Notes on Reinforcement Learning

ERMO WEI

Contents

Activation Function Activation Function are important for neural network to have non-linearity. The commonly used activation function are summarize below. Most of them are typically combined with an affine transformation $\mathbf{a} = \mathbf{b} + \mathbf{W}\mathbf{x}$ and applied element-wise.

$$\mathbf{h} = f(\mathbf{a}) \rightarrow h_i = f(a_i) = f(b_i + \mathbf{W}_{\mathbf{i},:} * \mathbf{x})$$

- Sigmoid

$$f(a) = \frac{1}{1 + e^{-a}}$$

- Hyperbolic tangent

$$f(a) = \tanh(a)$$

- Softmax
- Rectifier or rectified linear unit (ReLU)

$$f(a) = \max(0, a)$$

- Maxout
- Softplus
  A smooth version of the rectifier, basically the same shape. But this function has differentiability and non-zero derivative everywhere.

$$f(a) = \log(1 + e^a)$$

Actor-Critic Method Need more time to work on it ...

Back Propagation Let's start with linear neurons (also called linear filters). Here we first make two assumptions:

- The neuron has a real- valued output which is a weighted sum of its inputs
- The aim of learning is to minimize the square error summed over all training cases.

The model of the linear neuron is like this :

$$y = \sum w_i x_i = \mathbf{w}^T \mathbf{x}$$

Since we are dealing with a simple linear neuron, we first derive the special case of back propagation algorithm, called *delta rule*, which is a gradient descent learning rule for updating the weights of single layer neural network.

1. We first define the error of our training cases:

$$E = \frac{1}{2} \sum_{n \in \text{training set}} (t^n - y^n)^2$$

2. Now we take the derivative of the error with respect to the weights

$$\begin{aligned} \frac{\partial E}{\partial w_i} &= \frac{1}{2} \sum_n \frac{dE^n}{dy^n} \frac{\partial y^n}{\partial w_i} \\ &= -\sum_n x_i^n (t^n - y^n) \end{aligned}$$

3. The **batch** delta rule changes the weights in proportion to their error derivatives summed over all training cases

$$\Delta w_i = -\epsilon \frac{\partial E}{\partial w_i} = \sum_n \epsilon x_i^n (t^n - y^n)$$

One thing to be notice here, we can get as close as we desire to the best answer by making the learning rate small enough

Ok, since we have the simplest neuron, let's making things little interesting by adding the logistic function. And now, our model become this:

$$z = b + \sum_i w_i x_i$$
$$y = \frac{1}{1 + e^{-z}}$$

their corresponding derivatives are

$$\frac{\partial z}{\partial w_i} = x_i \,, \frac{\partial z}{\partial x_i} = w_i$$
$$\frac{dy}{dz} = y(1 - y)$$

Thus, take the derivative of y with respect to w_i is

$$\frac{\partial y}{\partial w_i} = \frac{dy}{dz} \frac{\partial z}{\partial w_i} = x_i y(1 - y)$$

and

$$\frac{\partial E}{\partial w_i} = \sum \frac{\partial E}{\partial y^n} \frac{\partial y^n}{\partial w_i} = -\sum_n x_i^n y^n (1 - y^n)(t^n - y^n)$$

In this equation, the first and last terms come from the delta rule, and the term in the middle is the slope of logistic

Need more time to work on it ...

Bellman Equation Due to the Markov property of MDP, we can define the value function of a policy recursively to be

$$\begin{aligned} V^\pi(s) &= \mathbb{E}_\pi\{R_t | s_t = s\} \\ &= \mathbb{E}_\pi\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s\} \\ &= \mathbb{E}_\pi\{r_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k r_{t+k+2} | s_t = s\} \end{aligned}$$

If policy is stochastic, then we have

$$\begin{aligned} V^\pi(s) &= \sum_a \pi(s,a) \sum_{s'} \Pr(s'|s,a) \big[R(s,a) + \gamma\, \mathbb{E}_\pi\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+2} | s_{t+1} = s'\}\big] \\ &= \sum_a \pi(s,a) \sum_{s'} \Pr(s'|s,a) \big[R(s,a) + \gamma V^\pi(s')\big] \\ &= \sum_a \pi(s,a) \{R(s,a) + \sum_{s'} \Pr(s'|s,a) \gamma V^\pi(s')\} \end{aligned}$$

If policy is deterministic, then the Value function of policy π at state s is

$$V^{\pi}(s) = R(s, \pi(s)) + \gamma \sum_{s'} \Pr(s'|s, \pi(s)) V^{\pi}(s')$$

Thus, the Bellman optimality equation can be defined as

$$V^{*}(s) = \max_{a} \{R(s, a) + \gamma \sum_{s'} \Pr(s'|s, a) V^{*}(s')\}$$

Detail see (Barto, 1998)

Bellman Operator Consider a MDP, Bellman Operator for policy π can be defined as a map from a value function to another value function

$$T^{\pi} : \mathbb{R}^N \to \mathbb{R}^N$$

where N is the cardinality of state space, i.e. $N = |S|$.

