

Lecture 7 Reinforcement Learning Basics

7.1 Introduction

Reinforcement learning has shown such promise in other fields such as playing the game of Go (<https://www.alphagomovie.com/?ref=mlq.ai>), playing Star-craft, self-driving cars, improving energy efficiency. The application of deep reinforcement learning for trading has been of in-depth explorations and fierce competition among leading trading firms.

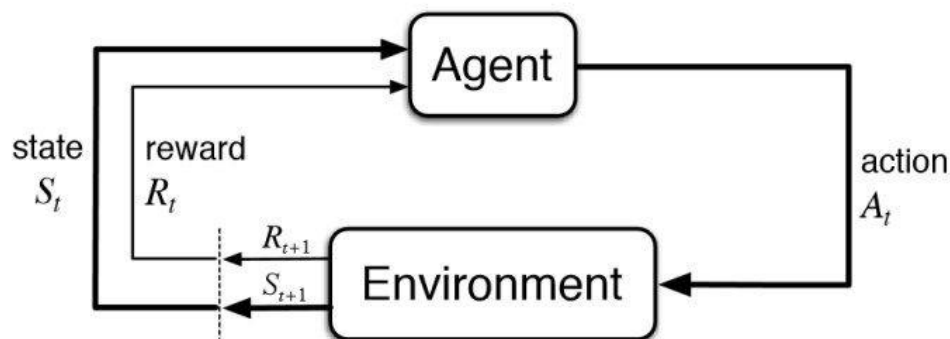
Reinforcement learning is a branch of machine learning that is based on training an agent how to operate in an environment based on a system of rewards. For example, if you're training an agent how to play a video game it would learn how to operate in the environment by the points earned or lost. In the context of trading, an agent would learn how to optimize a portfolio based on metrics like profit, loss, return, volatility, Sharpe-ratio and so on.

Reinforcement learning is quite different from both supervised & unsupervised machine learning techniques. Supervised and unsupervised learning algorithms are making predictions about data - for example, classifying an image, or identifying anomalies in data. Reinforcement learning is about training an agent to operate in an environment through interaction in order to maximize reward. In reinforcement learning, our agent is interacting with data, but the key difference is that its actions affect the environment, and the agent has a goal it wants to reach.

7.2 Basic Components of Reinforcement Learning

A few other key concepts in reinforcement learning include:

- Agent: This is our network that we train to interact within the environment
- Environment: This can be anything that gives us an observable state
- State: This is the current position of the agent
- Action: Based on the state, the agent determines the optimal action to take
- Reward: The environment returns a reward, which can be positive or negative



In the context of trading,

- The agent is the trader.
- The environment is the market.
- The state relates to statistics about the current market, which could be a number of different things like daily moving averages, high of day, volume, etc.
- The actions could include going long, short, closing a position, or choosing a particular asset for a portfolio.

- The reward could be profit, loss, volatility, Sharpe Ratio, or any performance metric of interests.

It is noteworthy that in the reinforcement learning for trading, the agent won't give us predictions like price target, market sentiment, and so on. Instead, the agent will take the predictions from other machine learning or statistical models and decide what the optimal actions are to maximize the cumulative expected reward.

7.2.1 Agent

The center piece of Agent is “policy.” Given an input, the policy returns the action that Agent will take. Policy is composed of a deep neural network. Here is an example of how to build a deep neural network for policy,

input

$$x_t = \begin{bmatrix} open_t \\ close_t \\ high_t \\ low_t \end{bmatrix} \rightarrow s_t = \begin{bmatrix} bullish_t \\ bearish_t \\ neutral_t \end{bmatrix}$$

The input may relate to some state that represents the bullish or bearish emotion of the market $s(t)$.

output

$$o_t = \begin{bmatrix} buy_t \\ sell_t \\ hold_t \end{bmatrix} = \begin{bmatrix} 0.63 \\ 0.35 \\ 0.02 \end{bmatrix} = \text{score of an action} \rightarrow a_t = buy_t$$

You take the action based on the probability obtained by the last neural network layer with the three neurons. Once the action is taken, there is a reward associated with it. Let's assume $a_t = buy_t$

