REINFORCEMENT LEARNING — THEORY AND PURPOSE

1. Introduction

The purpose of this project is to use reinforcement learning to train an *agent* to play blackjack and investigate the learning capabilities of the framework. We implement two different representations of the state space, one of which, albeit intuitive, does not satisfy the model's stationarity assumptions. In comparing the two different representations we investigate the importance of the stationarity assumption.

Reinforcement Learning

Reinforcement learning is, as many things are in machine learning, both the problems and solutions to a specific domain. The problem is how an *agent* ought to act in an *environment* with imperfect information and randomness, but with feedback. The solutions are many, but most common is the Markov decision process approach.

Markov Decision Process. Markov decision processes (MDP) is the formalism which allows us to reason about reinforcement learning in a mathematical way, and all standard reinforcement learning problems are formulated in a Markov decision process framework. In what follows we give the definition of a stationary Markov decision process.

The building blocks of a Markov decision process consist of

(i) S a finite space of *states*
(ii) A a finite space of *actions*
(iii) R a finite space of *rewards*
(iv) $P_a(s, s')$ a transition probability function defined for all $(s, a, s') \in S \times A \times S$
(v) $r(s, a)$ the immediate or expected immediate reward of taking action a in state s.
(vi) $\gamma \in [0, 1]$, the discount factor (here always equal to 1).

We define $S_t \in S$ to be the state of the system at time t; $A_t \in A$ is the (potentially random) action taken at time t, and $R_t \in R$ is the immediate reward at time t. The *history* H_t of the system up to time t is the random vector given by $(S_0, A_0, S_1, A_1, \ldots, S_{t-1}, A_{t-1}, S_t)$. An important concept going further is that of a policy: a policy $\pi = (\pi_0, \pi_1, \ldots)$ is a sequence of decision rules, where π_n, for each history of the process up to time n, determines a probability measure over A.

Now, we call the process $\{(S_t, Y_t, R_{t+1}),\ t \geq 0\}$ a Markov decision process if

$$\begin{aligned}\mathbb{P}(S_t = s | S_{t-1} = s_{t-1}, A_{t-1} = a_{t-1}, \ldots, S_0 = s_0, A_0 = a_0) \\ = \mathbb{P}(S_t = s | S_{t-1} = s_{t-1}, A_{t-1} = a_{t-1}) = P_a(s_{t-1}, s) \quad (1)\end{aligned}$$

and R_{t+1} given s_t, a_t is independent of H_t, with $\mathbb{E}(R_{t+1}|H_t) = r(S_t, A_t)$.

At first glance expression (1) makes sense, but there are subtle details which need to be addressed. For instance, we have said nothing about the process $\{A_t\}$ and therefore we should be careful defining conditional probabilities containing it. However, if $\{A_t\}$ is governed by a policy $\pi = (\pi_0, \pi_1, \ldots)$ then we can define the above probability. In order for the tuple to be a Markov decision process we formally require that (1) holds for all policies.

The goal of reinforcement learning is to take a MDP and choose a policy which maximizes expected (discounted) rewards. This can seem hopeless since policy decisions in general depends

on the whole history of the chain. However, it can be shown that for finite stationary MDPs (reward and transition probabilities does not depend on t) this is equivalent to maximizing rewards over policies which take into account only the current state of the process and which maps this to a single action, i.e. $\pi_n(H_n) = \pi(X_t) \in A$ — this is know as a *stationary Markov policy*, see Reference. This is something that make the optimizing problem considerably easier, and allows the *Q-learning* algorithm to converge to an optimal solution. Hence, from here on out we consider only stationary Markov policies.

For a given policy π define the *value function*

$$V^{\pi}(s) = \mathbb{E}_{\pi}(\sum_{n=0}^{\infty} R_{t+1}|X_0 = s)$$

i.e. the expected reward obtained when following policy π. If s_t is an *terminal* state — a state from which the process never leaves — we take $R_{t+1} = 0$, which implies that $V^{\pi}(s) = 0$ if s is terminal.

We say that a policy π^* is optimal if

$$V^{\pi^*}(s) \geq V^{\pi}(s) \qquad \forall s \text{ and policies } \pi.$$

It can be shown that for finite Markov decision processes such a policy exists uniquely. Under the optimal policy π^* let $V^*(s) = V^{\pi^*}(s)$. Say we know $V^*(s)$ for all s, can this be used to calculate an optimal policy? The answer is yes. Note by the Markov property of the process we must have

$$V^*(s) = r(s, \pi^*(s)) + \sum_{s'} P_{\pi^*}(s, s')V^*(s') = \max_{a \in A}(r(s, a) + \sum_{s'} P_a(s, s')V^*(s')).$$

Hence, $\pi^*(s)$ is given by the action a which achieves the above maximum. So if we can find a way to calculate the optimal value function we get the optimal strategy. In small state space the value function can actually be calculated but in large spaces approximation is the best one can hope for.

Q-Learning. Q-learning is an algorithm which can be used to approximate the optimal value function and corresponding strategy. Here we will assume that the underlying MDP is terminal.