Need more time to work on it ...

Boltzmann Selection Need more time to work on it ...

Contraction The operator F (function f) is a α-contraction ($0 \leq \alpha < 1$, called Lipschitz constant for F) with respect to some norm $\|\cdot\|$ if

$$\forall X, \overline{X} : \|FX - F\overline{X}\| \leq \alpha \|X - \overline{X}\| \quad \text{or} \quad \|f(X) - f(\overline{X})\| \leq \alpha \|X - \overline{X}\|$$

- Theorem 1. The sequence $X, FX, F^2X, \ldots$ converges for every X. e.g. $X, f(X), f(f(X)), \ldots$ converges for every X

  Proof:

  Useful fact : Cauchy sequences : If for $x_0, x_1, x_2, \ldots$, we have that

  $$\forall \epsilon, \exists K : \|x_M - x_N\| < \epsilon \quad \text{for} \quad M, N > K$$

  then we call $x_0, x_1, x_2, \ldots$, a Cauchy sequence.

  If $x_0, x_1, x_2, \ldots$, is a Cauchy sequence, and $x_i \in \mathbb{R}^n$, then there exists $x^* \in \mathbb{R}^n$ such that $\lim_{i \to \infty} x_i = x^*$.

  Proof: Assume $N > M$.

$$\begin{aligned}
\|F^M X - F^N X\| &= \|\sum_{i=M}^{N-1} (F^i X - F^{i+1} X)\| \\
&\leq \sum_{i=M}^{N-1} \|F^i X - F^{i+1} X\| && \text{by triangle inequality} \\
&\leq \sum_{i=M}^{N-1} \alpha^i \|X - FX\| && \text{use the condition in Theorem} \\
&= \|X - FX\| \sum_{i=M}^{N-1} \alpha^i \\
&= \|X - FX\| \frac{\alpha^M}{1 - \alpha}
\end{aligned}$$

  As $\|X - FX\| \frac{\alpha^M}{1-\alpha}$ goes to zero for M going to infinity, we have that for any $\epsilon > 0$ for $\|F^M X - F^N X\| \leq \epsilon$ to hold for all $M, N > K$, it suffices to pick K large enough. Hence, $X, FX, \ldots$ is Cauchy sequence and converges.

- Theorem 2 (Banach fixed-point theorem). F has a unique fixed point X^* which satisfies $FX^* = X^*$ and all sequences $X, FX, F^2X, \dots$ converge this unique fixed point X.

  Proof:

  Suppose F has two fixed points. Let's say

  $$FX_1 = X_1$$
  $$FX_2 = X_2$$

  this implies, $\|FX_1 - FX_2\| = \|X_1 - X_2\|$. At the same time we have from the contractive property of F

  $$\|FX_1 - FX_2\| \leq \alpha\|X_1 - X_2\|$$

  Combining both gives us

  $$\|X_1 - X_2\| \leq \alpha\|X_1 - X_2\|$$

  Hence, $X_1 = X_2$.

Detail see (Conrad, 2014)

Compatible Feature If we are using Softmax Policy, which is

$$\pi(s,a) = \frac{e^{\phi(s,a)^\top \theta}}{\sum_b e^{\phi(s,b)^\top \theta}}$$

if we have $p = e^{\phi(s,a)^\top \theta}$ and $q = \sum_b e^{\phi(s,b)^\top \theta}$, then $\pi(s,a) = p/q$

$$\begin{aligned}
\nabla_\theta \ln \pi(s,a) &= \frac{1}{\pi(s,a)} \frac{\partial \pi(s,a)}{\partial \theta} \\
&= \frac{q}{p} \frac{\frac{\partial p}{\partial \theta} q - \frac{\partial q}{\partial \theta} p}{q^2} \\
&= \frac{\frac{\partial p}{\partial \theta}}{p} - \frac{\frac{\partial q}{\partial \theta}}{q} \\
&= \frac{e^{\phi(s,a)^\top \theta} \phi(s,a)}{e^{\phi(s,a)^\top \theta}} - \frac{\sum_b (e^{\phi(s,b)^\top \theta} \phi(s,b))}{\sum_b e^{\phi(s,b)^\top \theta}} \\
&= \phi(s,a) - \sum_b \frac{e^{\phi(s,b)^\top \theta} \phi(s,b)}{\sum_b e^{\phi(s,b)^\top \theta}} \\
&= \phi(s,a) - \sum_b \Big[\frac{e^{\phi(s,b)^\top \theta}}{\sum_b e^{\phi(s,b)^\top \theta}} \phi(s,b) \Big] \\
&= \phi(s,a) - \sum_b \pi(s,b) \phi(s,b)
\end{aligned}$$

if we are using d-dimensional Gaussian Policy, suppose we fix the covariant matrix to $\mathbf{\Sigma} = c\mathbf{I}$, then our policy is

