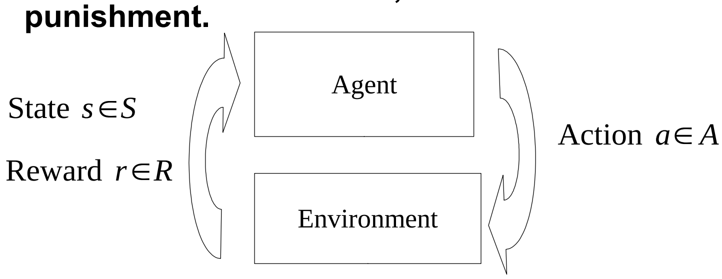
Reinforcement Learning

Philippe Pasquier
Office 565 (floor 14)
pasquier@sfu.ca

SFU

Reinforcement Learning

- **Reinforcement learning design a family of approaches describing how an agent can learn from success and failure, reward and punishment.**



The target function is a control policy $\pi : S \rightarrow A$

- **The agent's goal is to maximise the cumulated reward over time**

Examples of reinforcement learning

Examples of reinforcement learning:

- •Playing chess: Reward comes at end of game
- •Ping-pong: Reward on each point scored
- •Animals:
  - •Hunger and pain - negative reward
  - •food intake – positive reward

Many real world applications:

- •TD-Gammon (backgammon top player)
- •Robotics

Reinforcement Learning

- Notice how this differ from supervised learning
- While it is still function approximation:
  - Instead of getting $<s, \pi(s)>$ for the learning, we get the reward(s) and it does not directly give $\pi(s)$
  - Temporal credit assignement: the reward is not saying what actions are to be credited (example of chess)
  - The training examples themselves are influenced by the agent behavior: exploration of new states or exploitation of states that are already known to yeld high rewards (but not necessarily the highest!)
  - The agent behavior is interwined with the learning (not always though)
  - Life-long learning: not an isolated function aproximation task, several tasks have to be learned in parallel

Markov Decision Process

- **The basic framework for reinfocement learning is Markovian Decision Process (MDP):**

A MDP is defined as a tuple $< S, t, A, r >$, where:

- S is a finite set of distinct states
- A is a discrete set of actions
- $t: S * A \rightarrow S$ is a transition function $t(s_t, a_t) = s_{t+1}$
- $r: S * A \rightarrow R$ is a reward function $r(s_t, a_t) = r_t$

and: t and r just depend on the current state and action.

Note: In general t and r can be non-deterministic (stochastic).

Markov Decision Process

- **Discounted cumulative reward:**

$$V^{\pi}(s_t) = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots$$

$0 \leq \gamma < 1$ is the discount factor: finite of infinite horizon

If $\gamma = 0$, only the immediate reward is considered

The highier is γ the more the future matters

- **We want the agent to learn the policy that maximises the discounted cumulative reward for all states:**

$$\pi^* = argmax_{\pi} V^{\pi}(s), \forall s \in S$$

We note V^* the value function of the optimal policy

Markov Decision Process

- We want to learn how to choose the optimal action and: $\pi^*(s) = \underset{a}{argmax}[\underbrace{r(s,a) + \gamma V^*(t(s,a))}_{Q(s,a)}]$
- Assuming that we can learn V*, we would still need to know r() and t() and we don't!
- The idea is to learn Q through iterative approximation using a recursive equation:

$$V^*(s_t) = \max_{a_t} [r(s_t, a_t) + \gamma V^*(\underbrace{t(s_t, a_t)}))] = \max_{a_t} Q(s, a_t)$$

$$V^*(s_t) = \max_{a_t} Q(s_t, a_t), \text{ and } V^*(s_{t+1}) = \max_{a_{t+1}} Q(t(s_t, a_t), a_{t+1})$$

$$Q(s_t, a_t) = r(s_t, a_t) + \gamma \max_{a_{t+1}} Q(t(s_t, a_t), a_{t+1})$$

Q-learning algorithm

1) For each s, a initialise the table entry $\hat{Q}(s,a)$ to 0

2) Observe the initial state: $s_t \leftarrow s_0$

3) Select and action a_t and execute it

4) Receive immediate reward r_t

5) Observe the new state s_{t+1}

6) Update table entry for $\hat{Q}(s_t, a_t)$:

$$\hat{Q}(s_t, a_t) \leftarrow r_t + \gamma \max_{a_{t+1}} \hat{Q}(s_{t+1}, a_{t+1})$$

