Reinforcement Learning Notes from Guillaume Charpiat's Course

Hugo Richard

January 2018

1 Chapter 1: Bandits

1.1 Reinforcement learning framework

The Agent performs an action on the Environment and gets a reward that depends on the Environment. In bandits, the environment is static (no environment basically)

Ex: Medicine trial Want to make statistics on medicine A, B, C to know which one is the best. You also want to minimize the number of people dying in the process.

1.2 Strategies

- Greedy: you would try a few times each of them and then you say: "I am done exploring" then you exploit (you take always the best arm in the previous step). Issues: How many times is "a few time" ?
- To be better you need to keep exploring all the time: ϵ-Greedy: At each time-step you flip a coin a with probability epsilon you pick randomly uniformly any arm, otherwise (probability $1-\epsilon$) you take the best arm so far. At initialization you either chose a high or low estimate for each arm (high if you want to force exploration at the beginning)
- Softmax: $Q(a)$ is the average reward for the action a

$$p(a) = \frac{exp(\frac{Q(a)}{\tau})}{\sum_{a'} exp(\frac{Q(a')}{\tau})}$$

  if τ is close to 0 it tends to be close to the greedy strategy. if τ is close to infinity: you pick every arm with equi-probability.

1.3 Model

k arms to pull. Each time you pull one arm there is a distribution of possible rewards D_k with its value in $[0, 1]$. At time t we chose an arm $I_t \in \{1, .., K\}$ and get the reward x_t which follows D_{I_t} (x_t are iid)

We define: $\mu_k = E[x_k]$ and $\mu_* = max_k(\mu_k)$

The regret for n trials is defined as:

$$\begin{aligned} R_n &= n\mu_* - \sum_{t=1}^{n} x_t \\ &= \text{best-reward-on-average} - \text{current-reward} \end{aligned}$$

We have

$$\begin{aligned} \mathbb{E}[R_n] &= n\mu_* - \sum_t \mathbb{E}_{I_t}[\mu_{I_t}] \\ &= \mathbb{E}[\sum_k T_k(n)(\mu_* - \mu_k)] \end{aligned}$$

with $T_k(n)$ the number of time you select the arm k and $\delta_k = \mu_* - \mu_k > 0$.

1.4 UCB: Upper confidence bound

$$B_{t,s}(k) = \widehat{\mu_{k,s}} + \sqrt{\frac{2logt}{s}}$$

with $s = T_k(t-1)$ the number of time you picked arm k until now and $\widehat{\mu_{k,s}}$ the average reward so far for arm k

At time t, Select:

$$I_t = \text{argmax}_k B_{(t,s)}(k)$$

This is a way of selecting the arm with the highest optimistic average (average + error).

1.5 UCB Bound

The Chernoff-Hoeffding inequality reads:

$$p(\frac{\sum_t x_t}{s} - \mu \geq \epsilon) \leq exp(-2s\epsilon^2)$$

So we get

$$P(\widehat{\mu_{k,s}} + \sqrt{\frac{2logt}{s}}) <= \mu_k) <= exp(-4log(t)) = 1/t^4$$

Lemma If k is not optimal:

$$E[T_k(n)] \leq 8\frac{log(n)}{\delta_k^2} + \frac{\pi^2}{3}$$

We then have

$$\begin{aligned} E[R_n] &= \sum_k \delta_k E[T_k] \\ &\leq \sum_{k \neq k^*} 8\frac{logn}{\delta_k} + K\frac{\pi^2}{3} \end{aligned}$$

So $E[R_n]$ is bounded by a linear function of $logn$.

But δ_k could be very small. This is not a problem because Cauchy-Schwartz yields:

$$\begin{aligned} E[R_n] &= \sum_k \delta_k E[T_k] \\ &\leq \sum_k \sqrt{\delta_k^2 E[T_k]}\sqrt{E[T_k]} \\ &\leq \sqrt{\sum_k \delta_k^2 E[T_k]}\sqrt{\sum_k E[T_k]} \\ &\leq \sqrt{8Kn(logn + \frac{\pi^2}{3})} \end{aligned}$$

Proof of the lemma Suppose at time t

$$\mu_k - \sqrt{\frac{2logt}{s}} \leq \widehat{\mu_{k,s}} \leq \mu_k + \sqrt{\frac{2logt}{s}}$$

Consider k not optimal, k_* the best. It time t, arm k is picked, then

$$\begin{aligned} & B_{t,T_k(t-1)}(k) \geq B_{t,T_{k_*}(t-1)(k_*)} \\ & \widehat{\mu_{k,s}} + \sqrt{\frac{2logt}{s}} \geq \widehat{\mu_{k_*,s_*}} + \sqrt{\frac{2logt}{s_*}} \\ & \mu_k + \sqrt{\frac{2logt}{s}} \geq \mu_* \\ & s \leq 8\frac{logt}{\delta_k^2} \end{aligned}$$

