an action-producing function known as policy.
- There are properties of an environment that can be useful to an agent. 3 primary functions to learn in RL:

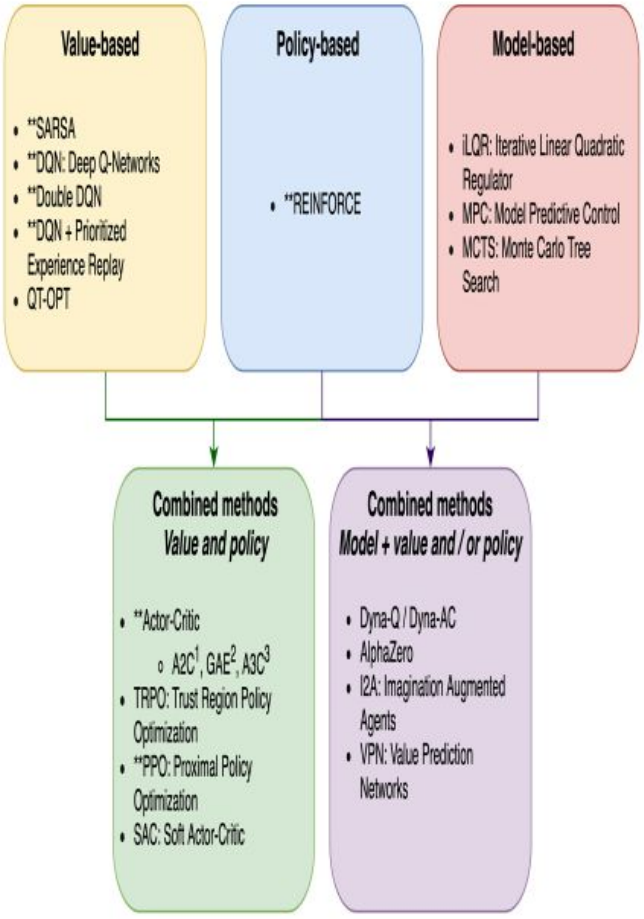
1. a policy, π, which maps state to action, $a \sim \pi(s)$
2. a value function, $V^{\pi}(s)$ or $Q^{\pi}(s, a)$, to estimate the expected return $E_{\tau}[R(\tau)]$
3. the environment model[3], $P(s'|s, a)$

    + Policy = how an agent produce actions in the environment to maximise the objective
    + A policy can be written as pi(a\s) = probability of an action given a state s.
    + A value function provide info about the objective. It help an agent understand how good states and available action are in terms of expected future return.
    + A Q value function evaluate how good or bad a state-action pair is.

==========

4/ Deep RL Algorithms:

- There are three major families of deep RL algorithms:
    + Policy-based
    + Value-based
    + Model_based
- These model can be combine:



**: discussed in this book
1. A2C: Advantage Actor-Critic
2. A3C: Asynchronous Advantage Actor-Critic
3. GAE: Actor-Critic with Generalized Advantage Estimation

a/ Policy-based Algorithms:

- This algorithms learn a policy pi. Good policies should generate action which produce trajectories that maximize an agent's object.
- If an agent needs to act in an environment, then it makes sense to learn a policy.
    + What determine a good action at a given moment depends on the state
    + Policy function take state s as input to produce an action a.
    + Advantage: the algorithms are very general class of optimization methods.
    + Disadvantage: very high variance and sample inefficient

b/ Value-based Algorithms:

- This algorithms learn V or Q function. It used the learned value function to evaluate (s, a) pairs and generate a policy.
- For ex: an agent's policy = select action a in state s with the highest estimated Q function.
- We use Q function better because it contain both state and action which is necessary to convert to policy.