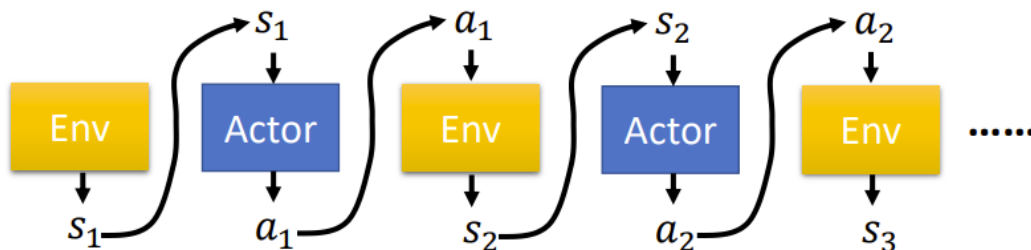
$$r_t = \$ 600$$

7.2.2 Episode and Reward

As time goes forward $t = 1 \dots T$, the agent directs you to take actions. When you take the last action $a(T)$, you receive the last reward $r(T)$. The environment determines that whether you are the winner or you “game-over (exhaust all your account balance).” The entire process is “so-called” an **Episode**. The total reward is:

$$R = \sum_{t=1}^T r_t$$

The goal of the agent is to obtain the maximum reward, given the environment. The above process can be described using the diagram below.



7.2.3 Trajectory

The process is thus a **Trajectory**.

$$\text{Trajectory: } \tau = \{s_1, a_1, s_2, a_2, \dots, s_T, a_T\}$$

There might be many trajectories possible, for each trajectory, we assume the probability to happen is $p_\theta(\tau)$, where θ is the parameter of policy neural network of the agent. When parameter values θ become available, you could calculate the probability to happen for a particular trajectory.

$$p_\theta(\tau) = p(s_1)p_\theta(a_1|s_1) \cdot p(s_2|s_1, a_1)p_\theta(a_2|s_2) \cdot p(s_3|s_2, a_2)p_\theta(a_3|s_3) \dots p(s_T|s_{T-1}, a_{T-1})p_\theta(a_T|s_T)$$

- $p(s_1)$ represent the probability that the environment output the state s_1 . It is determined by the environment.
- $p_\theta(a_1|s_1)$ represents the probability of taking the action a_1 based on the state s_1 . θ is estimated by the agent.
- $p(s_2|s_1, a_1)$ represent the probability of the state at $t = 2$. It is determined by the action at $t = 1$ (a_1) and related to the state at $t = 1$ (s_1).

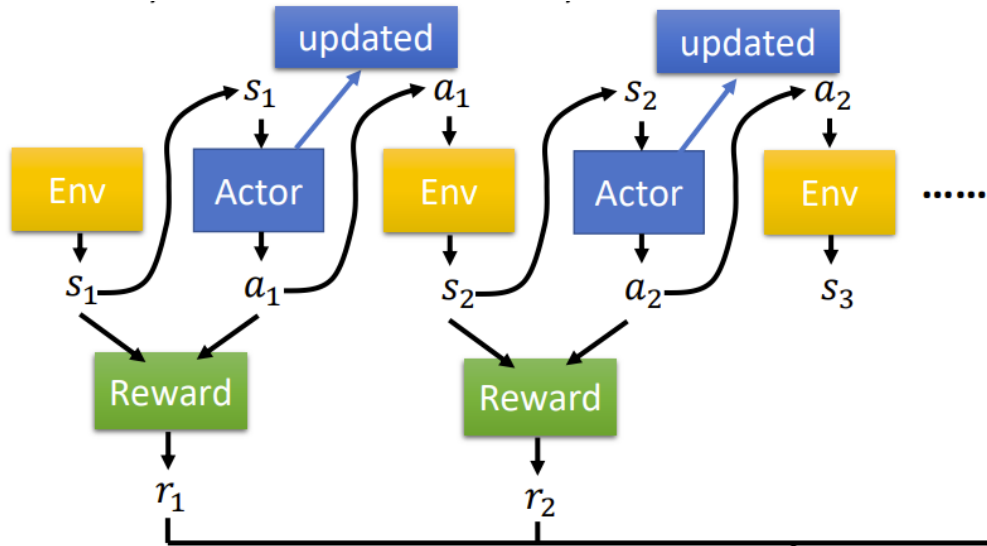
The process continues and can be summarized as follows:

$$p_\theta(\tau) = p(s_1) \prod_{t=1}^T p_\theta(a_t|s_t)p(s_{t+1}|s_t, a_t)$$

The trajectory is determined by two things. First, what is the rule of the environment, how does the state transits from s_t to s_{t+1} , e.g. what does the function look like, what are the parameters. The term $p(s_{t+1}|s_t, a_t)$ is mostly determined by the environment. Second, what action you are about to take at each timestamp given the state s_t . $p_\theta(a_t|s_t)$ is determined by you (agent). For different agents, each trajectory may receive different probabilities to happen from time to time.

