Reinforcement Learning - Monte Carlo Methods

And their application to Blackjack

M. Heinzer[1] E. Profumo[1]

[1]Deparment of Mathematics
ETH Zürich

Seminar in Statistics: Learning Blackjack, April 2016

Goals

- As blackjack players we want to **find an optimal strategy** that is a way of selecting the best action in each state of the game, the one which will maximize the probability of us winning.
- We will try to learn the optimal strategy presented in an earlier talk, but this time not by calculating it from the **probability distribution** of our environment but by **learning from experience** using Monte Carlo methods.

Mathematical setting

We pick a theoretical framework from decision theory. The evolution of a game is described by a finite **Markov Decision Process (MDP)**.

Mathematical setting

Notation :

- $t \in \{1, 2, \dots T_i\}$ describes the different **steps** of the episode i (we will drop i for clarity).
- $(S_t)_{t \in \{1,2,\dots T\}}$ the process of different **states of the game**.
- $(S_t, a_t)_{t \in \{1,2,\dots T\}}$ the **state-action** pairs. *The actions which can be taken depend on the current state*
- $(R_t)_{t \in \{1,2,\dots T\}}$ the process of **rewards** following a triple (state,action,resulting state).

Mathematical setting

At each time step t the decision process follows

1. We are in some state $S_t = s_t$
2. The player takes an action a_t from S_t.
3. the state-action pair $(S_t = s_t, a_t)$ leads to a random following state $S_{t+1}|s_t, a_t$.
4. the triple (s_t, a_t, s_{t+1}) leads to a reward R_{t+1}.

The agent and the environment

We have an **agent** which chooses actions in an **environment**.

1. An agent is equipped with a decision process which chooses an action depending on the current state, we call it a **policy function** π. It can be deterministic, like a greedy policy or stochastic. $\pi(a|s)$ is the probability of taking action a being in state s.
2. Actions are motivated by rewards. Our interest is to maximize the **expected aggregated reward** $\mathbb{E}\left(\sum_{t=0}^{\infty} R_t\right)$

Best policy, state and action-state value functions.

We need an indicator for the goodness of a policy, to be able to compare different strategies. Then we need to **construct optimal policies**. That is why we recall two crucial notions.

Best policy, state and action-state value functions.

State value function :

$$v_\pi(s) = \mathbb{E}_\pi \left(\sum_{i=0}^{\infty} R_{t+i+1} \middle| S_t = s \right)$$

Knowing that I am in state s which reward can I expect following policy π.

- The state value function **provides a measure for goodness of a policy**. It gives a **partial ordering of policies**, the higher the value function for each state, the better the policy.
- Best policies share the same optimal state value function $v_\star(s) = \max_\pi v_\pi(s) \quad \forall s \in S$, we denote them by $\pi_\star$.

Best policy, state and action-state value functions.

In order to construct policies with better state value functions, we need a tool that takes into account **actions**. We call it **Action-State value function :**

$$q_\pi(s,a) = \mathbb{E}_\pi\left(\sum_{i=0}^{\infty} R_{t+i+1} \middle| S_t = s, A_t = a\right)$$

Knowing that I am in state s, which reward can I expect by taking action a under policy π.

General idea

So far we have seen **Dynamic Programming**, which worked as follows :

- Compute the value of each state from the complete knowledge of the environment. We had to solve equations like this :

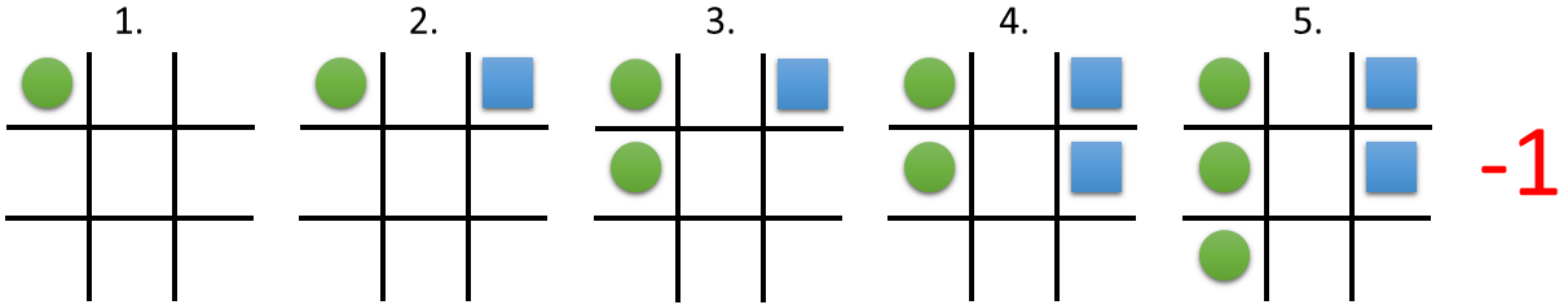
$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)[r + \gamma v_\pi(s')]$$

Now we will see **Monte Carlo** methods, which learn the value of each state from sample returns.

General idea

What is the structure of a Monte Carlo method ?

