Report of MVIG Training, Deep Reinforcement Learning

Yongxi Huang
SJTU
huangyongxi@sjtu.edu.cn

Abstract

*By reading **Reinforcement Learning: An Introduction** and learning from the online lecture by David Silver, I have learned some concepts and inplementation of reinforcement learning. In this report, I want to do a summary and combing of the knowledge learned. I also have trained some models. However, because of the limited time, I can't trained them totally. And I'm still working on the code these days.*

1. Introduction

To do a summary of the knowledge, first I want to combing some basic concept of reinforcement learning. And then introduce some classical algorithms solving RL problem such as DP, MC and Q-learning. Finally, I want to give my understanding of some approches introduced in some paper. However, I may have some misunderstanding about RL or make some mistakes. I hope anyone reading this report can correct them for me.

2. Basic concepts

2.1. Reinforcement Learning

Reinforcement learning is a kind of unsupervised learning method. In deep-learning, if we want to train the AI to do something, we must give it some correct answer like annotations and labels, and it can learning from the data by minimize the loss function. That is some kind of supervised learning, which means learning from some given labeled result. It have some feedback indicates the correct action to take, independently of the action actually taken

But in reinforcement learning, there is no supervisor. The agent can learn only from the reward signal it obtains. Besides, the reward feedback can be delayed and the actions of agents can change the environment and the reward. In summary, reinforcement learning agents learn by trail-and-error.

Reinforcement learning cant work on any kinds of environments. The agents always learn from its current state and the predicted future state. That means the current state must completely characterizes the process, otherwise the history of the process may make the learning process costly. In other words, the process of a reinforcement learning must has the Markov property:

$$\mathbb{P} = [S_{t+1}|S_t] = \mathbb{P} = [S_{t+1}|S_1, S_2, \ldots, S_t]$$

In short, the Markov property means the future is independent of the past given the present.

The environment with Markov states that the reinforcement learning agents work on is called Markov Decision Process (MDP). Its a Markov process with reward and actions. In fact, in Reinforcement Learning: An Introduction , reinforcement learning is regarded as any method that is suited to solving MDP problem. A MDP can be decision as ¡S,A,P,R,r¿. There are several useful concept about MDP.

2.2. States

In my opinion, this is a fundamental concept of MDP. A state contains all relevant information for the agent to make decision. However, when we talk about states, we are always talk about agent states instead of environment states. That means the states dont contains all the information of the environment, but the information that can be obtained by the agent. Besides, the states of a MDP have the Markov property. And the states can have different form.

2.3. Reward

The reward Rt is a scalar feedback based on the state or the action of the agent. The goal of the agent is to maximize the expected cumulative reward. The reward of a specific state can be defined as:

$$R_s = \mathbb{E}[R_{t+1}|S_t = s]$$

As we always care about the cumulative reward of a process, we can define the return of a process:

$$G_t = Rt + 1 + \gamma R_{t+2} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

In that case, the goal of agent is to maximize the expected return.

2.4. Policy

Policy is a map from states of environment to the actions of agent. It define the behavior of an agent. It is defined as:

$$\pi(a|s) = \mathbb{P}[A_t = a|S_t = s]$$

Just as its definition, the policy $\pi : S \to A$ has nothing to do with time. In practice, sometimes we estimate the future return according to the policy taken, which is called on-policy. And sometimes the policy doesn't influence the update of estimation of future return(policy evaluation), which is called off-policy.