$$\pi(s,a) = \frac{1}{\sqrt{(2\pi)^d |\Sigma|}} \exp\Big\{ -\frac{1}{2} (a - \phi(s)^\top \theta)^\top \Sigma^{-1} (a - \phi(s)^\top \theta) \Big\}$$

the score function is

$$\begin{aligned}
\nabla_\theta \ln \pi(s,a) &= \frac{1}{\pi(s,a)} \frac{\partial \pi(s,a)}{\partial \theta} \\
&= \frac{\sqrt{(2\pi)^d |\Sigma|}}{\exp\{\dots\}} \frac{\exp\{\dots\}}{\sqrt{(2\pi)^d |\Sigma|}} (-\frac{1}{2}) \Big(\frac{\partial (a - \phi(s)^\top \theta)^\top \Sigma^{-1} (a - \phi(s)^\top \theta)}{\partial \theta} \Big) \\
&= -\Sigma^{-1} \big[a - \phi(s)^\top \theta \big] \phi(s) =
\end{aligned}$$

ϵ-greedy ϵ-greedy is a common used exploration strategy for Model-free reinforcement learning. This strategy can be seen as a combination of greedy strategy and random strategy. Every time when agent choose an action to perform, it choose greedily with probability 1-ϵ (exploitation), and randomly with probability ϵ (exploration). However, based on the changes of ϵ value, this method have two versions. First, the ϵ value can decrease as the learning process goes on, this is the *decay exploration* method. Another one which are more commonly used is that we fix the value of ϵ.

The difference between those methods is that, the first one can be GLIE if the ϵ-value goes 0 eventually but the second one cannot. Suppose we have a counter of how many times a state have been visited, $n_t(s)$ and a constant c. As long as the ϵ value for the a state $\epsilon_t(s) = \frac{c}{n_t(s)}$ where $0 < c < 1$, this method can be consider GLIE. However, in practice, we usually use fixed value for ϵ and after the convergence of the estimation value of action, we switch to greedy selection.

Ergodic MDP An MDP is said to be ergodic if for each policy π the Markov chain induced by π is ergodic. We are giving the definition of Ergodicity below. But first, we will give some auxiliary definitions. Several definition of Markov chain:

- Reducibility
  A Markov chain is said to be irreducible if it is possible to get to any state from any state.
- Aperiodicity
  A state i has period k if any return to state i must occur in multiples of k time steps. For example, suppose it is possible to return to a state in $\{6, 8, 10, 12, \ldots\}$ time steps, then k would be 2, even though 2 does not appear in this list. If $k = 1$, then the state is said to be aperiodic: returns to state i can occur at irregular times.
- Recurrence
  A state i is said to be transient if, given that we start in state i, there is a non-zero probability that we will never return to i. State i is recurrent (or persistent) if it is not transient. Recurrent states are guaranteed to have a finite hitting time.

Here we have the definition of Ergodicity.

A state i is said to be ergodic if it is aperiodic and positive recurrent. In other words, a state i is ergodic if it is recurrent, has a period of 1 and it has finite mean recurrence time. If all states in an irreducible Markov chain are ergodic, then the chain is said to be ergodic.

It can be shown that a finite state irreducible Markov chain is ergodic if it has an aperiodic state. **A model has the ergodic property if there's a finite number N such that any state can be reached from any other state in exactly N steps.** For example, we will have $N = 1$ if we have a fully connected transition matrix where all transitions have a non-zero probability. **Additionally, an stationary distribution d^π of states exists and is independent start state s_0.**

Detail see (Ortner, 2007).

Experience Replay (Vanseijen and Sutton, 2015) (Riedmiller, 2005)

Function Approximation Function approximation is a way of combining supervised learning algorithms with learning procedures in Reinforcement Learning to be able to generalize value function into large scale MDP. There are three different objective function for function approximation.

1. Direct method
   In direct method, we are trying to minimize the square error of our estimation and the "true" value

function which we temporarily assume given by an oracle.

$$J(\theta) = \frac{1}{2}(v(s) - V_\theta(s))^2$$

The term $\frac{1}{2}$ is just for neat math like regular supervised learning, and $v(s)$ is the "true" value for state s which we want to approximate. To use the gradient descent method, we take the gradient of the $J(\theta)$, which gives us

$$\begin{aligned}\nabla_\theta J(\theta) &= \frac{1}{2} * 2 * (v(s) - V_\theta(s))\nabla_\theta V_\theta(s) \\ &= (v(s) - V_\theta(s))\nabla_\theta V_\theta(s)\end{aligned}$$

Note, in the equation above, $v(s)$ is just a constant number. Since we don't really have a oracle in learning, we need to find a way to estimate the "true" value function. Various method can be used here. For example, we can use the temporal difference learning rule, thus the gradient become $(r + \gamma V_\theta(s') - V_\theta(s))\nabla_\theta V_\theta(s)$, or we can use the monte carlo estimation, then the gradient become $(G_t - V_\theta(s))\nabla_\theta V_\theta(s)$ or eligibility trace.