- Firstly we generate experience. Go through a sequence of actions and states until we arrive at a terminal state. We call this sequence an **episode**, in this talk only finite episodes will be considered. Example :

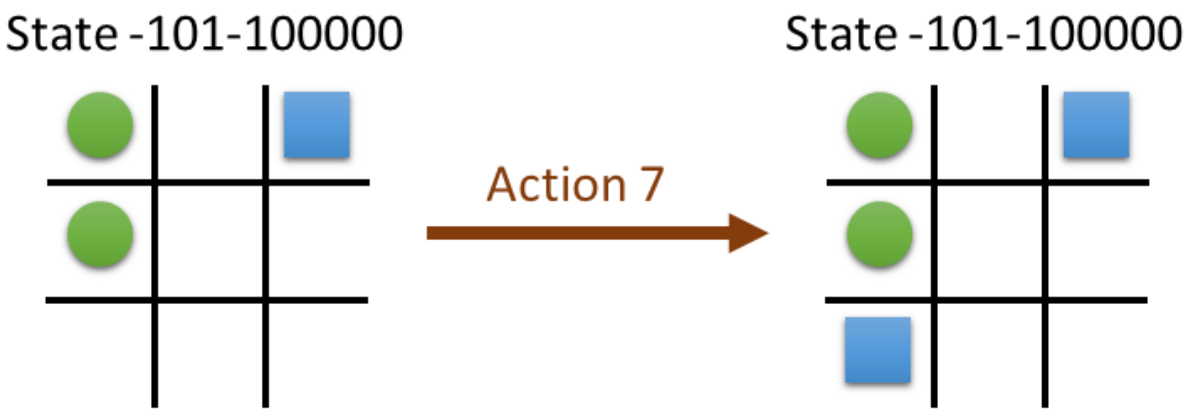


The reward is only given at the end of an episode. The value of each state we have gone through will be adapted accordingly.

General idea

What is the structure of a Monte Carlo method ?

- Secondly we update the policy according to our experience for each state. It will only be updated after the completion of one or more episodes. Example :



State - Action	2	5	6	7	8	9
-101-100000	-1	-1	-1	0.4	-1	-1

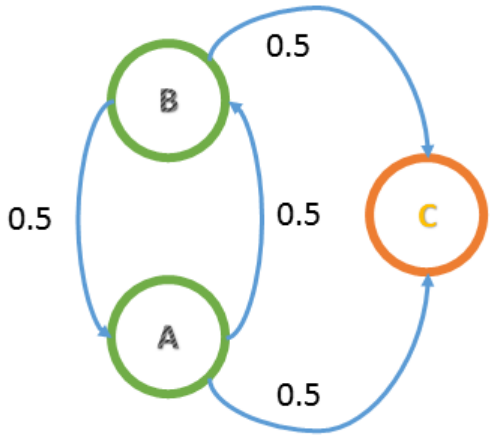
Our policy would be : When in state -101-100000 select action 7.

Monte Carlo Prediction : Estimating values

The first step is to estimate the value of each state given a policy π.
How do we calculate $v_\pi(s)$? Two possibilities :

- **First-Visit** MC
- **Every-Visit** MC

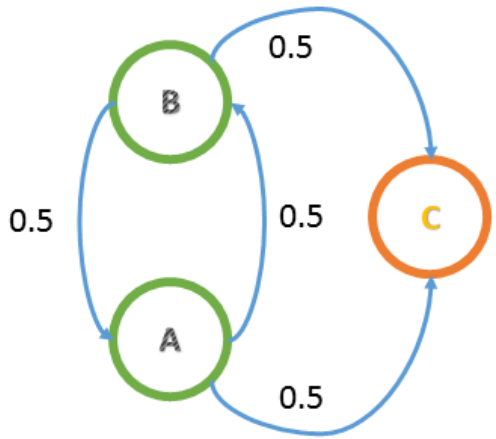
Example :



Form here on we will only consider First-Visit MC

Monte Carlo Prediction : Estimating values

First-visit MC averages the returns following a visit to a state s in all episodes. Suppose we have n episodes and let $N(s)$ be the enumeration of episodes which visited s.
An example of $N(s)$ with the previous process could be :



Episode 1: {A,B,A,B,C}
Episode 2: {A,C}
Episode 3: {A,B,A,C}

$N(A) = \{1, 2, 3\}$, $N(B) = \{1, 3\}$

Monte Carlo Prediction : Estimating values

First-visit MC averages the returns following a visit to a state s in all episodes. Suppose we have n episodes and let $N(s)$ be the enumeration of episodes during which s was visited.
Then we define $\hat{v}_\pi(s)$ as follows :

$$\hat{v}_\pi(s) = \frac{1}{|N(s)|} \sum_{i=1}^{n} R_i I_{i \in N(s)}$$

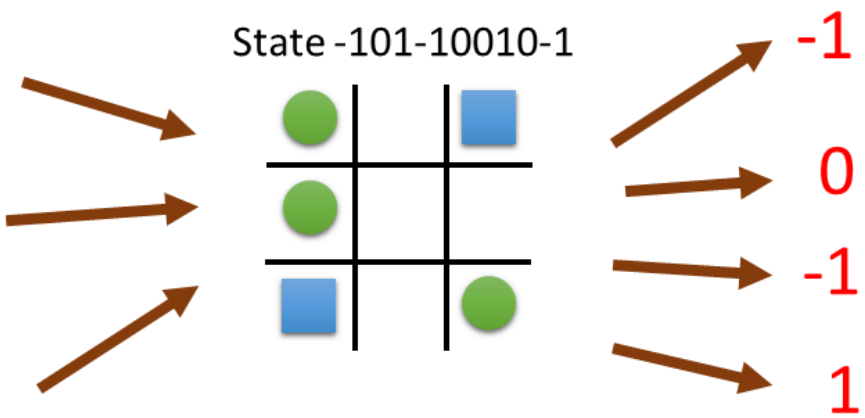
Each return is an i.i.d. estimate of the true value of $v_\pi(s)$.The law of large number gives us convergence to the expectation.

Monte Carlo Prediction : Estimating values

Suppose we give the following rewards for Tic Tac Toe :

- 1 if we win
- 0 for a draw
- -1 if we lose

In our toy example Tic Tac Toe we could have the following experience :



After those experiences we have $\hat{v}_\pi(s) = -\frac{1}{4}$

Monte Carlo Prediction : Estimating values

Other remarks :

- All of this is done for a given policy
- Estimates for each state are independent, in the sense that we do not use values from other states in our computation.
- Computation of the value of one state is independent of the number of states
- Assumption of infinite number of episodes !