$\forall u \in \mathbb{N}$,

$$\begin{aligned} T_k(n) &\leq u + \sum_{t=u+1}^{n} \mathbb{1}_{I_t = k \wedge T_k(t) \geq u} \\ &\leq u + \sum_{t=u+1}^{n} \mathbb{1}_{\exists s, s' u < s \leq t \wedge 1 \leq s_* \leq t \wedge B_{t,s}(k) \geq B_{t,s_*}(k_*)} \end{aligned}$$

Finally, consider $u = \frac{8logn}{\delta_k^2}$:

$$\begin{aligned} E[T_k(n)] &\leq \frac{8logn}{\delta_k^2} + \sum_{t=u+1}^{n} (\sum_{s=u+1}^{t} t^{-4} + \sum_{s=1}^{t} t^{-4}) \\ &\leq \frac{8logn}{\delta_k^2} + \frac{\pi^2}{3} \end{aligned}$$

1.6 Inf Bounds

[Lai et Robbins 1985]

$$\begin{aligned} &\text{limsup}_n \frac{E[T_k(n)]}{logn} >= \frac{1}{KL(D_k||D*)} \\ &E[T_k(n)] = \Omega(logn) \\ &E[R_n] = \Omega(logn) \end{aligned}$$

[Cesa-Bianchi and Lugos 2006]

$$inf_{algo} sup_{task} E[R_n] = \Omega(\sqrt(nK))$$

1.7 KL-UCB

$$B_{t,s}(k) = sup_D\{E[D], ||\widehat{\mu_{k,s}} - E[D]||^2 \leq \frac{f(t)}{T_k(t)}\}$$

D is a model. The term $||\widehat{\mu_{k,s}} - E[D]||^2$ is better replaced with $KL(\widehat{\mu_{k,s}}||D)$.

1.8 Variation on Softmax

$Q_t(k)$ is the total reward of arm k until now.

$$p_t(k) = exp(Q_t(k))$$

and finally:

$$p(h) = p_t(h)(1-\gamma) + \gamma 1/K$$

If $\gamma = \frac{KlogK}{(e-1)n}$

$$E[R_n] \leq O(\sqrt(nKlogK))$$

1.9 Combining expert advice

Time series prediction
Each expert gives a distribution of probabilities $D_{k,t}$.
A linear combination of all experts is often better than a single one.
Regret is redefined with respect to the new maximum expected gain.

2 Chapter 2 : Learning dynamics

2.1 Definitions

An **Agent** interacts with an **Environment**. The **State** is given both by the internal configuration of the agent and the environment. Generally speaking a state is a compact description of the world. You can say it is an a-priori where you put all information available for the decision to be taken. You can also put history in that. The difference between the agent and the environment is usually defined by what you can control and what you cannot.

At each step, the agent takes an **action**, go to a new **state** and gain a **reward**.

When you are in a particular state you choose an action based on a **policy**. We try to find the best policy to get maximum reward.

Example 1 Let us consider a Robot with an arm. The position of the arm represents the state of the robot.

An agent may not know the full state of the world. It may have only a partial view.

Example 2 Let us consider a Robot moving. The agent is the controller, the part making decision. It is not the hardware.

Example 3 Let us consider a chess game. If you specify sub-goals, for example gaining points when taking adverse pieces, you might end up taking the queen and losing the game: subgoals are dangerous.

The goal is to maximize rewards on the long run.

Let us consider a finite horizon T (finite number of turn in a game). At time t you want to maximize:

$$\mathop{\mathbb{E}}_{e,\pi}\left[\sum_{i=t+1}^{T} r_i\right]$$

where π is the policy, a_t, s_t, r_t are respectively action, state and reward at time t and e is the environment.

In practice there might not be a specified finite number of turn but there usually exists terminal state (states that end the game).

You might want to put less weight on rewards you will gain in a very far future.

$$\mathop{\mathbb{E}}_{e,\pi}[\sum_{k=0}^{\infty} \gamma^k r_{t+1+k}]$$

where γ, the discount rate is such that $0 \leq \gamma \leq 1$.