However, this direct method has some issue with convergence. detail here

2. Fixed point method
   Another kind of objective function is based on the "fixed point" notion in contraction. With the contraction property, we know the value function will converge to a fixed point

$$V = TV$$

where T is the bellman operator. Thus, after convergence, our value function should fulfill the Bellman equation which gives the second objective function

$$\begin{aligned}J(\theta) &= \frac{1}{2}(V_\theta(s) - TV_\theta(s))^2 \\ &= \frac{1}{2}(V_\theta(s) - (r + \gamma V_\theta(s')))^2\end{aligned}$$

when we take the gradient of this cost function $J(\theta)$, we have

$$\begin{aligned}\nabla_\theta J(\theta) &= \frac{1}{2} * 2 * (r + \gamma V_\theta(s') - V_\theta(s))\nabla_\theta(r + \gamma V_\theta(s') - V_\theta(s)) \\ &= (r + \gamma V_\theta(s') - V_\theta(s))(\nabla_\theta \gamma V_\theta(s') - \nabla_\theta V_\theta(s)) \\ &= (r + \gamma V_\theta(s') - V_\theta(s))(\gamma \nabla_\theta V_\theta(s') - \nabla_\theta V_\theta(s))\end{aligned}$$

need more stuff here

3. Projected fixed point method

GLIE GLIE stands for "Greedy in the Limit of Infinite Exploration" (Singh et al., 2000). The learning policies in RL can be divided into two broad categories: a *decay exploration* strategy which become more and more greedy and *persistent exploration* which always maintain a fix exploration rate. The advantage of the first one is that we can eventually converge to the optimal policy. The second one may have the advantage always be adaptive but may not converge to the optimal. (In here, we talk about convergence in the sense that the behavior will become optimal. It is possible that some of the algorithm converge to the correct Q-value but still behave randomly with some probability by using persistent exploration strategy, Q-learning with fix ϵ-greedy for example). We may want to consider this in the context of on-policy&off-policy.

If a *decay exploration* strategy has the following two characters:

1. each action is executed infinitely often in every state that is visited infinitely often, and
2. in the limit, the learning policy is greedy with respect to the Q-value function with probability 1.

Than we can consider this decay exploration strategy GLIE. Some example of GLIE include Boltzmann Selection, ϵ-greedy.

Inverse Reinforcement Learning Reason for using IRL instead of behavior cloning:

- Reward functions can generalize the expert's behavior to the state space regions not covered by the demonstration. For example, the transition model can change and thus, the optimal policy change as well.
- behavioral cloning cannot be applied whenever the expert demonstrates actions that cannot be executed by the learner (think at a humanoid robot that learns by observing a human expert)

Lipschitz Continuity Given two metric space two norm (a function that assigns a strictly positive length or size to each vector in a vector space)

Given two metric spaces (X, dX) and (Y, dY), where dX denotes the metric on the set X and dY is the metric on set Y (for example, Y might be the set of real numbers R with the metric dY(x, y) = —x ? y—, and X might be a subset of R), a function f : X ? Y is called Lipschitz continuous if there exists a real constant K ? 0 such that, for all x1 and x2 in X,

(Eriksson et al., 2013)

Linear Quadratic Regulator When we consider a special system with linear dynamic and quadratic cost, we have some very nice algorithm so solve this questions.

$$\begin{aligned} x_{t+1} &= A_t x_t + B_t u_t + w_t \quad \text{for} \quad t = 0, 1, \ldots, N-1 \\ C(x_t, u_t) &= x_t^\top Q_t x_t + u_t^\top R_t u_t \end{aligned}$$

In the equation above, x_t are the state with n-dimension, and u_t are the control (action) with m-dimension. A_t, B_t, Q_t, R_t are given and have appropriate dimension. Q_t are positive semidefinite symmetric and R_t are positive definite symmetric. w_t is the noise term with given probability distributions that do not depend on x_t ad u_t. It also has zero mean and finite variance.

The reason for use this kind of control is that we want to keep the state of the system close to the origin. The quadratic cost function will induce heavy penalty if the system is largely deviate from the it. So now we apply Dynamic Programming algorithm to it. First, we have the cost-to-go function of the last state N and other states.

$$\begin{aligned} J_N(x_N) &= x_N^\top Q_N x_N \\ J_t(x_t) &= \min_{u_k} \mathbb{E}\left\{x_t^\top Q_t x_t + u_t^\top R_t u_t + J_{t+1}(A_t x_t + B_t u_t + w_t)\right\} \end{aligned}$$

We can replace the t with $N-1$, then we have

$$\begin{aligned} J_{N-1}(x_{N-1}) = \min_{u_{N-1}} \mathbb{E}\big\{ & x_{N-1}^\top Q_{N-1} x_{N-1} + u_{N-1}^\top R_{N-1} u_{N-1} \\ & + (A_{N-1}x_{N-1} + B_{N-1}u_{N-1} + w_{N-1})^\top Q_N (A_{N-1}x_{N-1} + B_{N-1}u_{N-1} + w_{N-1})\big\} \end{aligned}$$