Monte Carlo Prediction : Estimating values

The whole approach can also be used to approximate action values. We just consider state action pairs and estimate their values. This is needed to improve the policy. Formally :

$$\hat{q}_\pi(s,a) = \frac{1}{|N(s,a)|} \sum_{i=1}^{n} R_i I_{i \in N(s,a)}$$

Monte Carlo Prediction : Estimating values

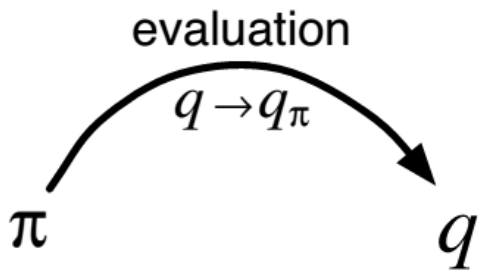
We need another assumption in order to estimate **all** pairs of state-action values, as many combinations may never be visited. Problem of maintaining exploration, as with n-armed bandits. Two possibilities :

- Every pair has non-zero probability of being selected as start. We call this **Exploring Starts**.
- Use **stochastic policies** which have a non-zero probability of selecting all available actions in each state.

Q : Do we have exploring starts in Tic Tac Toe ?
Q : Do we have exploring starts in Blackjack ?

Monte Carlo Control

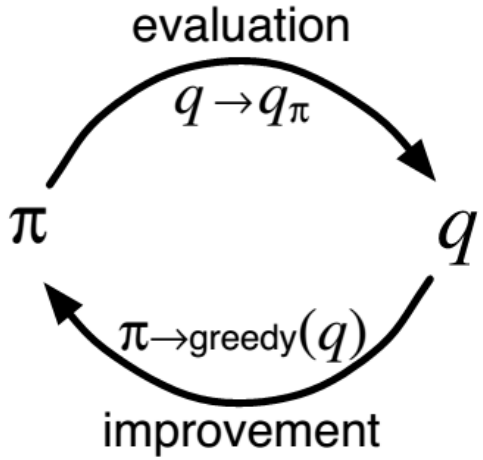
So far we have only considered how to approximate value functions.



But what about the other direction ?

Monte Carlo Control

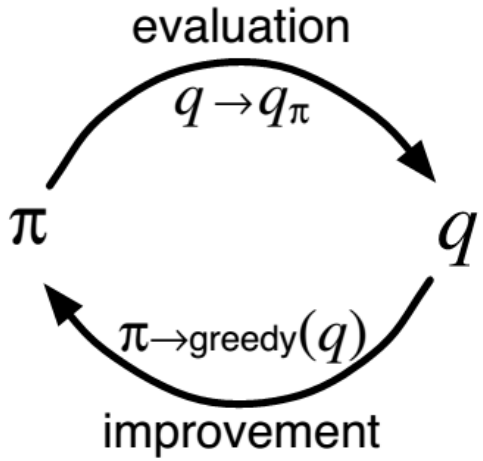
So far we have only considered how to approximate value functions.



Now we want to approximate optimal policies as well. The idea is the same as in **Generalized Policy Iteration** (GPI) in the previous presentation about dynamic programming.

Monte Carlo Control

So far we have only considered how to approximate value functions.

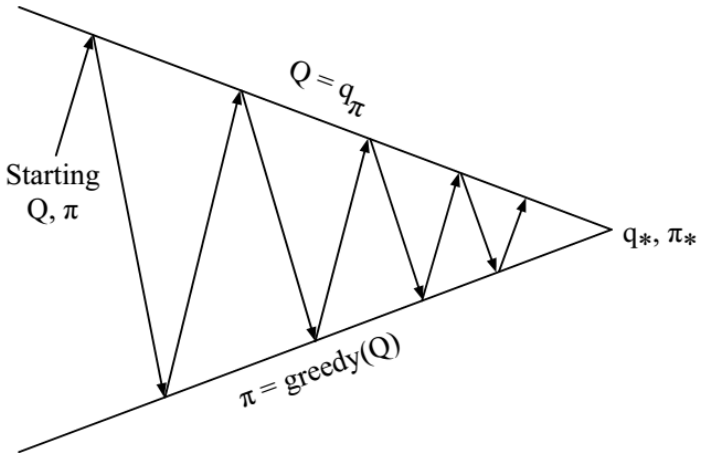


We start with an arbitrary policy π_0. At each step we evaluate the state-action function q_{π_i}, and select as new policy π_{i+1} the greedy policy corresponding to q_{π_i} :

$$\pi_{i+1}(s) = \arg\max_a q_i(s, a)$$

Monte Carlo Control

Q : Do we actually converge to the optimal policy ?



A : Yes ! (Be aware of the conditions)

Monte Carlo Control

Convergence is guaranteed by the policy improvement theorem. We only need to verify that the new policy is uniformly better. We have for all π_k, π_{k+1} and $s \in S$:

$$\begin{aligned} q_{\pi_i}(s, \pi_{i+1}(s)) &= q_{\pi_i}(s, \arg\max_a q_{\pi_i}(s, a)) \\ &= \max_a q_{\pi_i}(s, a) \\ &\geq q_{\pi_i}(s, \pi_i(s)) \\ &= v_{\pi_i}(s) \end{aligned}$$

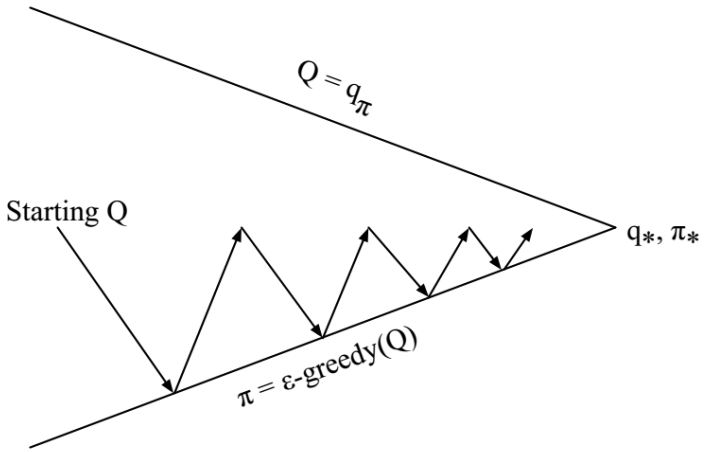
Therefore by the policy improvement theorem : $q_{\pi_{i+1}} \geq q_{\pi_i}$
Q : Do we have something useful ?