We define the return R by:

$$R = \sum_{k=0}^{\infty} \gamma^k r_{t+1+k}$$

2.1.1 Markov Decision Process (MDP)

To chose an action you might want to evaluate the quantity:

$$p = p(s_{t+1}, r_{t+1}|s_t, a_t, r_t, s_{t-1}, a_{t-1}, r_{t-1}, ...)$$

We make the hypothesis of a Markovian environment (a way to do that is to include history but remember we want short states...) so that we have $p = p(s_{t+1}, r_{t+1}|s_t, a_t)$

In a Markov Decision Process, everything is described by only two quantities:

- $p(s_{t+1}|s_t, a_t)$: this describes the dynamics (how we are moving in the space of state)
- $\mathbb{E}_e[r_{t+1}|s_t, a_t, s_{t+1}]$: the average reward (full distribution not required)

This can be described as a graph (like a Markov chain).

2.1.2 Value functions: V and Q

Let us suppose that an environment follows a Markov Decision Process and that we know all the state and reward graph.

Policy π is a function such that $\pi(state, action)$ is the probability of taking that action when in that state. Of course $\sum_a \pi(state, action) = 1$.

Let us assume you are in the state s_t you chose action a_t using policy distribution and a new state and reward is sampled from the environment.

A State value function V is a function such that $V(state)$ is the expected return when you are in that state.

$$V^{\Pi}(s) = \mathop{\mathbb{E}}_{\pi,e}[R_t|s_t = s] = \mathop{\mathbb{E}}_{e,\pi}[\sum_{k=0}^{\infty} \gamma^k r_{t+1+k}|s_t = s]$$

with s_T the terminal state. We have $V^{\pi}(s_T) = 0$.

An State-action value function Q is a function such that $Q(state, action)$ is the expected return when you are in that state and you take that action.

$$Q^{\pi}(s, a) = \mathop{\mathbb{E}}_{\pi,e}[R_t|s_t, a_t]$$

Consider any policy π and any state s. At each time step we choose an action a_t which depends on the policy π and the current state s_t. Depending on the action a_t the current state s_t and the environment e, we get a new state s_{t+1} and a reward r_{t+1}. We denote $\mathbb{E}_{e,\pi}$ the expectation taken on all possible

actions, states and rewards.

$$\begin{aligned} V^{\pi}(s) &= \mathop{\mathbb{E}}_{\pi,e}[R_t|s_t = s] \\ &= \mathop{\mathbb{E}}_{e,\pi}[\sum_{k=0}^{\infty} \gamma^k r_{t+1+k}|s_t = s] \\ &== \mathop{\mathbb{E}}_{e,\pi}[r_{t+1} + \sum_{k=0}^{\infty} \gamma^k r_{t+2+k}|s_t = s] \\ &= \mathop{\mathbb{E}}_{a_t \sim \pi, s_{t+1} \sim e}[r_{t+1} + \gamma V^{\pi}(s_{t+1})] \end{aligned}$$

2.1.3 Optimal Policies

Let us suppose we have a Markov decision process and that there is a partial order over policies. We say $\pi \geq \pi' \iff \forall s, V^{\pi}(s) \geq V^{\pi'}(s)$.

Properties

- Existence of Optimal policy:

$$\exists \pi^*, \forall \pi, \pi^* \geq \pi$$

- All possible π^* have the same value functions:

$$\forall s, Q^*(s,a) = \max_{\pi} Q^{\pi}(s,a)$$

Bellman optimality equation

$$V^*(s) = \max_a \sum_{s'} p(s'|s,a)(R^a_{s,s'} + \gamma V^*(s'))$$

$R^a_{s,s'}$ is the average reward when taking action a in state s and arriving in s' and $p(s'|s,a)$ depends only on the environment.

Similarly:

$$Q^*(s,a) = \sum_{s'} p(s'|s,a)(R^a_{s,s'} + \gamma \max_{a'} Q^*(s',a'))$$

Optimizing the policy When you have a policy π you want to find the value function V^{π} (this is called policy evaluation)

First step: Estimating the value function An iterative approach works as follow. Let us start with any value function V_k.

$$\begin{aligned} V_{k+1}(s) &= \mathop{\mathbb{E}}_{\pi,e}[r_{t+1} + \gamma V_k(s_{t+1})|s_t = s] \\ &= \sum_a \pi(s,a) \sum_{s'} p(s'|s,a)(R^a_{s,s'} + \gamma V_k(s')) \end{aligned}$$

I am in state s_t. I need to take an action a_t. I arrive in a new state (s_{t+1}, r_{t+1}) with some probabilities. My estimation of r_{t+1} is exact but the estimation of $V_k(s_{t+1})$ is the one I assumed before (at iteration k). This converges to the real value function since $\gamma < 1$. Making use of $V_k(s_{t+1})$ to re-estimate $V_{k+1}(s_t)$ is named bootstrapping.