Next, we expand the term inside square brackets.

$$\begin{aligned} J_{N-1}(x_{N-1}) = x_{N-1}^\top Q_{N-1} x_{N-1} + \min_{u_{N-1}} \big[& u_{N-1}^\top R_{N-1} u_{N-1} + u_{N-1}^\top B_{N-1}^\top Q_N B_{N-1} u_{N-1} \\ & + 2 x_{N-1}^\top A_{N-1}^\top Q_N B_{N-1} u_{N-1}\big] + x_{N-1}^\top A_{N-1}^\top Q_N A_{N-1} x_{N-1} + \mathbb{E}\left\{w_{N-1}^\top Q_N w_{N-1}\right\} \end{aligned}$$

Several things to note here. First, some of the terms does not depend on u_{N-1}, thus we move them out of the minimization operator. Second, $\mathbb{E}\{w_{N-1}\} = 0$ suggests that $\mathbb{E}\{w_{N-1}^\top Q_N (A_{N-1}x_{N-1} + B_{N-1}u_{N-1})\}$ is also zero. Third, we reverse the position of some terms due to the transpose. If we differentiate the equation with respect to u_{N-1} and set the derivative to zero, we get

$$(R_{N-1} + B_{N-1}^\top Q_N B_{N-1}) u_{N-1} = -B_{N-1}^\top Q_N A_{N-1} x_{N-1}$$

Since R_{N-1} is positive definite and $B_{N-1}^\top Q_{N-1} B_{N-1}$ is positive semidefinite, matrix multiplying u_{N-1} on the left is positive definite (and hence invertible). Thus, the minimizing control vector is given by

$$u_{N-1}^* = -(R_{N-1} + B_{N-1}^\top Q_N B_{N-1})^{-1} B_{N-1}^\top Q_N A_{N-1} x_{N-1}$$

By substitution into the expression for J_{N-1}, we have

$$J_{N-1}(x_{N-1}) = x_{N-1}^\top K_{N-1} x_{N-1} + \mathbb{E}\left\{w_{N-1}^\top Q_N w_{N-1}\right\}$$

and matrix K_{N-1} should be

$$K_{N-1} = A_{N-1}^\top (Q_N - Q_N B_{N-1}(B_{N-1}^\top Q_N B_{N-1} + R_{N-1})^{-1} B_{N-1}^\top Q_N) A_{N-1} + Q_{N-1}$$

LSPI Need more time to work on it ...

LSTD Need more time to work on it ... Related paper

Long-Short Term Memory Need more time to work on it ...

Markov Decision Process blah blah here

Markov Decision Process can be seen as a extension of Markov Chain with additional action set (allowing selection) and reward function (motivation). It can be reduced to Markov chain if we have only one action per state and same reward for all the state.

Metric Space Need more time to work on it ...

Definition (Metric Space, metric). A *metric space* is a pair (X, d), where X is a set and d is a metric on X (distance function on X), that is, a function defined[1] on $X \times X$ such that for all $x, y, z \in X$ we have:

1. d is real-valued, finite and nonnegative.
2. $d(x, y) = 0$ if and only if $x = y$
3. $d(x, y) = d(y, x)$ (Symmetry)
4. $d(x, y) \leq d(x, z) + d(z, y)$ (Triangle inequality)

In this definition, X is called the underlying set of (X, d). Its elements are called points. Here are some examples of metric space:

- Real line $\mathbb{R}$
  The set of all real numbers and $d(x, y) = |x - y|$
- Euclidean plane $\mathbb{R}^2$
  Suppose we have $x = (\epsilon_1, \epsilon_2)$ and $y = (\upsilon_1, \upsilon_2)$. The euclidean metric defined by

$$d(x, y) = \sqrt{(\epsilon_1 - \upsilon_1)^2 + (\epsilon_2 - \upsilon_2)^2}$$

- Euclidean space $\mathbb{R}^n$
  If $x = (\epsilon_1, \ldots, \epsilon_n)$ and $y = (\upsilon_1, \ldots, \upsilon_n)$, then euclidean metric defined by

$$d(x, y) = \sqrt{(\epsilon_1 - \upsilon_1)^2 + \ldots + (\epsilon_n - \upsilon_n)^2}$$

[1] The symbol $\times$ denotes the Cartesian product of sets: $A \times B$ is the set of all ordered pairs (a, b), where $a \in A$, and $b \in B$. Hence, $X \times X$ is the set of all ordered pairs of elements of X