Monte Carlo Control

Our assumptions of exploring starts and infinitely many episodes have to be removed. The latter is easy to remove. We can :

- Collect enough episodes until the margin of error is small enough, can still be computationally intensive
- Update the policy after each episode,

We move our value function towards the real one :



Q : Does it still converge ? Open problem !

Monte Carlo Control

How do we take away the Exploring Starts assumption ? Two main ideas :

- **On-policy methods** :
  Select the policy in such a way that
  $\pi(a|s) > 0 \quad \forall s \in S, a \in A(s)$.
  We call this a soft policy. Example : ϵ-greedy
- **Off-policy methods** :
  Maintain two policies, one for exploring and another for optimization.

Monte Carlo Control - on policy

We will check if on policy learning still fulfills the condition of the policy improvement theorem. First let us consider an ϵ-greedy policy as follows :

$$\pi(a|s) = \begin{cases} \frac{\epsilon}{|A(s)|} & \text{for the non-greedy action} \\ 1 - \epsilon + \frac{\epsilon}{|A(s)|} & \text{for the greedy action} \end{cases}$$

Monte Carlo Control - on policy

We have for any $s \in S$:

$$
\begin{aligned}
q_\pi(s, \pi'(s)) &= \sum_a \pi'(a|s) q_\pi(s, a) \\
&= \frac{\epsilon}{|A(s)|} \sum_a q_\pi(s, a) + (1 - \epsilon) \max_a q_\pi(s, a) \\
&\geq \frac{\epsilon}{|A(s)|} \sum_a q_\pi(s, a) + (1 - \epsilon) \sum_a \frac{\pi(a|s) - \frac{\epsilon}{|A(s)|}}{1 - \epsilon} q_\pi(s, a) \\
&= \frac{\epsilon}{|A(s)|} \left(\sum_a q_\pi(s, a) - \sum_a q_\pi(s, a) \right) + \sum_a \pi(a|s) q_\pi(s, a) \\
&= v_\pi(v)
\end{aligned}
$$

Thus $\pi' \geq \pi$ We have thus shown that policy iteration works for ϵ-soft policies.

Monte Carlo Control - off policy

Suppose we want to estimate v_π or q_π of a policy π but we can't test it directly. How do we gather experience ?
An example : Suppose you want to test your new Black Jack strategy, but as poor student you don't have enough money to play at the casino. Off policy methods allow you to assess your strategy observing other players.
Formally we use another policy μ to generate data, and estimate v_π or q_π. We call

- π the **target policy**
- μ the **behavior policy**

Moreover we do not need the assumption of exploring starts.

However we need μ to satisfy a condition :

$$\pi(a, s) > 0 \Rightarrow \mu(a, s) > 0$$

Every action which is taken under policy π must have a non-zero probability to be taken as well under policy μ. We call this the assumption of **coverage**.
Typically the target policy π would be a greedy policy with respect to the current action-value function.

Monte Carlo Control - off policy

The tool we use for estimation is called **importance sampling**. It is a general technique for estimating expected values under one distribution given samples from another.
Given a state S_t, the probability of a subsequent state-action trajectory $A_t, S_{t+1}, A_{t+1}, \ldots, S_T$ occurring under policy π is

$$P_\pi(\{\{S_t, A_t\}, \ldots, \{S_T\}\}) = \prod_{k=t}^{T-1} \pi(A_k|S_k) p(S_{k+1}|S_k, A_k)$$

where p is the state-transition probability.

Monte Carlo Control - off policy

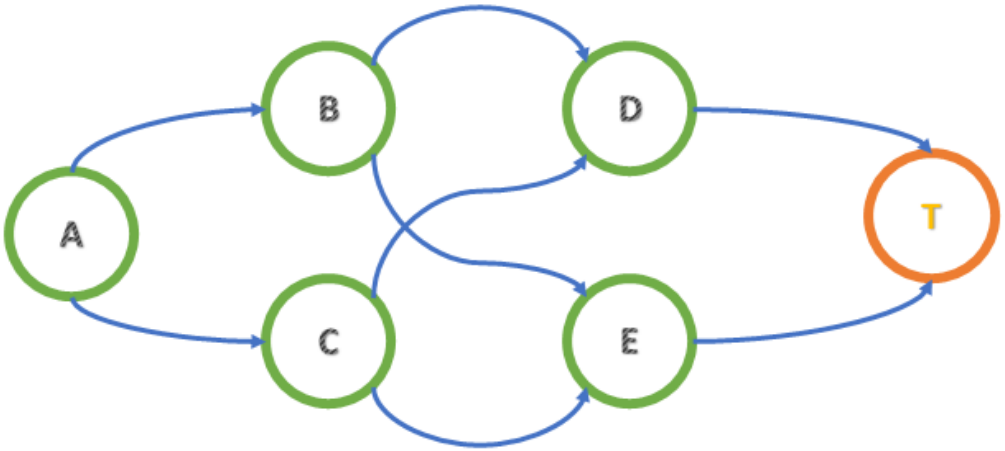
The relative probability of the trajectory under the target and behavior policies, or the importance sampling, is :

$$\rho_t^T := \frac{\prod_{k=t}^{T-1} \pi(A_k|S_k)p(S_{k+1}|S_k, A_k)}{\prod_{k=t}^{T-1} \mu(A_k|S_k)p(S_{k+1}|S_k, A_k)} = \prod_{k=t}^{T-1} \frac{\pi(A_k|S_k)}{\mu(A_k|S_k)}$$

The trajectory probabilities depend on the MDP, which are generally unknown, but cancel each other out.