2.5. Value function

Once we have reward and return, we can define the value function to evaluate a state or action. Its a prediction of future reward which can be defined as:

$$v_\pi = \mathbb{E}[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots | S_t = s]$$

Value function is a function about states under the policy π. The agent choose different actions based on the value function. The value function defined above only evaluate the quality of states, which is called the state-value function. If we take the action further into account, we can define the action-value function:

$$q_\pi(s, a) = \mathbb{E}_\pi[G_t|S_t = s, A_t = a]$$

It not only evaluate the quality of a state, but also the action taken on this state. In short, value function is the expected return. Find a proper way to evaluate value function is an vital part of reinforcement learning. After define these concept, the goal of reinforcement learning can be seen as find a optimal policy p_i^* that can maximize the value function.

$$\pi^* = argmax_\pi V^\pi(s), \forall s$$

3. Basic Algorithms

3.1. Dynamic Programming

However, DP is not only a strategy to solve the reinforcement learning problem, but also an idea of other strategy. I think the reason why we can use Dynamic Programming to solve this problem is also the Markov property of MDP. Because the state-value function or action-value function have nothing to do with the history of a process, we can decomposed them using the Bellman Expectation Equation:

$$\begin{aligned} v_\pi(s) =& \mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1})|S_t = s] \\ q_\pi(s, a) =& \mathbb{E}_{\pi\pi}[R_{t+1} + \gamma q_\pi(S_{t+1}, A_{t+1})|S_t = s, A_t = a] \end{aligned}$$

3.1.1 Policy Evaluation

According to Bellman Expectation, the value function can be decomposed into immediate reward and the value of successor states. When given the policy π, we can calculate the value function of a specific state using Bellman Expectation Equation. We can see this process as a policy evaluation.

Using DP to evaluate a policy is shown in Algorithm 1.

Algorithm 1: Policy evaluation of DP

input : An policy π
output: The estimated value function v_π

while $\Delta > \theta$*(a small positive number)* **do**
  $\Delta \leftarrow 0$
  for *each* $s \in S$ **do**
    $temp \leftarrow v(s)$
    $v(s) \leftarrow \sum_a \pi(a|s) \sum_{s'} p(s'|s, a)[r(s, a, s') + \gamma v(s')]$
    $\Delta \leftarrow max(\Delta, |temp - v(s)|)$
Output $v \approx v_\pi$

3.1.2 Policy improvement

However, the algorithm above can only estimate the value function of a policy π. In other words, it can only do policy evaluation. If we want to find an optimal policy, we need to do policy improvement. As we can calculate the value function of an existing policy π according to the algorithm above, a simple way to find the optimal policy is to change the action of a specific state and see whether it can improve the value function. The action-value can be calculated by

$$\begin{aligned} q_\pi &= \mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1})|S_t = s.A_t = a] \\ &= \sum_{s',r} p(s', r|s, a)[r + \gamma v_\pi(s')] \end{aligned}$$

If $q_\pi(s, a)$ is greater than $v_\pi(s)$, its better to select a in s. and we can get a new greedy policy π' by

$$\begin{aligned} \pi'(s) &= argmax_a q_\pi(s, a) \\ &= argmax_a \mathbb{E}[R_{t+1} + \gamma v_\pi(S_{t+1})|S_t = s, A_t = a] \\ &= argmax_a \sum_{s',r} p(s', r|a, a)[r + \gamma v_\pi(s')] \end{aligned}$$

This policy improvement algorithm is shown in Algorithm 2.

As we improve the policy iteratively in this algorithm, its called policy iteration. Another way to find the optimal

Algorithm 2: Policy iteration of DP

input : Old policy π, value function V.
output: Optimal policy

$policy - stable \leftarrow true$
for $eachs \in S$ **do**
  $old - action \leftarrow \pi(s)$
  $\pi(s) \leftarrow argmax_a \sum_{s',r} p(s', r|a, a)[r + \gamma V(s')$
  if $old - action \neq \pi(s)$ **then**
    $policy - stable \leftarrow false, break$
if *policy-stable* **then**
  return $V \approx v_* \ and \ \pi \approx \pi_\pi *$
else
  go to policy evaluation

Algorithm 3: Value iteration of DP

Initialize array V arbitrarily
while $\Delta > \theta$*(a small positive number)* **do**
  $\Delta \leftarrow 0$
  for *each* $s \in S$ **do**
    $v \leftarrow V(s)$
    $V(s) \leftarrow max_a \sum_{s',r} p(s', r|s, a)[r + \gamma V(s')]$
    $\Delta \leftarrow max(\Delta, |v - V(s)|)$
Output a deterministic policy, $\pi \approx \pi_\pi *, suchthat$
$\pi(s) = argmax_a \sim_{s'r} p(s', r|s, a)[r + \gamma V(s')]$

policy is value iteration. We can see value iteration as combining the policy improvement and policy evaluation.