- Unitary space $\mathbb{C}^n$
  a n-dimensional unitary space $\mathbb{C}^n$ is the space of all ordered n-tuples of complex numbers with distance function
  $$d(x, y) = \sqrt{|\epsilon_1 - \upsilon_1|^2 + \ldots + |\epsilon_n - \upsilon_n|^2}$$
- Sequence space l^∞
  This example and the next one give a first impression of how surprisingly the concept of a metric space is. As a set X we take the set of all bounded sequences of complex numbers; that is, every element of X is a complex sequence
  $$x = (\epsilon_1, \epsilon_2, \ldots) \quad \text{briefly} \quad x = (\epsilon_j)$$
  such that for all $j = 1, 2, \ldots$ we have
  $$|\epsilon_j| \leq c_x$$
  where c_x is a real number which may depend on x, but does not depend on j. We choose the metric defined by
  $$d(x, y) = \sup_{j \in \mathbb{N}} |\epsilon_j - \upsilon_j|$$
  where $y = (\upsilon_j) \in X$ and $\mathbb{N} = 1, 2, \ldots$, and sup denotes the supremum (least upper bound).
- Function space $C[a, b]$
  As a set X we take the set of all real-valued functions $x, y, \ldots$ which are functions of an independent real variable t and are defined and continuous on a given closed interval $J = [a.b]$. Choosing the metric defined by
  $$d(x, y) = \max_{t \in J} |x(t) - y(t)|$$
  where max denotes the maximum, we obtain a metric space which is denoted by $C[a, b]$. (The letter C suggests "continuous.") This is a function space because every point of $C[a, b]$ is a function.

(Kreyszig, 1989).

Monte Carlo Method Monte Carlo method is a way of making the prediction in model-free environment. The question it wants to solve is that suppose we have a policy π known, how good is this policy? In this case, we evaluate the policy by giving the method episodes of experience $\{s_1, a_1, r_2, \ldots, s_T\}$ generated by following policy π and wants the value function V^π as output.

As we know, the value of being in a state s is the expectation of the discounted rewards received afterwards.

$$V^\pi(s) = \mathbb{E}_\pi[r_{t+1} + \gamma r_{t+2} + \ldots + \gamma^{T-1} r_T]$$

two methods can be used here : first-visit and every-visit method

Off-Policy Policy Gradient Method In REINFORCE method, when we try to evaluate the expected return of a policy π, we have to run the policy several time to be able to have reasonable estimation of how good the policy is. Let $J(\theta)$ denote the expected return of policy π_θ, and τ denote the trajectory (essentially a state-action sequence : $s_0, a_0, s_1, a_1, \ldots, s_T, a_T$) or rollout or history (these terms are interchangeable) of executing the policy π_θ, then we know

$$\begin{aligned} J(\theta) &= \mathbb{E}_{\pi_\theta}\left[\sum_{t=0}^{H} \gamma^t R(s_t, u_t) | \pi_\theta\right] \\ &= \sum_\tau P(\tau|\theta) R(\tau) \end{aligned}$$

where $P(\tau|\theta)$ is the probability of having a trajectory τ by following policy π_θ and $R(\tau)$ is just the accumulated reward of that trajectory. In policy gradient method, after we evaluate the policy π_θ, we may want to improve

it and use a new policy. Thus, the sample we collected during the process of following policy π_θ are discarded. However, it will preferable if we can reuse the data gathered of following one policy to estimate the value of following another policy. The method "likelihood ratio" estimation make this data reuse possible.

In practice, if we want to evaluate $J(\theta)$, we may want to draw the rollout samples from the distribution induced by policy π_θ. After taking N samples $\{\tau_0, \tau_1, \ldots, \tau_N\}$, we have a unbiased estimator:

$$\hat{J(\theta)} = \frac{1}{N} \sum_i R(\tau_i)$$

Imagine, however, instead of π_θ, we only have $\pi_{\theta'}$ available, we can do some trick like this

$$\begin{aligned} J(\theta) &= \sum_\tau P(\tau|\theta) R(\tau) \\ &= \sum_\tau P(\tau|\theta) \frac{P(\tau|\theta')}{P(\tau|\theta')} R(\tau) \\ &= \sum_\tau P(\tau|\theta') \frac{P(\tau|\theta)}{P(\tau|\theta')} R(\tau) \\ &= \mathbb{E}_{\pi_{\theta'}} \left[\frac{P(\tau|\theta)}{P(\tau|\theta')} R(\tau) \right] \end{aligned}$$

Note that, $P(\tau|\theta)$ and $P(\tau|\theta')$ is the probability of same sample τ under different distribution defined by θ and θ'. Now, we can estimate the $J(\theta)$ with respect to the the distribution induced by $\pi_{\theta'}$. This method of estimating one expectation with respect to another distribution is called ''importance sampling". So, we can rewrite the $\hat{J(\theta)}$ as

$$\hat{J(\theta)} = \frac{1}{N} \sum_i R(\tau_i) \frac{P(\tau|\theta)}{P(\tau|\theta')}$$