7.2.4 Total Reward

The role of the reward function is to decide that given the state s_t at t and given the action a_t , what is the reward associated. r_t is a function of the state s_t and the action a_t .



The total reward of the trajectory τ over the entire episode is calculated by taking sum of all rewards over time.

$$R(\tau) = \sum_{t=1}^T r_t$$

7.3 Estimation

Our goal is to adjust the parameter values to maximize the value of $R(\tau)$. However, reward $R(\tau)$ is not a scalar. The key challenge here is that $R(\tau)$ is stochastic. This is because of two reasons. First, the agent may take different actions (based on probability) even when given the same state. Second, the environment may yield different states (stochastic) when fed with the same action and input. All best you can do is to calculate its expectation. Given the parameter value θ , the expected reward is:

$$\bar{R}_\theta = \sum_{n=1}^N R(\tau^n) p_\theta(\tau^n) = E_{\tau \sim p_\theta(\tau)}[R(\tau^n)]$$

The estimation takes the following steps:

First, we enumerate all possible trajectories τ and calculate its corresponding probability to happen $p_\theta(\tau)$. For instance, a good trading strategy has a higher probability of happening (surviving). A poor trading strategy has a lower probability of surviving.

Second, we calculate the total reward for each trajectory.

Third, we use the probability of happening for each trajectory $p_\theta(\tau)$ to multiply the reward of each trajectory $R(\tau^n)$. We then sum over across all N trajectories: $n = 1 \dots N$.

Finally, we use gradient ascent (or gradient descent on negation of expected rewards) to estimate parameter values.

7.3.1 Policy Gradient

$$\nabla \bar{R}_\theta = \sum_{n=1}^N R(\tau^n) \nabla p_\theta(\tau^n) = \sum_{n=1}^N R(\tau^n) p_\theta(\tau^n) \frac{\nabla p_\theta(\tau^n)}{p_\theta(\tau^n)}$$

$R(\tau)$ do not have to be differentiable. It can be a “black-box” or even non-model based. Given the fact that: $\nabla f(x) = f(x) \nabla \log f(x)$. We have

$$\nabla p_\theta(\tau) = p_\theta(\tau) \nabla \log p_\theta(\tau)$$

The above expected rewards can be expressed as

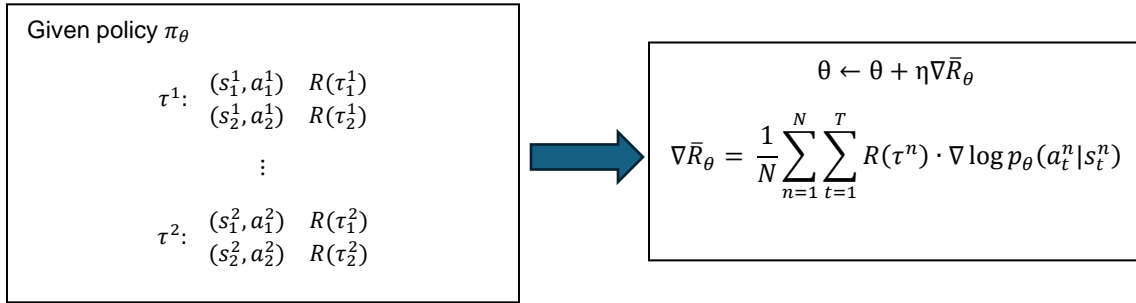
$$\nabla \bar{R}_\theta = \sum_{n=1}^N R(\tau^n) p_\theta(\tau^n) \nabla \log p_\theta(\tau^n)$$

$$\nabla \bar{R}_\theta = E_{\tau \sim p_\theta(\tau)}[R(\tau) \nabla \log p_\theta(\tau)] \approx \frac{1}{N} \sum_{n=1}^N R(\tau^n) \cdot \nabla \log p_\theta(\tau^n)$$

$$\nabla \bar{R}_\theta = \frac{1}{N} \sum_{n=1}^N \sum_{t=1}^T R(\tau^n) \cdot \nabla \log p_\theta(a_t^n | s_t^n)$$

Therefore, $\nabla \bar{R}_\theta = E_{\tau \sim p_\theta(\tau)}[R(\tau) \nabla \log p_\theta(\tau)]$. The following diagram shows how the policy gradient works.