However DP is not really practical. First, the full backup operation used to update value requires high computation. And DP is based on all possible next states rather than on a sample next states, which means it is model based. So the classical DP algorithms are of limited utility in reinforcement leaning both because of their assumption of a perfect model and because of their great computation expense. But DP is the base of other algorithms. An important concept created by DP, for example, is generalized policy iteration (GPI) which means letting policy evaluation and policy improvement processes interact. In **Reinforcement Learning: An Introduction**, it describe these two process as this:

One process takes the policy as given and performs some form of policy evaluation, changing the value function to be more like the true value function for the policy. The other process takes the value function as given and performs some form of policy improvement, changing the policy to make it better, assuming that the value function is its value function.

And in my opinion, GPI is essential in many other methods solving reinforcement learning problems.

3.2. Monte Carlo Methods (MC)

A big improvement from DP to MC is that MC is model free which means we dont assume complete knowledge of the environment. However, model-free doesnt mean that there is no model. The model in MC just need to generate the sample transitions without the complete probability distributions. The basic idea of MC is value = mean return. And mean return can be estimated by empirical mean return. And the idea of GPI is just like DP. Besides, the value function can be updated incrementally.

$$N(S_t) \leftarrow N(S_t) + 1$$
$$V(S_t) \leftarrow V(S_t) + \frac{1}{N(S_t)}(G_t - V(S_t))$$

This formula uses a method that always used to estimate some target:

$$NewEstimate \leftarrow OldEstimate + StepSize[Target - OldEstimate]$$

The $Target - OldEstimate$ is called as error, which is reduced by taking a step toward the Target. And when it comes to policy improvement, it proceeds according to GPI defined in DP. And no model is needed as the policy is made with respect to the current value function. But here is a delemma between exploration and evaluation in MC method. To solve it, we can use $\varepsilon - greedy$ which means most of the time the agent chooses the action with maximal estimated action value, and with probability ε it chooses other actions. A MC algorithm is shown in Algorithm 4.

The on-policy in fact makes a compromise that it learns action values for both optimal policy and the near-optimal policy. Another solution for the evaluation and exploration dilemma is off-policy learning. It use two different policy for evaluation (target policy) and exploration (behavior policy). As these two policy is separated, the target policy may be deterministic while the behavior policy can continue to sample all possible actions.

A drawback of MC is, as we can see, the updating of policy happens after each episode. So MC can only solve episodic tasks but not continuing tasks. An idea is to update the policy online. So here comes TD method.

4. TD learning

Temporal-Difference Learning (TD learning) can be seen as a combination of MC and DP. In MC, the value function is updated until the return following the visit is known. The constant- MC for example, the increment is calculated

Algorithm 4: On-policy MC control

Initialize
  $Q(s,a) \leftarrow arbitrary$
  $Returns(s,a) \leftarrow emptylist$
  $\pi(a|s) \leftarrow anarbitrary\varepsilon - softpolicy$
while *true* **do**
  Generate an episode using π
  for *each pair s,a appearing in the episode* **do**
    $G \leftarrow$ following the first occurrence of s,a
    Append G to Returns(s,a)
    $Q(s,a) \leftarrow average(Returns(s,a))$
  for *rach s in the episoce* **do**
    $A_* \leftarrow argmax_a Q(s,a)$
    for *all* $a \in A(s)$ **do**
      $\pi(a|s) = 1 - \varepsilon + \varepsilon/|A(s)|$, if $a = A^*$
      $\pi(a|s)\varepsilon/|A(s)|$, if $a \neq A^*$

through

$$V(S_t) \leftarrow V(S_t) + \alpha[G_t - V(S_t)]$$

But TD methods need wait only until the next time step rather than the next episode. The value prediction of TD(0) is

$$V(S_t) \leftarrow V(S_t) + \alpha[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)]$$