Note, in the equation above, we have a term $\frac{P(\tau|\theta)}{P(\tau|\theta')}$. Since we don't have the model of the world, it's not possible to directly compute $P(\tau|\theta)$ or $P(\tau|\theta')$. But if we expand the fraction term (note that $P(\tau|\theta)$ and $P(\tau|\theta')$ both evaluate on the same trajectory sample τ, therefore the state action sequence is same for both), we can see that

$$\begin{aligned} \frac{P(\tau|\theta)}{P(\tau|\theta')} &= \frac{P(s_0) \prod_{t=0}^T \left[P(s_{t+1}|s_t, a_t) P(a_t|s_t, \theta) \right]}{P(s_0) \prod_{t=0}^T \left[P(s_{t+1}|s_t, a_t) P(a_t|s_t, \theta') \right]} \\ &= \frac{P(s_0) \prod_{t=0}^T P(s_{t+1}|s_t, a_t) \prod_{t=0}^T P(a_t|s_t, \theta)}{P(s_0) \prod_{t=0}^T P(s_{t+1}|s_t, a_t) \prod_{t=0}^T P(a_t|s_t, \theta')} \\ &= \frac{\prod_{t=0}^T P(a_t|s_t, \theta)}{\prod_{t=0}^T P(a_t|s_t, \theta')} \\ &= \frac{\prod_{t=0}^T \pi_\theta(a_t|s_t)}{\prod_{t=0}^T \pi_{\theta'}(a_t|s_t)} \end{aligned}$$

Thus, we should be able to calculate the likelihood $\frac{P(\tau|\theta)}{P(\tau|\theta')}$ for any two policies θ and θ'. Actually, we can have a mixture distributions, where $P(\tau|\theta')$ is replaced by $\frac{1}{N} \sum_j P(\tau_j|\theta_j)$ where τ_j are trajectory of following θ_j, then we can make use of multiple source of distributions.

On-policy and Off-policy An RL algorithm can be essentially divided into two parts, the *learning policy* and *update rule*. The first one is a non-stationary policy that maps experience (state visited, action chosen, reward received) to into a currently choice of action. The second part is how the algorithm uses experience

to change its estimation of the optimal value function. In off-policy algorithm, the *update rules* doesn't have relationship with *learning policy*, that is the *update rules* doesn't care the what action agent take. Q-learning can be consider as the off-policy algorithm.

$$Q_{t+1}(s,a) = (1-\alpha)Q_t(s,a) + \alpha(r_t + \gamma \max_{a'} Q(s',a'))$$

We can see that the Q-value is update based on the $\max_{a'} Q(s',a')$, which doesn't depend on the action the agent was taking.

However, if we take a look of SARSA(0), which is very similar to Q-learning.

$$Q_{t+1}(s,a) = (1-\alpha)Q_t(s,a) + \alpha(r_t + \gamma Q(s',a'))$$

We can see the update is based on the Q-value of the next action of the agent. Thus it is an on-policy algorithm. The convergence condition are heavily depend on the *learning policy*, The Q-value of SARSA(0) can only converge to optimality in the limit only if the learning policy behavior optimal in the limit. The SARSA(0) and Q-learning will be same if we use greedy action selection strategy.

Detail see (Singh et al., 2000).

Policy Gradient Reinforcement Learning Instead of determining the action based on value function, another method of determine the action to take is the direct policy search method. It is well known that value-function combined with function approximation are unable to converge to any policy even for simple MDPs (in mean-squared-error, residual-gradient, temporal-difference, and dynamic-programming sense)

Let θ denote the parameters of the function approximator, neural network for example, and $J(\theta)$ the performance of the corresponding policy (e.g. the average reward per step). Then in the policy gradient approach, we can update the policy parameters proportional to the gradient:

$$\Delta\theta = \alpha \frac{\partial J(\theta)}{\partial \theta}$$

Let's begin with the definition of the two formulations:

- Average reward formulation

$$J(\theta) = \lim_{n\to\infty} \frac{1}{n} \mathbb{E}\{r_1 + r_2 + \ldots + r_n | \pi\} = \sum_s d^\pi(s) \sum_a \pi(s,a) R(s,a)$$

  where $d^\pi(s)$ is the stationary distribution of the states induced by π

where α is a positive-definite step size.

Policy Gradient Theorem Using the same notation in Policy Gradient,

Policy Iteration Need more time to work on it ...

REINFORCE REINFORCE algorithm also finds an unbiased estimate of the gradient, but without the assistance of a learned value function. REINFORCE learns much more slowly than RL methods using value functions and has received relatively little attention. Learning a value function and using it to reduce the variance of the gradient estimate appears to be essential for rapid learning.

Likelihood Ratio Methods

$$\begin{aligned}
J(\theta) &= \mathbb{E}\left[r(\tau)\right] \\
&= \int_{\tau} p_\theta(\tau)\, r(\tau) \\
\nabla_\theta J(\theta) &= \nabla_\theta \int_{\tau} p_\theta(\tau)\, r(\tau) \\
&= \int_{\tau} \nabla_\theta p_\theta(\tau)\, r(\tau) \\
&= \int_{\tau} \nabla_\theta p_\theta(\tau)\, \frac{p_\theta(\tau)}{p_\theta(\tau)}\, r(\tau) \\
&= \int_{\tau} p_\theta(\tau)\, \frac{\nabla_\theta p_\theta(\tau)}{p_\theta(\tau)}\, r(\tau) \\
&= \int_{\tau} p_\theta(\tau)\, \nabla_\theta \ln p_\theta(\tau)\, r(\tau) \\
&= \mathbb{E}\left[\nabla_\theta \ln p_\theta(\tau)\, r(\tau)\right]
\end{aligned}$$