The proof of convergence is given using

$$\begin{aligned}
|(V_{k+1} - V^\pi)(s)| &= |\sum_a \pi(s,a) \sum_{s'} p(s'|s,a)[R^a_{s,s'} + \gamma V_k(s')) - R^a_{s,s'} - \gamma V^\pi(s')]| \\
&\leq \sum_a \pi(s,a) \sum_{s'} p(s'|s,a)\gamma |V_k(s') - V^\pi(s')| \\
&\leq \sum_a \pi(s,a) \sum_{s'} p(s'|s,a)\gamma \|V_k - V^\pi\|_\infty \\
&\leq \gamma \|V_k - V^\pi\|_\infty
\end{aligned}$$

We define the infinite norm by $\|V_{k+1} - V^\pi\|_\infty = \max_s |(V_{k+1} - V^\pi)(s)|$ and we find

$$||V_{k+1} - V^\pi||_\infty \leq \gamma ||V_k - V^\pi||_\infty$$

which gives convergence at an exponential rate:

$$||V_k - V^\pi||_\infty \leq \gamma^k ||V_0 - V^\pi||_\infty.$$

There exists an in-place version where updates for s are done in series rather than in parallel.

Policy improvement We have some policy π, we describe how good it is with V^π and Q^π. Define greedy policy π' which consists in taking the best action according to the previous value function:

$$\pi'(s) = \text{argmax}_a Q^\pi(s,a)$$

Then $\pi'(s) \geq \pi(s)$.

If no improvement the policy is optimal already. Doing **policy evaluation** and **policy improvement** repetitively is called **policy iteration**.

Let us introduce T the Bellman operator such that $T(\pi) = \pi'$ is the result of the policy improvement algorithm on π.

The optimal policy $\pi*'$ satisfies $T\pi* = \pi*$ which we can write $B\pi' = 0$ with $B = T - Id$.

The optimization we do is equivalent to a Newton Method-like step to find a 0 of B:

$$\pi_{k+1} = \pi_k - \left(\frac{dB}{d\pi}\right)^{-1} B(\pi_k)$$

A method called **truncated policy evaluation** works as follow. You don't wait for convergence, you just apply one step of policy evaluation before doing policy improvement. Still same guaranties of convergence. The proof is the same.

This can be generalized to any number of steps, the same guaranties always apply. With this dynamic programming algorithm, finding optimal policy is polynomial in the number of actions m and states s. Without this trick (naive evaluation of all policies) we would get exponential complexity $O(m^n)$.

2.2 Monte Carlo (MC) methods

The value function $V^\pi(s)$ is the average return in state s.

When running simulations, we are playing episodes = runs of the game. Averaging the empirical value function for a particular state seen during many episodes (one episode = one sample) might be a good evaluation of the value function.

Whether you consider only the first visit or every visit to the state, your estimate is always going to converge to $V^\pi(s)$ with precision $\frac{1}{\sqrt{n_{\text{samples}}}}$.

We have a fixed policy π you sample using Monte Carlo Q and you update your policy using the greedy policy improvement presented in last section.

2.2.1 On-Policy MC-control

We sample according to the policy we want to use. An additional hypothesis is that you have ϵ-soft policy with $\epsilon > 0$. This means $\forall s, a\ \pi(s,a) > \epsilon$ (ϵ-greedy is one example of such policy).

2.2.2 Off-Policy evaluation

We can evaluate π based on experiments with π' It requires that $\pi(s,a) > 0 \implies \pi'(s,a) > 0$. Consider the first visit to a state s. Let us denote p_i the likelihood of future state, reward and action for policy π and let us denote p'_i the likelihood of future state, reward and action for policy π'.

$$p_i(s) = \prod_{k=t}^{T-1} \pi(s_k, a_k) p(s_{t+1}|a_k, s_k)$$

$$p'_i(s) = \prod_{k=t}^{T-1} \pi'(s_k, a_k) p(s_{t+1}|a_k, s_k)$$

Instead of averaging like $V(s) = 1/n \sum_i R_i(s)$, we consider

$$\begin{aligned} V(s) &= \frac{1}{Z} \sum_i \frac{p_i(s)}{p'_i(s)} R_i(s) \\ &= \frac{1}{Z} \sum_i \frac{\pi(s_k, a_k)}{\pi'(s_k, a_k)} R_i(s) \end{aligned}$$

The behaviour policy is different than the estimation policy. The behaviour policy is used for sampling (running experiments). The estimation policy is the policy you really want to use in the end (the one you want to improve).