Monte Carlo Control - off policy

Suppose we have gathered experience in the form of n episodes.
Let $N(s)$ be the enumeration of episodes which visited state s.
Let $T(s,k)$ be the first time when state s is visited in episode k.
The time of the terminal state of episode k is denoted as $T(k)$.
An example could look like this :



Episode 1: {A,B,E,T}, {5}
Episode 2: {A,C,E,T}, {8}

We would have : $N(E) = \{1,2\}$, $T(B,1) = 2$, $T(1) = 4$ and $G_2 = 8$.

Monte Carlo Control - off policy

One way to estimate $v_\pi(s)$ is to scale the returns by the number of times we visited state s :

$$\hat{v}_\pi(s) = \frac{\sum_{i \in N(s)} \rho_{T(s,i)}^{T(i)} G_i}{|N(s)|}$$

This is what we call **ordinary importance sampling**.
Q : What problem does this estimator have ?
The variance is unbounded in general !
Example : One visit of s with ratio 100 and return 1, $\hat{v}_\pi(s) = 100$.

Monte Carlo Control - off policy

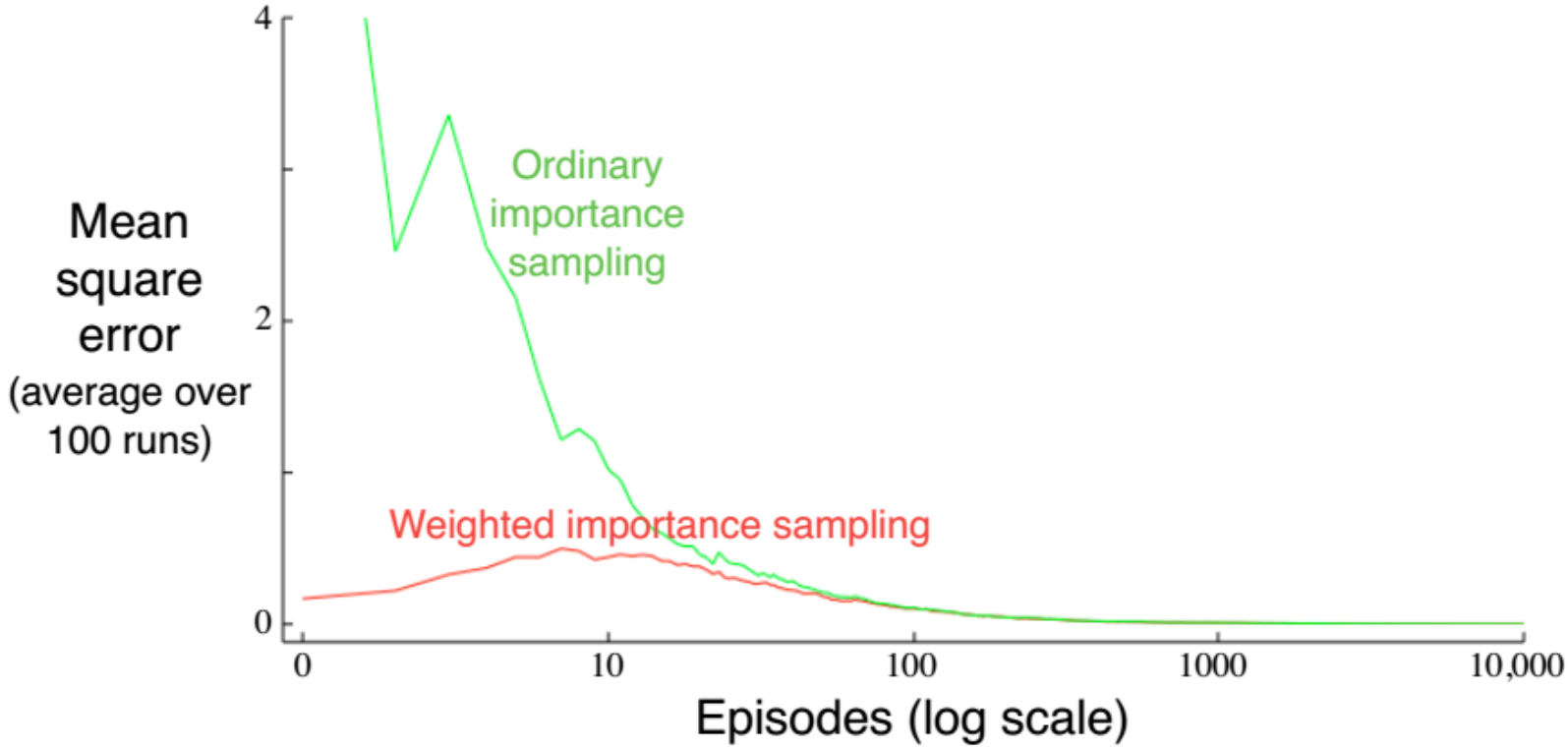
An alternative is **weighted importance sampling**, which is using an weighted average :

$$\hat{v}_\pi(s) = \frac{\sum_{i \in N(s)} \rho_{N(s,i)}^{T(i)} G_i}{\sum_{i \in N(s)} \rho_{N(s,i)}^{T(i)}}$$

Q : What is the problem with this estimator ?
The estimator is biased in the statistical sense, its expectation is $v_\mu(s)$ instead of $v_\pi(s)$.