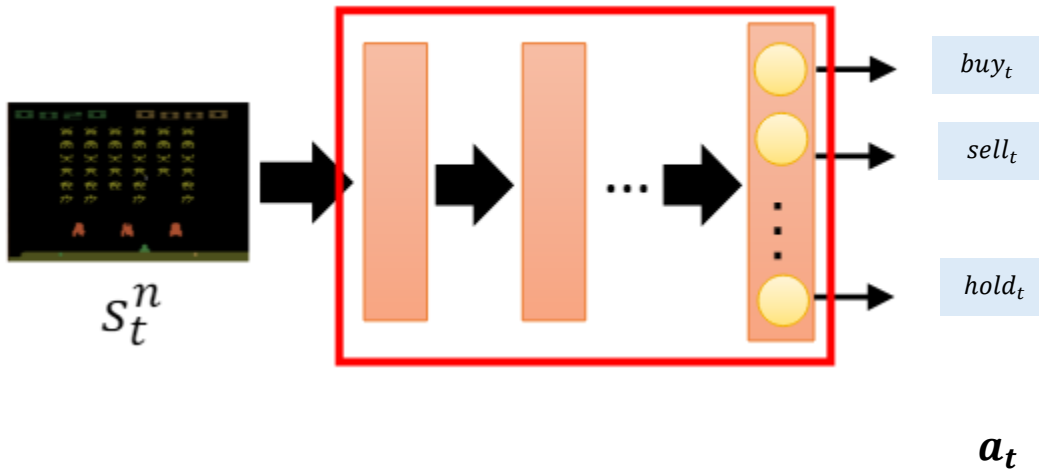
For the trajectory τ^1 , given state s_1^1 , we take the action a_1^1 and receive the reward $R(\tau_1^1)$. Given state s_2^1 , we take the action a_2^1 . for the trajectory τ^1 and receive the reward $R(\tau_2^1)$. For the trajectory τ^2 , given state s_1^2 , we take the action a_1^2 and receive the reward $R(\tau_1^2)$. Given state s_2^2 , we take the action a_2^2 and receive the reward $R(\tau_2^2)$.



- **Step 1:** Take each value pair of state and action (s_t^n, a_t^n) and calculate the log probability $p_\theta(a_t^n | s_t^n)$.
- **Step 2:** Calculate gradient of the log probability.
- **Step 3:** The gradient is then weighted by the reward of the trajectory, so that you can update the policy gradient $\nabla \bar{R}_\theta$ and hence the θ value.
- **Step 4:** Once the above steps are completed, the θ value is updated. You need to take sample from the data again and repeat Step 1 - 3. You need to repeatedly collect data throughout the parameter estimation process.

7.3.2 Parameter Estimation

It is similar to a classification problem. For each collection of state s_t^n action a_t^n and reward $R(\tau^n)$.



The states are information processed and extracted from your data, e.g. trading signals from the candlesticks, moving averages and oscillators. Actions are outcome behaviors indicated by states. During the training, you treat actions as the “ground truth.” For example, when you see the short moving averages cross the long moving averages from below, you take the action “buy.” Although the model may not assign the highest probability to action buy, the training process asks it to do so. Another example is when the market environment is bullish (rallies for several weeks) and the candlestick chart (state) shows a “doji,”

you take the action “sell.” Although the model may not assign the highest probability to the action sell, the training process asks it to do so.