$\forall s, a$
$Q(s,a) \leftarrow random()$
$N(s,a) \leftarrow 0$
$D(s,a) \leftarrow 0$
$\pi \leftarrow random()$
repeat
generate an episode $s_0, a_0, r_0, s_1, a_1, r_1, ...$ with any soft policy π'
take the tail ($t > \tau$) so that all $a_{t>\tau}$ follows $\pi(s_t)$
$\forall (s,a)$ in that tail
$w \leftarrow \prod_{k=1}^{T-1} \frac{1}{\pi'(s_k,a_k)}$
$N(s,a) += wR_t$
$D(s,a) += w$
$Q(s,a) = \frac{N}{D}(s,a)$
$\forall s, \pi(s) = \text{argmax}_a Q(s,a)$
until enough iteration are made

On the good size of MC methods, there is no need to estimate environment bootstrap or probabilities. However, many explorations might be needed.

2.2.3 Temporal difference

Immediate MC on reward (not returns).

$$V(s_t) += \alpha \left([r_{t+1} + \gamma V(s_{t+1})] - V(s_t) \right)$$

with $\alpha \in [0, 1]$

This is bootstraping (using $V(s_{t+1})$).

Proved to converge to V^π if α is small enough or $\alpha_t \to 0$ and $\sum \alpha_t \to \infty$

You can also do this in batch mode: You run a number of episodes, average, and then update (faster convergence).

2.2.4 On-policy TD control: SARSA

We have a policy π we want to estimate the value $Q(s,a)$ (same as before replacing V with Q)

$$Q(s_t, a_t) += \alpha \left([r_{t+1} + \gamma Q(s_{t+1}, a_{t+1})] - Q(s_t, a_t) \right)$$

Update for each 5-uple $(s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1})$ hence the name SARSA.

π derived from Q needs to be soft e. g. ϵ-greedy.

2.2.5 Off-policy TD control: Q-learning

[Watkins, 1989] Breakthrough at that time because it is working better than other approaches.

- Exploitation

$$Q_t(s_t, a_t) += \alpha \left(\left[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) \right] - Q(s_t, a_t) \right)$$

- Exploration: Choose an action a using a soft policy derived from Q e.g ϵ-greedy (for softness).

Distinguishing the exploitation policy from the exploration one solves the *cliff example* issue that on-policy methods (such as SARSA) suffer from.

3 Eligibility traces

3.1 n-step prediction

We have different ways of estimating the return:

- Monte Carlo (run a full episode and get the sum of all rewards):

$$R_t = r_{t+1} + \gamma r_{t+2} + ... + \gamma^{T-t-1} r_T$$

- Temporal difference TD(0) (bootstrapping at t+1):

$$R_t^{(1)} = r_{t+1} + \gamma V_t(s_{t+1})$$

We can interpolate between these two methods (start running an episode and bootstrap at time t+n):

- n-step return

$$R_t^{(n)} = r_{t+1} + \gamma r_{t+2} + ... + \gamma^{n-1} r_{t+n} + \gamma^n V_t(s_{t+n})$$

How to choose the right value of n? Don't chose, mix all possibilities:

- Temporal difference TD(λ) (λ with $\lambda \in [0, 1]$)

$$R_t^\lambda = (1 - \lambda) \sum_{n=1}^{\infty} \lambda^{n-1} R_t^{(n)}$$

- Note, if terminal state reached in finite time: Temporal difference (λ with $\lambda \in [0, 1]$ and $T \leq \infty$)

$$R_t^\lambda = (1 - \lambda) \sum_{n=1}^{T-t-1} \lambda^{n-1} R_t^{(n)} + \lambda^{T-t-1} R_t$$

with R_t the complete return. When λ goes to 1, R_t^λ goes to R_t. When λ goes to 0, R_t^λ goes to $R_t^{(0)}$ which corresponds to Temporal difference (0)

3.2 n-step backup

Update of the value function for state s_t:

$$\delta(V_t(s_t)) = \alpha[R_t^{(n)} - V_t(s_t)]$$

$$V_{t+1}(s_t) = V_s(s_t) + \alpha[R_t^{(n)} - V_t(s_t)]$$

3.3 λ-return algorithm

$$\delta(V_t(s_t)) = \alpha[R_t^{\lambda} - V_t(s_t)]$$

only for one s_t.
R_t^{λ} is the target.

3.4 TD(λ) Backward view

Eligibility trace $e_t(s) \geq 0$ depends on the state and the time defined by:

$$\begin{aligned} e_t(s) &:= \gamma\lambda e_{t-1}(s) && \text{if } s \neq s_t \\ &:= \gamma\lambda e_{t-1}(s) + 1 && \text{otherwise} \end{aligned}$$

$\gamma\lambda < 1$ so it goes to 0 if you stay in the same state $s_t \neq s$.
For each particular state: it "jumps" by +1 if the state comes, and otherwise decreases exponentially. The eligibility $e_t(s)$ is the accumulation of fading traces.
λ is the decreasing speed of the trace (i.e. of the influence from the past).