However, we know that

$$p_\theta(\tau) = p(x_0) \prod_{k=0}^{H} p(x_{k+1}|x_k, u_k)\pi_\theta(u_k|x_k)$$

Thus

$$\begin{aligned}
\nabla_\theta \ln p_\theta(\tau) &= \nabla_\theta \big[\ln p(x_0) + \sum_{k=0}^{H} (\ln p(x_{k+1}|x_k, u_k) + \ln \pi_\theta(u_k|x_k)) \big] \\
&= \nabla_\theta \big[\ln p(x_0) + \sum_{k=0}^{H} \ln p(x_{k+1}|x_k, u_k) + \sum_{k=0}^{H} \ln \pi_\theta(u_k|x_k) \big] \\
&= \nabla_\theta \ln p(x_0) + \sum_{k=0}^{H} \nabla_\theta \ln p(x_{k+1}|x_k, u_k) + \sum_{k=0}^{H} \nabla_\theta \ln \pi_\theta(u_k|x_k) \\
&= 0 + 0 + \sum_{k=0}^{H} \nabla_\theta \ln \pi_\theta(u_k|x_k) \\
&= \sum_{k=0}^{H} \nabla_\theta \ln \pi_\theta(u_k|x_k)
\end{aligned}$$

Note, if instead of stochastic policy, we are using a deterministic policy, then we have

$$p_\theta(\tau) = p(x_0) \prod_{k=0}^{H} p(x_{k+1}|x_k, \pi_\theta(x_k))$$

In this case,

$$\begin{aligned}
\nabla_\theta \ln p_\theta(\tau) &= \nabla_\theta \big[\ln p(x_0) + \sum_{k=0}^{H} \ln p(x_{k+1}|x_k, \pi_\theta(x_k)) \big] \\
&= \nabla_\theta \ln p(x_0) + \sum_{k=0}^{H} \nabla_\theta \ln p(x_{k+1}|x_k, \pi_\theta(x_k)) \\
&= \nabla_\theta \ln p(x_0) + \sum_{k=0}^{H} \nabla_{u_k} \ln p(x_{k+1}|x_k, u_k) \nabla_\theta \pi_\theta(x_k) \\
&= 0 + \sum_{k=0}^{H} \nabla_{u_k} \ln p(x_{k+1}|x_k, u_k) \nabla_\theta \pi(x_k) \\
&= \sum_{k=0}^{H} \nabla_{u_k} \ln p(x_{k+1}|x_k, u_k) \nabla_\theta \pi(x_k)
\end{aligned}$$

Since we need to compute $\nabla_{u_k} \ln p(x_{k+1}|x_k, u_k)$, thus we need to know the model of the system

Stationary Distribution Need more time to work on it ...

Stochastic Game Stochastic game can been seen as an extension of MDP.

Universal Approximator Accroding to (**?**), Multilayer feedforward networks are universal approximators. Single hidden layer feedforward networks can approximate any measurable function arbitrarily well regardless of the activation function Ψ, the dimension of the input space r, and the input space environment μ

1. universal approximators: standard multilayer feedforward networks are capable of approximating any measurable function to any desired degree of accuracy
2. there are no theoretical constraints for the success of feedforward networks
3. lack of success is due to inadequate learning, insufficient number of hidden units or the lack of a deterministic relationship between input and target
4. rate of convergence as the number of hidden units grows
5. rate of increase of the number of hidden units as the input dimension increases for a fixed accuracy

Value Iteration Need more time to work on it ...

Vector Space Need more time to work on it ...

References

A. G. Barto. *Reinforcement learning: An introduction*. MIT press, 1998.

K. Conrad. The contraction mapping theorem. *Expository paper. University of Connecticut, College of Liberal Arts and Sciences, Department of Mathematics*, 2014.

K. Eriksson, D. Estep, and C. Johnson. *Applied Mathematics: Body and Soul: Volume 1: Derivatives and Geometry in IR3*. Springer Science & Business Media, 2013.

E. Kreyszig. *Introductory functional analysis with applications*, volume 81. 1989.

R. Ortner. Linear dependence of stationary distributions in ergodic markov decision processes. *Operations research letters*, 35(5):619–626, 2007.

M. Riedmiller. Neural fitted q iteration – first experiences with a data efficient neural reinforcement learning method. In *Machine Learning: ECML 2005*, volume 3720 of *Lecture Notes in Computer Science*, pages 317–328. 2005.

S. Singh, T. Jaakkola, M. L. Littman, and C. Szepesvári. Convergence results for single-step on-policy reinforcement-learning algorithms. *Machine Learning*, 38(3):287–308, 2000.

H. Vanseijen and R. Sutton. A deeper look at planning as learning from replay. In *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*, pages 2314–2322, 2015.