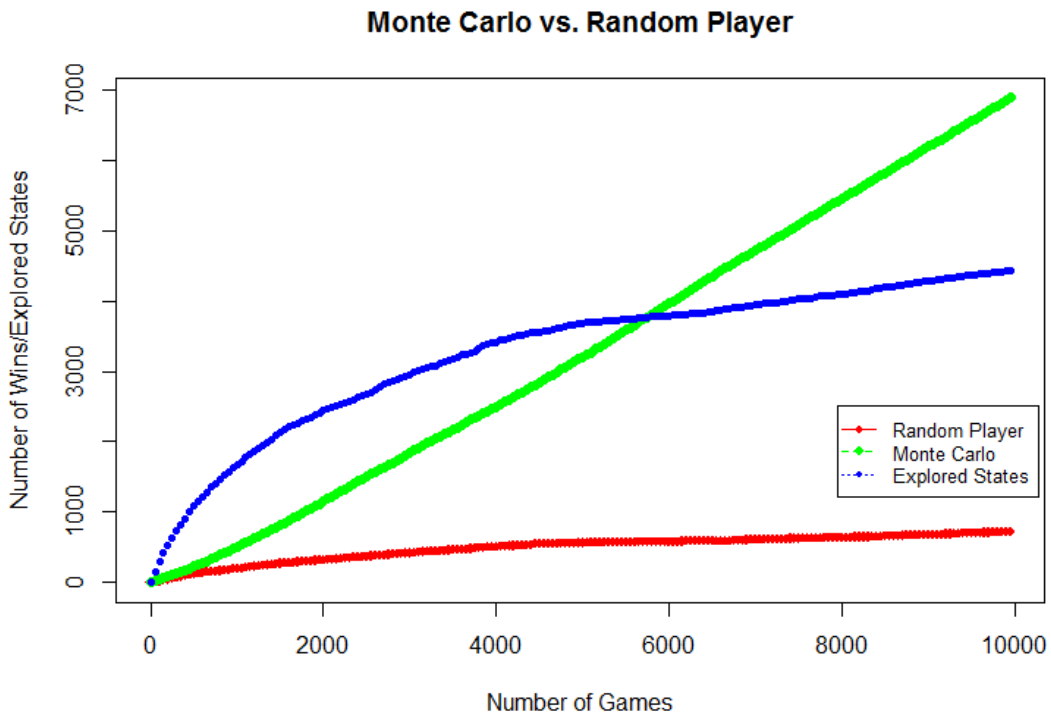
Monte Carlo Control - off policy

In practice the weighted estimator has dramatically lower variance and is therefore strongly preferred. Example of a blackjack state :



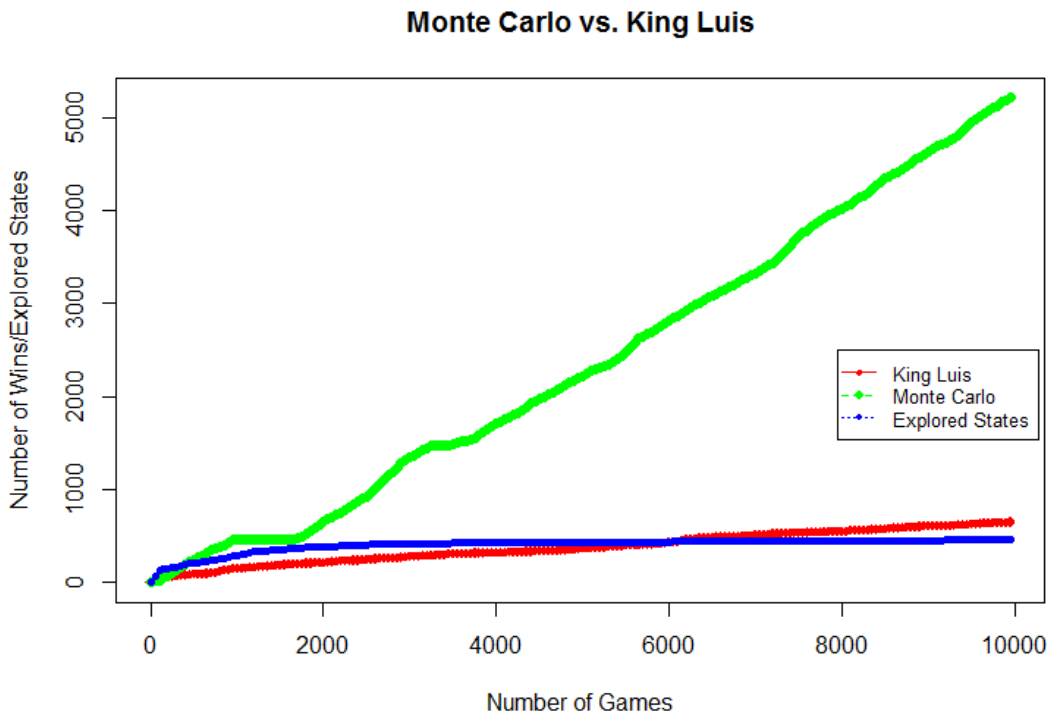
Monte Carlo - Learning Tic Tac Toe

We let our Monte Carlo learner play against a random strategy :