In general classification problems, your objective function almost always uses “minimizing cross entropy.” Minimizing cross entropy is essentially the same as maximizing log likelihood. Your objective function looks like the following:

$$\theta = \arg \max \frac{1}{N} \sum_{n=1}^N \sum_{t=1}^T \log p_{\theta}(a_t^n | s_t^n)$$

Tensor-flow and PyTorch have built-components to calculate the log likelihood for you. All you need is to calculate the gradient as follows.

$$\frac{1}{N} \sum_{n=1}^N \sum_{t=1}^T \log p_{\theta}(a_t^n | s_t^n) \rightarrow \frac{1}{N} \sum_{n=1}^N \sum_{t=1}^T \nabla \log p_{\theta}(a_t^n | s_t^n)$$

In reinforcement learning problems, the only difference from the above is to **weight the loss function by the rewards**, which is to say, given the state today s_t^n and the action a_t^n today, how much you are supposed to get after the entire episode.

$$\frac{1}{N} \sum_{n=1}^N \sum_{t=1}^T R(\tau^n) \cdot \log p_{\theta}(a_t^n | s_t^n) \rightarrow \frac{1}{N} \sum_{n=1}^N \sum_{t=1}^T R(\tau^n) \cdot \nabla \log p_{\theta}(a_t^n | s_t^n)$$

7.3.3 Tip: Add a Baseline

The idea is given the state today s_t^n and the action a_t^n today - if the reward is large and positive, you need to increase $p_{\theta}(a_t^n | s_t^n)$ so as to increase the likelihood. If the reward is negative, you need to decrease $p_{\theta}(a_t^n | s_t^n)$ so as to decrease the likelihood.

However, it is technically possible to have $R(\tau^n)$ always stay positive (e.g. Gaming). You can add a baseline to maximize:

$$\nabla \bar{R}_{\theta} \approx \frac{1}{N} \sum_{n=1}^N \sum_{t=1}^T [R(\tau^n) - b] \cdot \nabla \log p_{\theta}(a_t^n | s_t^n)$$

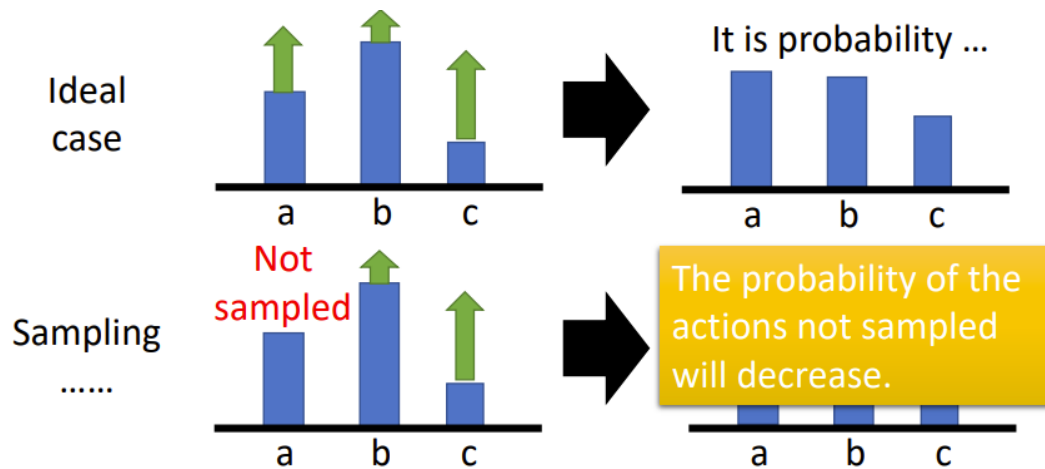
If $R(\tau^n) - b > 0$, the weight applied to the state-action pair is positive. If $R(\tau^n) - b \leq 0$, the weight applied to some state-action pair becomes negative. Only those state-action pairs that receive positive reward will have a higher probability of being chosen. For instance, b could take be mean rewards across all trajectories.

$$b = E[R(\tau)]$$

Parameter estimation is then updated using the following formula:



$$\theta \leftarrow \theta + \eta \nabla \bar{R}_{\theta}$$

The following example shows a problem that a positive-only reward specification may induce. Assume a , b and c are three actions you could take today. The height of the bar represents the log probability. The training aims to maximize the log probability as high as possible. However, the rewards applied to a , b and c are completely different. If action b has a large reward, the log probability of b jumps to the top. If action a has a small reward, the log probability of a shrinks to the smallest. Action a may never be sampled. The process requires the sum of log probability to be 0 (because sum of probability to be 1).