The target for the TD update is $R_{t+1} + \gamma V(S_{t+1})$ rather than G_t. This kind of updating method is called bootstrapping method. And its just what used in DP. In summary, MC uses a sample return as the real expected return is unknown. And DP use a target at the current estimate as the new estimate is unknown. The TD target combines DP and MC because it samples the expected values and it uses the current estimate V instead of the true v_π. A simple TD prediction algorithm TD(0) is shown as follow:

From another perspective, TD and MC both use sample backups while DP uses full backups. And we can see the update in TD(0) as updating by error $[R + \gamma V(S') - V(S)]$ which measuring the difference between the estimated value of S_t and the better estimate $R_{t+1} + V(S_{t+1})$. This error is called TD error.

Another TD prediction method is batch updating which means although some states have been visited, the value function is changed only once by the sum of the increments after a batch of error. Many another RL algorithms use TD prediction.

4.1. Sarsa

Sarsa is an algorithm using TD prediction. In order improve the policy, it considers action value rather than state

Algorithm 5: TD(0) for estimating v_π

input : Policy π.
output: Value evaluation v_π of policy π

Initialize V(s) arbitrarily
while *true* **do**
  Generate an episode S using π
  for *each step of episode* **do**
    $A \leftarrow actionfromS$
    $TakeactionAandobserveR, S'$
    $V(S) \leftarrow V(S) + \alpha[R + \gamma V(S') - V(S)]$
    $S \leftarrow S'$

value, which means its on policy. Also, its a GPI process. In order to use TD(0) to update action value rather than state value, we can change the formula into this: $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)]$. A sarsa algorithm using $\varepsilon - greedy$ is shown in Algorithm 6.

Algorithm 6: Sarsa, On-policy TD control

Initialize Q(s,a) arbitrarily
for *each episode* **do**
  Initializa S
  Choose A from S using policy derived from Q (e.g. $\varepsilon - greedy$)
  for *each step of episode* **do**
    Take action A, observe R, S'
    Choose A' from S' using policy derived from Q (e.g. $\varepsilon - greedy$)
    $Q(S,A) \leftarrow Q(S,A) + \alpha[R + \gamma Q(S', A') - Q(S,A)]$
    $S \leftarrow S'; A \leftarrow A'$

4.2. Q-learning

Q-learning also use the idea of TD prediction, but its an off-policy TD control method. It is defined by

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma max_a Q(S_{t+1}, a) - Q(S_t, A_t)]$$

In that case, the updating of the action value function has nothing to do with the action taken before. So its off-policy. The algorithm is shown in Algorithm 7.

We can consider Q as the knowledge learned. And the optimal can be derived greedily from Q.

However, there is still many basic knowledge about reinforcement learning I dont understand as I just finish half of the reference book Reinforcement Learning: An Introduction. I will keep working on it. But for now, I want to

Algorithm 7: Q-learning, Off-policy TD control

Initialize Q(s,a) arbitrarily
for *each episode* **do**
  Initializa S
  for *each step of episode* **do**
    Choose A from S using policy derived from Q (e.g. $\varepsilon - greedy$)
    Take action A, observe R, S'
    $Q(S,A) \leftarrow Q(S,A) + \alpha[R + \gamma max_a Q(S',a) - Q(S,A)]$
    $S \leftarrow S'$

talk about something I learn from some paper. However, its really helpful for understanding the paper to learn some basic concept and fundamental algorithm of reinforcement learning.

5. Paper reading

5.1. DQN

In Playing Atari with Deep Reinforcement Learning , an approach using neural network is introduced. It uses the idea of Q-learning, while using convolutional neural network to estimate the reward. It is model free as we just input reward, terminal signals and possible actions. As the environment of the agent cant be described in a single screen shot, the method considers a sequence of actions and observations as one state. And the reward is depended on the scores of games.