TD (temporal difference) TD for prediction given the policy π:

$$\delta_t := [r_{t+1} + \gamma V_t(s_{t+1})] - V_t(s_t)$$

Defined only for the current state s_t.

For any state s,

$$\Delta V_t(s) := \alpha\delta_t e_t(s)$$

α is the update speed,
δ_t is the quantity update I want to add,
e_t is the trace. The trace is the credit to state s for what happens now at s_t.

One can show that this algorithm computes the right total update:

$$\sum_t \Delta V_t(s) = \alpha(R_t^{alpha}(s) - V_t(s))$$

Proof Very easy (TODO).

Particular case $TD(\lambda = 1)$ is the same as MonteCarlo method but with updates at each time for all states instead of waiting for the end of the game.

3.5 SARSA-λ

On-policy TD-control using Q

For all states s and action a:

- Define $e_t(s,a)$
- update of $Q_{t+1}(s,a) = Q_t(s,a) + \alpha\delta_t e_t(s,a)$

$$\delta_t = [r_{t+1} + \gamma Q_t(s_{t+1}, a_{t+1})] - Q_t(s_t, a_t)$$

$$\begin{aligned} e_t(s,a) &= \gamma\lambda e_{t-1}(s,a) && \text{if}(s,a) \neq (s_t, a_t) \\ &= \gamma\lambda e_{t-1}(s,a) + 1 && \text{otherwise} \end{aligned}$$

On-policy: you update the policy according to an ϵ-soft policy based on Q (i.e ϵ-greedy).

3.6 $Q(\lambda)$

Update is the following:

$$Q_{t+1}(s,a) = Q_t(s,a) + \alpha\delta_t e_t(s,a)$$

with $\delta_t = [r_{t+1} + \gamma \max_a Q_t(s_{t+1}, a')] - Q_t(s_t, a_t)$
and $e_t(s,a) = \mathbb{1}_{s=s_t \wedge a=a_t} + \mathbb{1}_{Q_{t-1}(s_t,a_t)=\max_a Q_{t-1}(s_t,a)}\gamma\lambda e_{t-1}(s,a)$.
Transmission only if optimal action is chosen.

Until now we have estimated only a value for a state $V(s)$ one value for each state s. It is a huge vector in $\mathbb{R}^{|S|}$ with $|S|$ the number of states. This can lead to huge dimensions problem. You might want to consider a much smaller set of states using a parameterization of the problem.

3.7 Parametrized familly $V_\theta(s)$

You compute $f_\theta(s)$ with f_θ a function depending on parameters θ (like a neural network) and you do the learning on $f_\theta(s)$, i.e. by learning the right value of θ.

An update $\delta\theta$ of the vector of parameters θ leads to an update of the value function:

$$V_{\theta+\delta\theta}(s) = V_\theta(s) + \frac{dV(s)}{d\theta}\delta\theta.$$

Reciprocally, given a desired target value δV, if $\frac{dV(s)}{d\theta}$ is invertible, the parameters θ should be updated as: $\delta\theta = (\frac{dV(s)}{d\theta})^{-1}(\delta V(s))$.

The target values δV can be:

- $DP : \mathbb{E}_{\pi,e}[r_{t+1} + \gamma V_t(s_{t+1})|s_t = s]$
- $MC : R_t$
- $TD(0) : r_{t+1} + \gamma V_t(s_{t+1})$
- $TD(\lambda)$: see above

Policy evaluation $\pi \mapsto V^\pi$

$V_{\theta_t}(s)$ does not cover the full space of states so we cannot always find a θ such that V_theta would be exactly V^π for all s.

How can we say that a θ is better than another one ? You can define a criterion (i.e Mean Square Error):

$$\begin{aligned} MSE(\theta_t) &= \sum_s p(s)[V^\pi(s) - V_{\theta_t}(s)]^2 \\ &= \mathop{\mathbb{E}}_{\pi,e}[(V^\pi - V_{\theta_t})^2] \end{aligned}$$

We can estimate θ using gradient descent:

$$\theta \ -= \ \tau \nabla_\theta MSE(V_\theta)$$

$$\theta_{t+1} = \theta_t \ + \ 2\tau \left(V^\pi(s_t) - V_{\theta_t}(s_t)\right) \nabla_\theta V_{\theta_t}(s_t)$$

θ_{t+1} is a vector, the new values of the parameters.
$\nabla_\theta V_{\theta_t}(s_t)$ is a vector of same dimension as θ_t.
$V^\pi(s_t)$ is the target (a real number) (see possible target values above), and $V_{\theta_t}(s_t)$ is the current value. The value of the target $V^\pi(s_t)$ might be unknown, in which case one can replace it with an approximation v_t of it.