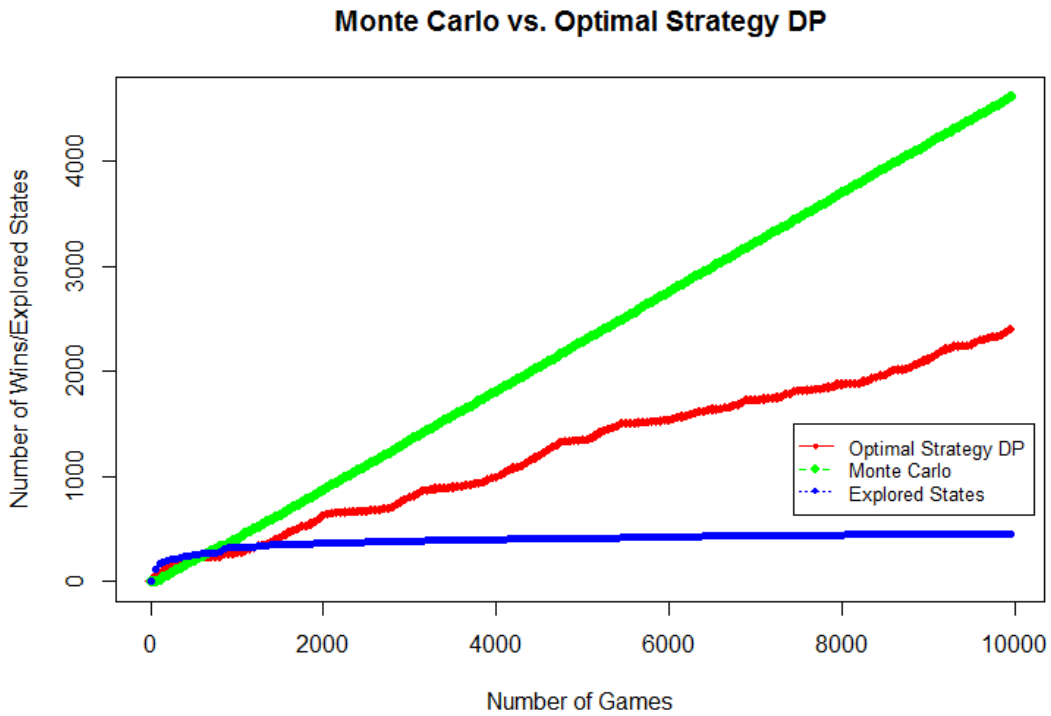
Monte Carlo - Learning Tic Tac Toe

We let our Monte Carlo learner play against the King Luis :



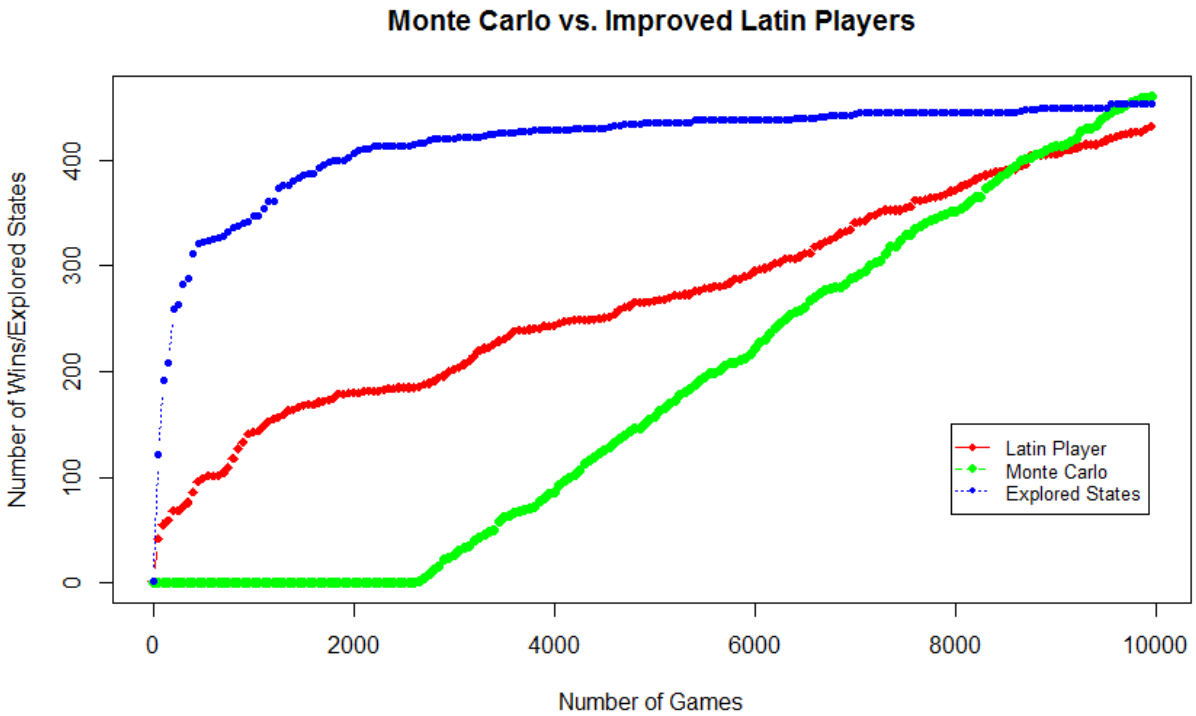
Monte Carlo - Learning Tic Tac Toe

We let our Monte Carlo learner play against the optimal strategy from DP :