7) $s_t \leftarrow s_t + 1$

8) Go to step 3)

Q-learning convergence

- Theorem: Q-learning converges toward the true Q values iff:
  - The MDP is deterministic
  - Rewards are bounded by a constant c
  - Each state-action pair is visited infinitely often
- Proof: the proof consists in showing that the maximum error over the estimated Q values is decreasing each time all the states are visited and eventually converge (the error's limit is null).
- In practice, we do not need an infinite number of visits (but many thousands)

Q-learning algorythm

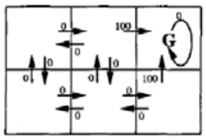
- We did not specify how the agent selects the action to execute. There are several possibilities:
  - Take a random one (exploration)
  - Exploiting our Q-value: $a = \underset{a_t}{argmax}\,\hat{Q}(s_t, a_t)$
  - That is exploiting what we know, but because of the preceding theorem we need to find a balance between exploitation and exploration
  - We use a probabilistic approach:

$$P(a_i|s) = \frac{k^{\hat{Q}(s,a_i)}}{\sum_j k^{\hat{Q}(s,a_j)}}, \text{ where } k>0$$

the larger is k the more probable actions with $\hat{Q}$ values above average will be

  - Often, k is gradually increased with the number of iterations

Example



$r(s,a)$ (immediate reward) values

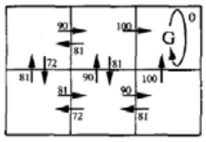
We assume $\gamma=0.9$

Only 6 states

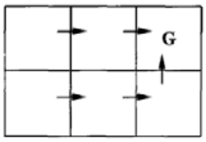
$A=\{$up, down, left, right, still$\}$

G is an absorbing state

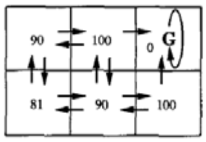
t is determinist: $t(G, still)=still$



$Q(s,a)$ values

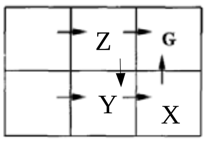


One optimal policy

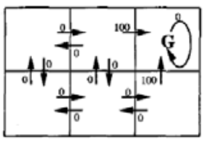


$V^*(s)$ values

Example

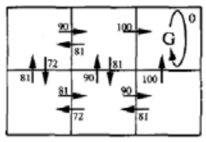


One optimal policy



$r(s,a)$ (immediate reward) values

The first time G is visited from X (t(X,up) is consumed):
Q(X,up)=100
The first time X is visited from Y (t(Y,right) is consumed):
Q(Y,right)=0 + 0.9*100=90
The first time Y is visited from Z (t(Z,down) is consumed):
Q(Z,Down)=0+ 0.9*90=81
....



$Q(s,a)$ values

Q-learning extensions

- Since the lookup table can be very big, a neural network can be used to store/approximate the Q function.
- Extension to the non-deterministic case:
  - The reward and/or transition functions can be non-determinist (in particular stochastic)
  - Example of the TD-gammon (the use of a dice make the transition function stochastic)
  - A non-deterministic MDP is one for which the probability distribution for t(s,a) and r(s,a) only depend on s and a
  - The main difference is that we then deal with expected cumulated values over these non-deterministic outcomes.

Q-learning extensions

- The Q-learning algo learns by iteratively reducing discrepency between Q value estimates for adjacent states. In that sens, it is a special case of temporal difference algorithms (that can deal with more distant descendants or ancestrors)
    - TD(λ) is a generalisation of Q-learning
    - Q-learning is equivalent to TD(0)
- Othertypes of reward can be used:
  - Discounted cumulated rewards over a finite horizon
  - Average reward
  - More complex reward functions

Conclusion on Reinforcement Learning

- Reinforcement learning addresses the problem of learning a control strategy for autonomous agents
- It assumes that the training information is available as a real-valued reward signal and that the goal of the agent is to maximise the cumulated discounted reward
- Q-learning provides a solution to that problem in both the deterministic and nondeterministic cases.
- Q-learning is a particular type of temporal difference learning algorithm.

?

"Live as if you were to die tomorrow.
Learn as if you were to live forever."

Mahatma Gandhi