- Value-based algorithms are typically more sample-efficient than policy-based algorithms.

c/ Model-based Algorithms:

- This algorithms either lean a model of an environment's transition dynamics or make use of a known dynamics model.
- Once an agent has a model of the environment, it can "imagine" what will happen in the future by predicting the trajectory of for a few time steps.
- An agent can complete many different trajectory predictions with different action sequences, then examine these different option to decide on the best action a to take.
- EX: Monte Carlo Tree Search.
- Model-based are very appealing since a perfect model can play out scenarios and understand the consequences of its action without having to actually act in the environment.
- However, the models are hard to come by. First, if the environment with large state space and action space can be very difficult to model. Second, the models are only useful when they can accurately predict the transition of the environment many step into the future.
- Model-based vs Model-free:
    + Model-based = any algorithms that make use of the transition dynamics of an environment, whether learned or known in advance.
    + Model-free algorithms = don't explicitly make use of the environment transition dynamics.

d/ Combined Methods:

- Actor-Critic algorithms = the policy acts and the value function critique the action.
    + During training, a learned value function can provide a more informative feedback signal to a policy than the sequence of rewards available from the environment.
    + During training, the policy learns using information provided by the learned value function

e/ On-Policy and Off-Policy Algorithms:

- Another distinction: An algo is on-policy **<u>if it learned on the policy</u>** ie, training can only utilize data generated from the current policy pi.
    + This implies that as training iterates through version of policy pi1, pi2, pi3,... each training iteration only used the current policy that time to generate training data.
    + All data must be discarded after training. This make on-policy methods sample inefficient, so they require more training data
    + REINFORCE, SARSA, Actor-Critics
- Off-policy: Any collected data can be reused in training. This method is more sample efficient but may require a large memory to store data
    + DQN, DDQN, PPO

<u>f/ SUMMARY:</u>

- Policy-based, value-based, model-based, or combined methods
- Model-based or model-free: whether an algorithms use a model of the environment's transition dynamics

- On-policy or off-policy: whether an algorithm learn with data gathered using just the current policy

==========

5/ Deep Learning for Reinforcement Learning:

- Assuming that we know the basic steps of training network
    + Sample random batch from dataset
    + Compute forward pass with the network using the input x to produce the predicted output
    + Computer the loss using predicted by network and actual label
    + Calculate gradient (autograd with pytorch)
    + Use optimizer to update the network param using the gradient.

=> The training step is repeated until the network's outputs stop changing or the loss has minimized or plateaued (network has converged)

==========

6/ Reinforcement Learning and Supervised Learning:

There are three main differences between them:

a/ Lack of the Oracle:

- SL: The correct answer for each model input is available
- RL: The correct answer for each model input is not available.
- In RL: after an agent takes action a in state s is only has access to the reward it received.
    + The agent is not told what the best action to take was.
    + Instead, it is only given an indication, via the reward, of how good or bad action a is.
    + To learn about (s,a,r) an agent must experience the transition (s,a,r,s')/ Consequently, an agent may have no knowledge about important parts of the state and action space because it has not experienced them.
    + One way to deal with this problem is to initialize episode to start in states we want an agent to learn about. However, it is not always possible to o describe the entire action we want to do. There is no guarantee that the agent will reach that reward that we want it to.

b/ Sparsity of Feedback:

- SL does not have the problems that RL has since each pair of output has a corresponding input.

c/ Data:

- In SL: The training data is independent from the algorithms training.
- In RL: Data must be generate by an agent interacting with an environment. The data and algorithm are coupled. The quality of an algorithm affect the data it is trained on which is turn affects an algorithm's performance.
- RL is also interactive - actions made by the agent actually change the environment -> changes the agent's decisio -> changes the data the agent sees….
- In RL: instead of minize the error of the loss between the network's output and desire target, **rewards from the environment are used to construct an objective, the the networks are trained s as to maximize this objective**

==========

7/ Summary:

- Agents lean how to act in the environment using a policy in order to maximize the expected sum or rewards.
- The three primary learnable function in RL: policy, value function, models
- Some differences between SL and RL.