If the approximation v_t of the target $V^\pi(s_t)$ is an unbiased estimator, i.e. if $\mathbb{E}_{\pi,e}[v_t] = V^\pi(s_t)$, then θ_t converges to a local optimum provided α_t goes to 0 and $\sum_t \alpha_t$ goes to infinity when t goes to infinity.

If $v_t = R_t^\lambda$ it is bootstraping (this is biased).

Approximation $v_t = R_t^\lambda$:

$$\theta_{t+1} = \theta_t + \alpha \left[(r_{t+1} + \gamma V_{\theta_t}(s_{t+1})) - V_{\theta_t}(s_{t+1})\right] e_t$$

where
$e_t = \gamma\lambda e_t + \nabla_\theta V_{\theta_t}(s_t)$
$e_0 = 0$

Example of a parameterized family: the linear case

$$V_{\theta_t}(s) \ = \ < \theta_t | \phi(s) > \ = \ \sum_i \theta_t^i \phi_i(s)$$

where $\phi(s)$ is a representation of state s (features). Note that $\phi(s)$ and θ are vectors (of same dimension). $V_{\theta_t}(s)$ is a linear combination of the features $\phi_i(s)$ with weights θ^i.

Gradient descent on θ: $\nabla_\theta V_{\theta_t} = \phi(s)$

In the case of a convex problem (such as MSE) with an unbiased estimate of $V^\pi(s)$: we will reach the globally optimum parameters.

In the $TD(\lambda)$ case, the estimate is biased, and we get only the following convergence guarantee:

$$MSE(\theta_\infty) \leq \frac{1-\gamma\lambda}{1-\gamma} MSE(\theta^*).$$

Which features $\phi(s)$?

- Best case: Modeling.
- Rich enough features (kernels, neural networks)

Control with function approximation:

Gradient descent on parameters:

$$\theta_{t+1} = \theta_t + \alpha(v_t - Q_{\theta_t}(s_t, a_t))\nabla_{\theta_t} Q_{\theta_t}(s_t, a_t),$$

where v_t is a real number, it is the target. Many choices are possible, for example $r_{t+1} + \gamma Q_{\theta_t}(s_{t+1}, a_{t+1})$ (bootstrap).
$\nabla_\theta Q_{\theta_t}(s_t, a_t)$ is the gradient vector.
We can also use traces: $e_t = \gamma\lambda e_{t-1} + \nabla_\theta Q_{\theta_t}(s_t, a_t)$.
With these particular settings and with an ϵ-soft policy derived from Q_{θ_t}, one gets the "function approximation" version of SARSA(λ).
Without bootstrap: one obtains a real gradient descent on an energy (so you have good properties).
With bootstrap: one gets only near-optimal guaranties. If on-policy: not an issue, if off-policy it can even diverge.

The Mean Square Error as a criterion to optimize is also a matter of choice. Example of alternative to MSE:

$$\sum_s p(s) \mathop{\mathbb{E}}_{\pi,e} \left[\left((r_{t+1} + \gamma V_{\theta_t}(s_{t+1})) - V_{\theta_t}(s) \right)^2 \right].$$

The difference is that θ_t appears twice (and both occurrences will appear in the computation of the gradient).

4 Policy Gradient

4.1 New framework: Optimize policy directly

We estimate directly the policy $\Pi(a|s,\theta)$ itself. θ is our parametrization of the problem.

Example - Softmax

$$\Pi(a|s,\theta) = \frac{exp(h(s,a,\theta))}{\sum_b exp(h(s,b,\theta))}$$

This is a general way of expressing probability distributions.

4.2 Policy Gradient Theorem

The goal is to maximize some criterion.

$$E(\theta) = v_{\Pi_\theta}(s_0)$$

For example: $v_{\Pi_\theta}(s_0)$ can be the expected return from state s_0 under policy Π_θ or the average reward over time.

Let us state the policy gradient theorem

$$\Delta_\theta E = \sum_s d_\Pi(s)(\sum_a Q_{\Pi_\theta}(s,a)\Delta_\theta \Pi(a|s,\theta))$$

$d_\Pi(s)$ is the probability to be in state s when following Π_θ for a long while (stationary distribution).

If a quantity $q(s)$ is added to $Q_{\Pi_\theta}(s,a)$ I it does not change $\Delta_\theta E$ (proof is easy).

4.3 Reinforce algorithm, Actor Critic, Eligibility traces

4.3.1 Reinforce

It is a Monte Carlo policy gradient.