In order to specify the algorithm we need some definitions. Define the state-action function as

$$Q^{\pi}(s, a) = r(s, a) + \sum_{s'} P_a(s, s')V^{\pi}(s').$$

That is the expected reward of starting in state s taking action a and then following the policy π. For terminal state we set $Q = 0$. Furthermore, define the optimal state-action function as

$$Q^*(s, a) = (r(s, a) + \sum_{s'} P_a(s, s')V^*(s'))$$

i.e. the expected reward of being in state s taking action a and then following the optimal policy π^*. We see that $\max_{a \in A} Q^*(s, a) = V^*(s)$, and with the same argument as above $\pi^*(s) = \max_{a \in A} Q^*(s, a)$.

We can now specify the algorithm:

(i) Initialize $Q(s, a)$ for all s, a, learning rate $\alpha \in (0, 1)$, and decide on some policy e.g. ϵ-greedy.

(ii) Repeat for each N episodes (start $\rightarrow$ final state):
While s_t not terminal: $Q(s_t, a_t) \leftarrow (1 - \alpha)Q(s_t, a_t) + \alpha_t(r_t + \max_{a \in A} Q(s_{t+1}, a))$

Again, this is known as Q-learning, and it will produce estimates which converges to Q^*

Reinforcement Learning for Blackjack

In order to test this reinforcement learning framework we have chosen to work with the simplest form of Black Jack. We assume the following setup:

- One player against the dealer, with player staking one unit on each hand;
- infinite number of decks;
- two actions possible: ask for another card, or stay;
- cards 2-10 counts as their numerical value, suites counts as 10, and ace counts as either 1 or 11 depending on whichever is best.

The goal of the player is to beat the dealer in one of the following ways

- Get 21 points on the first two cards, knows as a blackjack, without a dealer blackjack, **net profit**: 1.5 times stake;
- Reach a final score higher than the dealer without exceeding 21, **net profit**: stake;
- Dealer gets points exceeding 21 and player does not, **net profit**: stake.

The house always plays according to the same strategy: draw cards until it has a card sum greater than or equal to 17. The game starts with the dealer giving the player two cards (visible) and himself two cards (one visible, one hidden). The player is then allowed to take actions until done, after which the dealer follows his strategy until done. If player's points exceeds 21 his stake is lost regardless of dealers outcome (this is where the house edge comes from); if player's points equals dealer's points this is a push and stake is returned to player; otherwise payout is made according to above rules.

As a Markov Decision Process. Next we fit the above situation in to the Markov decision process framework. We do it in two ways: one with a large state space such that all MDP model assumptions is satisfied; one with a smaller state space, where the resulting process is no longer stationary. In both cases actions and rewards are as specified above, and we need only specify the state space in order to have the model.

Stationary Markov Decision Process. We begin by noting that the color and suite of the cards does not matter, only their numerical values. We use this as their identifier, with aces equal to 1. We use the following representation of the state space

$$S = \{(s_{p1}, \dots, s_{p10}, \Sigma_d)\ s_{pi}, \in \{1, 2, \dots, 21\}, \Sigma_d \in \{1, 2, \dots, 10\}\}.$$

Hence, an element of the state space is a vector of length 11, where the 10 first elements indicates the number of each card the player is holding, and the last element indicates which card the dealer is holding. Note that a player can never hold more than 21 cards — this would be 21 aces and after that the game necessarily terminates. At first glance this state space seems enormous, and while that is true the *effective* state space is much smaller, i.e. the states which the system realistically will visit is relatively small. Also, with this state representation it is always possible to determine if a state is terminal — we need only check if the player or dealer is bust, or if dealer's card sum is exceeding 17. The rewards $R_1, R_2, \dots$ follows the above payout specification, with $R_t = 0$ if s_t is not terminal. It is also possible, albeit tedious, to calculate the transition probabilities and to show that the resulting process is a Markov decision process.

Non-stationary Markov Decision Process. Most strategies in blackjack involves keeping track only of your own card sum (with some possible extensions) and not every card you are holding. Hence, it seems natural to represent the state space in the following way

$$S = \{(\Sigma_p, a_p, \Sigma_d)\}$$

where Σ_p, Σ_d is the card sum of the player and the dealer, and a_p is the number of aces the player is holding. Although, this representation yields a MDP the process is not stationary. To see this, compare $\mathbb{P}(S_1 = (20, 1, x, y)|S_0 = (20, 1, x, y), \text{hit})$ with $\mathbb{P}(S_9 = (20, 1, x, y)|S_8 = (20, 1, x, y), \text{hit})$. The first probability is larger than 0 since $S_0 = (20, 1, x, y)$ happens only if player gets an ace and a nine on its first hand (9+11 = 20). If the player hits and gets a ten the system moves to $S_1 = (20, 1, x, y)$ (1+9+10, recall ace can count as 1 or 11). However, $S_8 = (20, 1, x, y)$ can only happen if the player have gotten 8 twos 1 three and 1 ace ($2 \cdot 8 + 3 + 1 = 20$), and whichever card the player get next will change the state of the system — i.e. $\mathbb{P}(S_9 = (20, 1, x, y)|S_8 = (20, 1, x, y), \text{hit}) = 0$. We see that the transition probabilities depend on t. This means that we cannot be sure that the optimal policy is given by a stationary Markov policy.

Results

Conclusion