Just as other reinforcement learning problem, we should do policy evaluation and policy improvement to find the optimal policy. For policy evaluation, a simple idea is using the value iteration just as classical Q-learning like Qi+1(s; a) =E[r + maxa0 Qi(s0; a0)js; a]. But in this paper, it use a function approximator to estimate the action value function. More specifically, it use a neural network to approximate it, which called Q-network. And the Q-network can be trained by minimizing the loss function $L_i(\theta_i) = \mathbb{E}_{s,a\sim\rho(\cdot)}[(y_i - Q(s,a;\theta_i))^2]$ using stochastic gradient descent, where $y_i = \mathbb{E}_{s'\sim\varepsilon}[r + \gamma max'_a Q(s',a';\theta_{i-1})|s,a]$.

In that case, y_i can be regarded as the estimate value function and the loss function can measure the difference between the estimation and the knowledge it has learned. As for $\rho(s;a)$, I think its a kind of behavior distribution like $\epsilon - greedy$ method. However, the target y_i here is different from the target in deep learning, which is fixed before training. y_i is changing in the training process.

5.2. Double DQN

The basic idea of Double DQN is that DQN uses the same values to select and evaluate an action, which resulting in overoptimistic value estimates. And the idea of Double Q-learning is to reduce overestimations by decomposing the max operation in the target into actionn selection and action evaluation.

There are two set of weights θ and θ' in Double DQN. One is used to determine the greedy policy and the other to determine its value. If we write the target in DQN as:

$$Y_t^Q = R_{t+1} + \gamma Q(S_{t+1}, argmax_a Q(S_{t+1}, a; \theta_t); \theta_t)$$

the target in Double Q-learning can be writen as

$$Y_t^{DoubleQ} \equiv R_{t+1} + \gamma Q(S_{t+1}, argmax_a Q(S_{t+1}, a; \theta_t); \theta'_t)$$

In that case, the weight θ_t is used for action selection and the weight θ'_t is used for action evaluation.

5.3. Dueling Network

In Dueling Network Architectures for Deep Reinforcement Learning, it creates a new architecture of neural network that suitable for RL. The Dueling network has two streams to separately estimate (scalar) state-value and the advantages for each action.

A basic idea of Dueling Network is that in some states, it's important to know which action to take while in many other states the choice of action has no repercussion on what happens.

In dueling, there is an advantage function besides state value function and state-action value function. It is defined as

$$A^\pi(s,a) = Q^\pi(s,a) - V^\pi(s)$$

The value function V measures how good it is to be in a state s and the Q function measures the value of choosing a particular action in this state. In that case, the advantage function can be seen as a relative measure of the importance of each action.

As for inplementation, dueling network uses two streams of fully connected layers following the convolutional layers.

References

[1] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. Openai gym, 2016.

[2] carpedm20. deep-rl-tensorflow. https://github.com/carpedm20/deep-rl-tensorflow.

[3] D. S. Hado van Hasselt, Arthur Guez. Deep reinforcement learning with double q-learning, 2016.

[4] S. D. G. A. A. I. W. D. . R. M. Mnih V., Kavukcuoglu K. Playing atari with deep reinforcement learning, 2013.

[5] V. Mnih1, K. Kavukcuoglu1, and D. S. et al. Human-level control through deep reinforcement learning, 2015.

[6] M. Riedmiller. Neural fitted q iteration - first experiences with a data efficient neural reinforcement learning method, 2005.

[7] R. S. Sutton and A. G. Barto. Reinforcement learning: An introduction, 2016.

[8] M. M. e. a. Volodymyr Mnih, Adri Puigdomnech Badia. Asynchronous methods for deep reinforcement learning, 2016.

[9] M. H. e. a. Ziyu Wang, Tom Schaul. Dueling network architectures for deep reinforcement learning, 2015.