We have

$$\begin{aligned}\Delta_\theta E &= E_{s_t\ \Pi_\theta,e}[\gamma^t \sum_{a_t} Q_{\Pi_\theta}(s_t,a_t)\Delta_\theta \Pi(a_t|s_t,\theta) \\ &= E_{a_t,s_t\ \Pi_\theta,e}[\gamma^t Q_{\Pi_\theta}(s_t,a_t)\frac{\Delta_\theta \Pi(a_t|\theta,s_t)}{\Pi(a_t|s_t,\theta)} \\ &= E_{s,a,R\ \Pi_\theta,e}[\gamma^t R_t \Delta_\theta log \Pi(a_t|s_t,\theta)]\end{aligned}$$

R_t is the return from time t. The algorithm is simply the following:

$$\theta_{t+1} = \theta_t + \alpha\gamma^t R_t \Delta_\theta log(\Pi_\theta(a_t|s_t,\theta)$$

4.4 Actor Critic

As said before we can any quantity to R as long as it does not depend on the action. So we can replace R_t by $R_t - V(s_t)$ which balances the amplitudes and reduces the variance. In this case we should have to parametrize V with an other parameter θ_2 and do the optimization like in the previous courses.

Actor-critic is this process of comparing the return (or expected return, or reward depending on the settings) to the ones you would get performing another action (we are doing this when subtracting the average).

If you use Actor-critic and bootstrap you get:

$$\theta_{t+1} = \theta_t + \alpha(r_{t+1} + \gamma V(s_{t+1,\theta_2}) - V(s_t, \theta_2))\Delta_\theta log(\Pi(a_t|s_t, \theta))$$

$r_{t+1}+\gamma V(s_{t+1,\theta})]$ is the estimate of the return R_t (Monte Carlo) and $-V(s_t, \theta_2))$ is the actor-critic model.

$$\theta_2 = \theta_2 - \alpha_2(r_{t+1} + \gamma V(s_{t+1,\theta_2}) - V(s_t, \theta_2))\Delta_{\theta_2} V$$

4.4.1 Eligibility traces

$$e_{t+1} = \lambda e_t + I\Delta_\theta log\Pi_\theta(a|s)$$

with I defined such as

$$\begin{aligned} I &= 1 && \text{if} t = 0 \\ I_{t+1} &= \gamma I_t && \text{otherwise} \end{aligned}$$

$$\theta = \theta + \alpha dte$$

with dt defined as

$$dt = (r_{t+1} + \gamma V(s_{t+1,\theta_2}) - V(s_t, \theta_2))$$

For the value function updates are defined as :

$$e_2 = \lambda e_2 + I\Delta_{\theta_2} V_{\theta_2}(s)$$

and

$$\theta_2 = \theta_2 + \alpha dte_2$$

4.4.2 Different Goal

In case of a very long game you might prefer to estimate the average reward $E(\theta)$ is the average reward.

If is the same reasoning but with

$$\delta = r_{t+1} - \bar{t} + V_{\theta_2}(s_{t+1}) - V_{\theta_2}(s_t)$$

define as Return $-T\overline{r}$

$\overline{r}$ is the average reward.

4.4.3 Deep Mind DQN: Atari Games

Q_{θ_t} is the target we use $E[r + \gamma max_a Q]$ The loss to minimize is

$$U_{Q_{t+1}} = E[target - Q_{\theta_{t+1}}$$

Using gradient descent.

Q_{θ_t}: Neural network 3 layers

- conv
- conv
- fully connected

Input: 4 frames 84×84 pixels
Output: 18 actions

$$\theta_{t+1} = \theta_t + \alpha[r_{t+1} + \gamma max_a Q_\theta(s_{t+1}, a) - Q_\theta(s_t, a_t)]$$

4.4.4 Monte Carlo Tree search

Minimax approach You alternate the best action that you can take and the best action the adversary can take (which is the worst for you).

In practice you cannot go very far in the tree with that approach. When you have to stop you can use estimate of your probability to win.

$\alpha - \beta$ **prunning** Branch and bound to discard branches that cannot lead to a better score.

Upper confidence Tree Use bandits, one bandit per action. This was done in CrazyStone (a go player). We take the arm i that maximizes the following quantity:

$$\overline{r_i} + \sqrt{\frac{logn}{t_i(n)} F_i}$$

with $F_i = min(1/4, r_i^2 + \sqrt{\frac{2logn}{T_i(n)}})$

At some point you reach the end of the game. The good thing is that bandits give you a policy without having to estimate the probability to win.

MOGO = Monte Carlo Tree SEarch (MCTS Bandits each time you play. Instead of developping the full-tree. I always follow the bandit used so far. When I am in a new state, I play an average random game. You have an estimate of the state, action value and create a new bandit node for that action. You remember only the nodes seen more than twice.

Alpha-GO See the paper in Nature: https://storage.googleapis.com/deepmind-media/alphago/AlphaGoNaturePaper.pdf