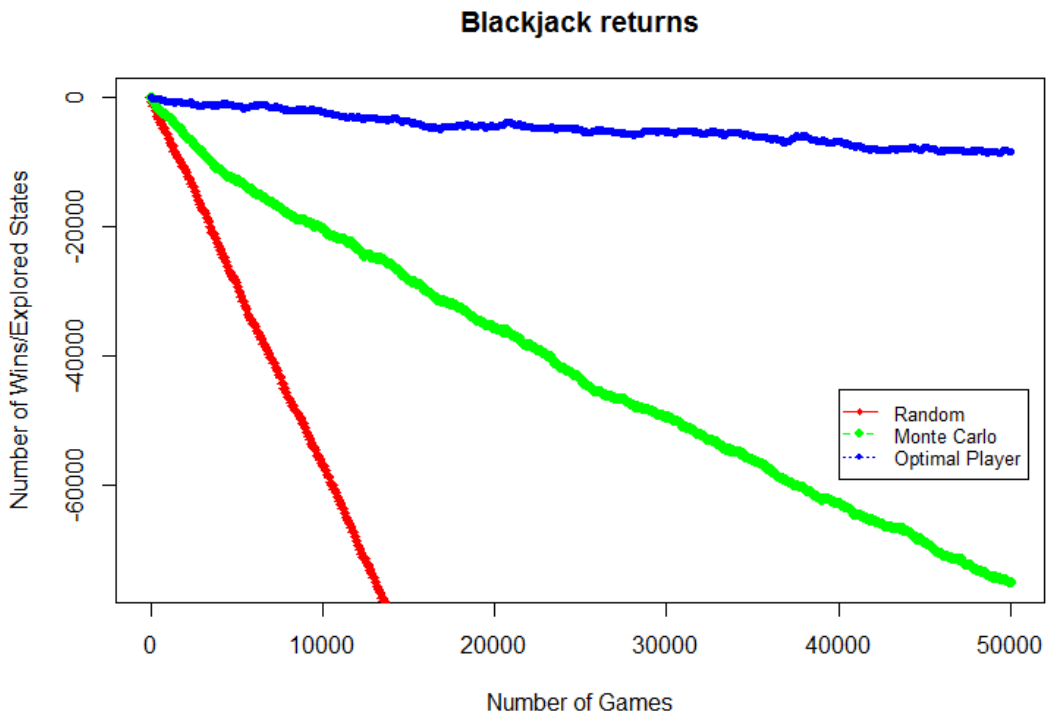
Monte Carlo - Learning Tic Tac Toe

We used it to play Tic Tac Toe against the best strategy we could come up with (Improved Latin Players) :



Monte Carlo - Learning Black Jack

We let it play Black Jack with the Random and the Optimal Player :



Due to the ϵ-soft policy it still takes a lot of wrong decisions.

Monte Carlo - Learning Black Jack

We update our strategy after each game, using a running average :

```
def update_strategy(self, gameid):
    #update the value function of the state actions
    for stateAction in self.episode:
        if stateAction in self.strategy.table:
            val = self.strategy.table[stateAction]
            avg = val[0]
            n = val[1]+1
            avg -= avg/n
            avg += self.gainInGame/n
            val[0] = avg
            val[1] = n
        else:
            val = [self.gainInGame+0.0,1]
            self.strategy.table[stateAction] = val
    #clear list
    del self.episode[:]
```

Monte Carlo - Learning Black Jack

The three crucial methods which generate and select actions :

```
def getPossibleActions(self, h):
    actions = []
    actions.append(self.get_state_mc(h)+"1")
    actions.append(self.get_state_mc(h)+"2")
    if h is 0:
        actions.append(self.get_state_mc(h)+"3")
    if self.hands[h].can_split():
        actions.append(self.get_state_mc(h)+"4")
    return actions

def getGreedyAction(self, asValues):
    maxIndex = np.argmax(asValues)
    if asValues.count(asValues[maxIndex]) == 1:
        return maxIndex
    places = []
    maxVal = asValues[maxIndex]
    for i in range(len(asValues)):
        if asValues[i] == maxVal:
            places.append(i)
    return places[int(random.uniform(0,1)*(len(places)-1))]

def action(self, h):
    #Test if we are still in the same state,
    #if yes its not necessary to recalculate the current the action
    if self.actionCalculatedFor == self.get_state_mc_hash(h):
        return self.currentAction
```

Monte Carlo - Learning Black Jack

A excerpt of the strategy table, without an ace and splitting possibility :

Epsilon 0.1 and 100000 games																	
Card Valu	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
Dealer: 2	s	s	h	h	h	h	h	h	s	s	s	s	s	s	s	s	s
Dealer: 3	h	s	h	s	s	h	h	s	h	s	s	s	s	s	s	s	s
Dealer: 4	s	h	h	h	h	h	h	s	h	s	s	s	s	s	s	s	s
Dealer: 5	h	s	h	h	h	h	h	h	s	s	h	s	s	s	s	s	s
Dealer: 6	h	s	h	h	h	h	h	s	h	s	s	s	s	s	s	s	s
Dealer: 7	s	h	h	h	h	h	h	h	h	s	h	h	s	s	s	s	s
Dealer: 8	s	s	h	h	h	h	h	h	h	s	s	h	s	s	s	s	s
Dealer: 9	s	s	s	h	h	s	h	h	h	h	s	s	s	s	s	s	s
Dealer: 10	h	s	s	h	h	h	h	h	h	h	s	s	s	s	s	s	s
Epsilon 0.5 and 200000 games																	
Card Valu	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
Dealer: 2	h	s	h	h	h	h	h	s	s	s	s	s	s	s	s	s	s
Dealer: 3	s	h	h	s	h	h	h	s	s	s	s	s	s	s	s	s	s
Dealer: 4	h	s	h	h	h	h	h	s	s	s	s	s	s	s	s	s	s
Dealer: 5	s	s	s	h	h	h	h	s	s	s	s	s	s	s	s	s	s
Dealer: 6	s	h	h	h	h	h	h	s	s	s	s	s	s	s	s	s	s
Dealer: 7	h	h	h	h	h	h	h	h	h	s	h	s	s	s	s	s	s
Dealer: 8	s	h	h	h	h	h	h	h	h	h	h	s	s	s	s	s	s
Dealer: 9	s	h	h	h	h	h	h	h	h	h	s	h	s	s	s	s	s
Dealer: 10	h	s	h	h	h	h	h	h	h	s	s	s	s	s	s	s	s