7.4 From on-policy to off-policy

The purpose of reinforcement learning is to learn about the agent.

Type	Content	Analogue
On-policy	The agent learned and the agent interacting with the environment is the same.	Naruto fights a battle by himself. 
Off-policy	The agent learned and the agent interacting with the environment is different.	Naruto fights a battle using shadow clones. 

7.4.1 On-policy -> Off-policy

On-Policy: Policy Gradient belongs to on-policy.

$$\nabla \bar{R}_\theta = E_{\tau \sim p_\theta(\tau)} [R(\tau) \nabla \log p_\theta(\tau)]$$

Given the trajectory τ and their corresponding reward, policy gradient updates the parameter value θ (policy) until the expected rewards reaches maximum. The policy learned by reinforcement learning and the policy interacts with the environment is the same policy. Policy gradient needs to use large amount of data sample. When using π_θ to collect data, once the policy θ is updated, both the probability of trajectory and the expected rewards of trajectory change. We have to resample the training data again so as to update the parameter (policy).

Goal: use the sample from $\pi_{\theta'}$ to train θ . We fix θ' and then use the θ' to collect data. We then update θ several times before we resample the training data again (we can re-use the sample data).

Importance Sampling

$$E_{x \sim p}[f(x)] \approx \frac{1}{N} \sum_{i=1}^N f(x^i)$$

x^i is a sample from distribution $p(x)$. However, if we do not know $p(x)$, we can only sample x^i from $q(x)$

$$E_{x \sim p}[f(x)] = \int_{-\infty}^{+\infty} f(x)p(x)dx = \int_{-\infty}^{+\infty} f(x) \frac{p(x)}{q(x)} q(x)dx = E_{x \sim q} \left[f(x) \frac{p(x)}{q(x)} \right]$$

The last term could be decomposed into two parts: the expectation of x^i drawn from distribution $q(x)$ and the importance weight $\frac{p(x)}{q(x)}$. The importance weight is to correct the difference between distribution $p(x)$ and distribution $q(x)$.

- It is noteworthy that $E_{x \sim p}[f(x)] = E_{x \sim q} \left[f(x) \frac{p(x)}{q(x)} \right]$ does not necessarily imply:

$$Var_{x \sim p}[f(x)] = Var_{x \sim q} \left[f(x) \frac{p(x)}{q(x)} \right]$$

- When $p(x)$ and $q(x)$ are far away, $E_{x \sim p}[f(x)]$ may fall negative.

$$E_{x \sim p}[f(x)] = E_{x \sim q} \left[f(x) \frac{p(x)}{q(x)} \right]$$

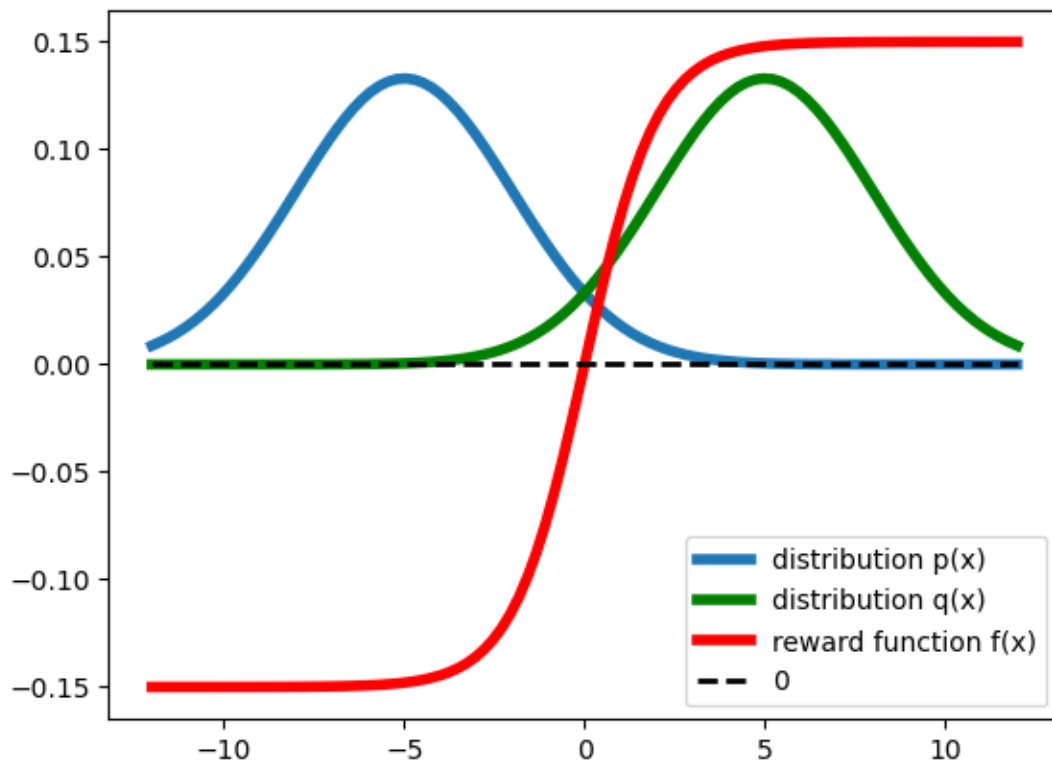
Note 1: $Var_{x \sim p}[f(x)]$ and $Var_{x \sim q} \left[f(x) \frac{p(x)}{q(x)} \right]$

According to $VAR[X] = E[X^2] - (E[X])^2$, we have $E_{x \sim p}[f(x)] = E_{x \sim q} \left[f(x) \frac{p(x)}{q(x)} \right]$

$$Var_{x \sim p}[f(x)] = E_{x \sim p}[f(x)^2] - \{E_{x \sim p}[f(x)]\}^2$$

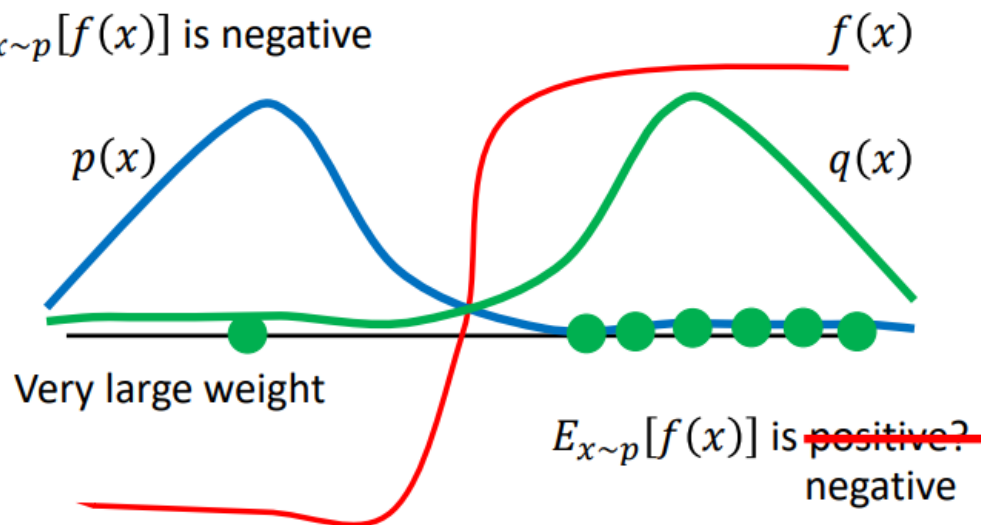
$$Var_{x \sim q} \left[f(x) \frac{p(x)}{q(x)} \right] = E_{x \sim q} \left[\left(f(x) \frac{p(x)}{q(x)} \right)^2 \right] - \left(E_{x \sim q} \left[f(x) \frac{p(x)}{q(x)} \right] \right)^2 = E_{x \sim q} \left[f(x)^2 \left(\frac{p(x)}{q(x)} \right)^2 \right] - \{E_{x \sim p}[f(x)]\}^2$$

Note 2: $E_{x \sim p}[f(x)]$ can fall negative in some circumstances.



When $p(x)$ and $q(x)$ are very different. When most samples are drawn from the right tail of $q(x)$, $\frac{p(x)}{q(x)}$ is close to 0. $f(x) \frac{p(x)}{q(x)}$ stays positive and small. However, when some samples are drawn from the left tail of $q(x)$, $\frac{p(x)}{q(x)}$ is large. $f(x) \frac{p(x)}{q(x)}$ falls far below 0. It leads to the expected rewards $E_{x \sim p}[f(x)] = E_{x \sim q}\left[f(x) \frac{p(x)}{q(x)}\right] < 0$

$E_{x \sim p}[f(x)]$ is negative



7.4.2 Off-Policy

On-Policy Design

$$\nabla \bar{R}_\theta = E_{\tau \sim p_\theta(\tau)} [R(\tau) \nabla \log p_\theta(\tau)]$$

- Use π_θ to collect data, when θ is updated, we have to sample the data again so as to update the parameter (policy).
- Goal: use the sample from $\pi_{\theta'}$ to train θ . θ' is fixed, so we can re-use the sample data.

Off-Policy Design

$$\nabla \bar{R}_\theta = E_{\tau \sim p_{\theta'}(\tau)} \left[\frac{p_\theta(\tau)}{p_{\theta'}(\tau)} R(\tau) \nabla \log p_\theta(\tau) \right]$$

In on-policy, we use θ to interact with the environment and sample the trajectory τ so as to calculate the expected rewards $R(\tau) \nabla \log p_\theta(\tau)$.

In off-policy, we stop the direct interaction between θ and the environment. We adopt another policy θ' instead. The role of θ' is demonstration - θ' can tell θ what would happen when it interacts with the environment. The trajectory τ is now sampled from distribution $p_{\theta'}(\tau)$ instead of $p_\theta(\tau)$.

Using the idea of importance sampling,

$$E_{x \sim p}[f(x)] = E_{x \sim q} \left[f(x) \frac{p(x)}{q(x)} \right]$$

The importance weight is $\frac{p_\theta(\tau)}{p_{\theta'}(\tau)}$. It is the probability of the trajectory calculated using θ divided by is the probability of the trajectory calculated using θ' .

- We sample data from θ' .
- Use the data to train θ many times – until it becomes necessary for θ' resampling the data.

Gradient for update is

$$J^{\theta'}(\theta) = E_{(s_t, a_t) \sim \pi_\theta} [A^\theta(s_t, a_t) \nabla \log p_\theta(a_t^n | s_t^n)]$$

$$J^{\theta'}(\theta) = E_{(s_t, a_t) \sim \pi_{\theta'}} \left[\frac{p_\theta(s_t, a_t)}{p_{\theta'}(s_t, a_t)} A^{\theta'}(s_t, a_t) \nabla \log p_\theta(a_t^n | s_t^n) \right]$$

$$J^{\theta'}(\theta) = E_{(s_t, a_t) \sim \pi_{\theta'}} \left[\frac{p_\theta(a_t | s_t)}{p_{\theta'}(a_t | s_t)} \frac{p_\theta(s_t)}{p_{\theta'}(s_t)} A^{\theta'}(s_t, a_t) \nabla \log p_\theta(a_t^n | s_t^n) \right]$$

$A^\theta(s_t, a_t)$ represents the advantage, which is reward minus baseline.

$p_\theta(s_t)$ and $p_{\theta'}(s_t)$ is very close to one another by configuration. Given the fact that: $\nabla f(x) = f(x) \nabla \log f(x)$. We have obtained a new objective function.

$$J^{\theta'}(\theta) = E_{(s_t, a_t) \sim \pi_{\theta'}} \left[\frac{p_\theta(a_t | s_t)}{p_{\theta'}(a_t | s_t)} A^{\theta'}(s_t, a_t) \right]$$

Up to here, we replaced the on-policy with the off-policy.

7.5 Summary

Deep reinforcement learning puts an agent into a new environment where it learns to take the best decisions based on the circumstances of each state it encounters. The goal of the agent is to collect information about the environment in order to make an informed decision. It does this by testing how its actions influence its own rewards in the environment. The more frequently the agent interacts with the environment, the faster it learns how to maximize its expected rewards. The main difference between deep reinforcement and other types of machine learning algorithms is that the DRL agents are given a high degree of freedom when it comes to the learning process.

Reference

https://github.com/AI4Finance-Foundation/FinRL-Trading/tree/master/old_repo_ensemble_strategy

<https://www.mlq.ai/deep-reinforcement-learning-trading-strategies-automl/>

https://openfin.engineering.columbia.edu/sites/default/files/content/publications/neurips_2018.pdf

<https://www.sciencedirect.com/science/article/abs/pii/S0957417423033031>