# Reinforcement Learning:
## A Tutorial.

L. O. Olagoke[1]

[1]School of Computer And Communication Systems
Ecole Polytechnique Federale de Lausanne

Statistical Seq Processing
June 2017

# Table of Contents

# Reinforcement Learning: What is it about?

It is about (the agent) learning from interactions (with environment) to achieve a goal by taking actions aimed at maximizing numerical reward. It is simply learning the act of decision making.

The learner and decision maker is called the agent. The thing it interacts with, comprising of everything outside the agent is called the environment. The learner is not told which action to take as in most forms of machine learning but instead must discover which actions maximize a reward by trying them out. The environment respond to those actions and present a new situation.

# Difference from other Machine Learning Paradigms

How does Reinforcement Learning differs from other Machine learning paradigms?

- There is no supervision only a reward signal
- Trial-and-error search and delayed reward
- Feedback is delayed, not instantaneous
- Time matters (sequential non i.i.d data points)
- Agent action affect the response it gets
- RL is defined by characterizing a learning problem and not by characterizing a learning method. The idea here is to characterize the important aspect of the real problem facing the agent interacting with the environment to achieve a goal through its actions. The goal of the agant must relate to the state of the environment.

# Reward

A Reward is a scalar feedback signal. It indicates how well the agent is doing at step t. The agent goal is to maximise cumulative reward. Reinforcement Learning is based on reward hypothesis.

**Definition - Reward Hypothesis**

All goals can be described by the maximization of expected cumulative reward. ??

# Agent and Environment

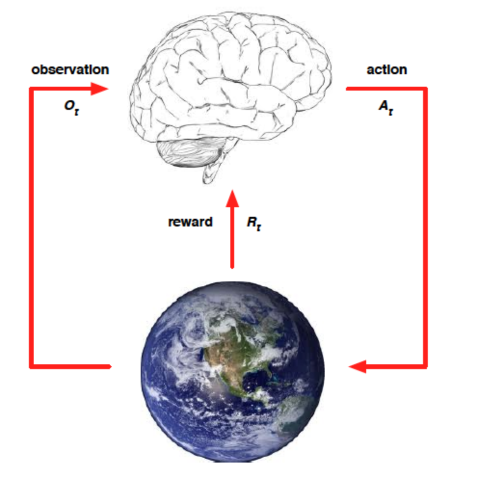



Figure: Agent and Environment

- At each step t the agent:
  - Executes action $A_t$
  - Receives Observation $O_t$
  - Receives scalar reward $R_t$
- The environment:
  - Receives action $A_t$
  - Emits observation $O_{t+1}$
  - Emits scalar reward $R_{t+1}$
- t increments at environment step.

# Environment State

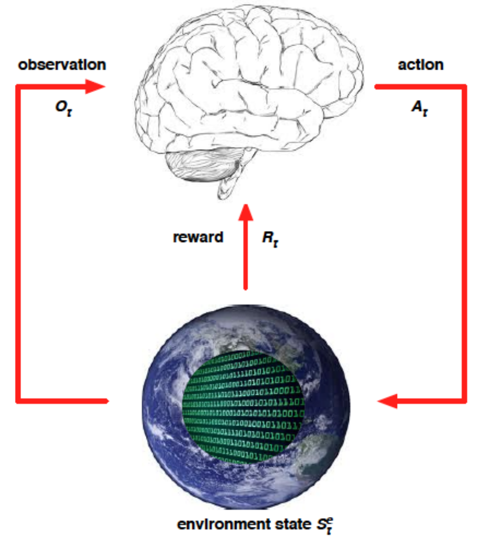



Figure: Agent and Environment

- The environment sate $S^e_t$ is the environment's private representation. It is whatever process/algorithm/data by which the environment chooses its next observation or reward.
- The environment state is not usually visible to the agent.
- Even if the $S^e_t$ is visible to the agent, it may contain irrelevant information.

# Agent State

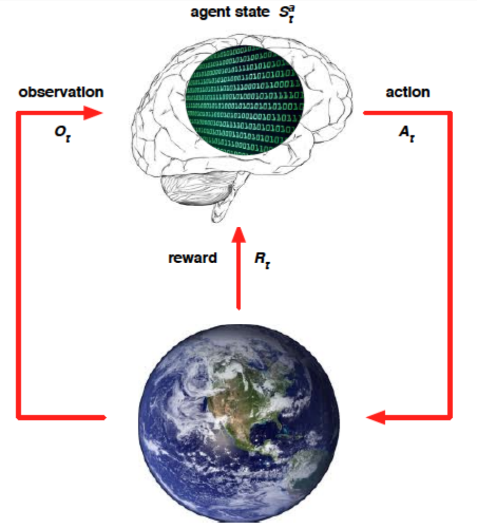



Figure: Agent and Environment

- The agent sate $S_t^a$ is the agent's private representation.
- It is the information used by learning algorithm
- It can be any function of history

$$S_t^a = f(H_t)$$

# Partially And Fully Observable Environments

- Full observability: agent directly observes environment state.
- Agent State = Environment State = Markov State
- This defines Markov Decision Process

- Partial observability: agent indirectly observes environment.
- Agent state $\neq$ Environment state
- This defines a Partially Observabe Markov Decision Process (POMDP)

In POMDP, the agent must construct its own state representation $S_t^a$. Using for E.g.

- Complete History : $S_t^a = H_t$
- Beliefs of the environment : $S_t^a = (\mathbb{P}[S_t^e = s^1], ...\mathbb{P}[S_t^e = s^n])$
- Recurrent neural network. $S_t^a = \sigma(S_{t-1}^a W_s + O_t W_o)$

# RL Agents Components

- Policy.

## Definition

Agent behaviour function. Roughly, we say it is a mapping from perceived states to actions to be taken when in those states. It is sufficient enough to determine agent's behaviour.

- deterministic policy: action taken in state s under deterministic policy $\pi$.

$$a = \pi(s)$$

- Stochastic policy: probability of taking action a in state s under stochastic policy $\pi$.

$$\pi(a|s) = P(A_t = a|S_t = s)$$

# RL Agents Components

- Policy.
- Value Function

## Definition

Value function of a state is the total reward an agent is expected to accumulate over the future starting from that state. It specifies how good a action or function is. E.g

$$V_\pi = E_\pi \left( R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots \mid S_t = s \right)$$

# RL Agents Components

- Policy.
- Value Function
- Model

**Definition**

Model: is the agent representation of the environment. It is whatever an agent can use to predict how the environment will respond to its actions.

- $\mathcal{P}$ predicts the next state [1]:

$$\mathcal{P}_{ss'}^{a} = \mathbb{P}\left[S_{t+1} = s' \mid S_t = s, A_t = a\right]$$

- $\mathcal{R}$ predicts the next immediate reward.

$$\mathcal{R}_{s}^{a} = \mathbb{E}\left[R_{t+1} \mid S_t = s, A_t = a\right]$$

[1] We assume that the Markov property is satisfied

# Categorizing RL Agents (1)

- Value Based
  - Value Function
  - No Policy (Implicit)
- Policy based
  - Policy
  - No Value Function
- Actor critic
  - Policy
  - Value Function
- Model Free
  - Policy and/or Value Function

# Exploration Vs Exploitation

- Exploration: involves finding more information about the environment
- Exploitation involves exploiting known information to maximise reward
- Balance between Exploitation and exploration should be sought
- Prediction : evaluate future, given a policy
- control : Optimise future , by finding best policy

# Markov Reward process

A Markov Reward process is a Markov chain with values.

## Definition

A Markov Reward process is a tuple $\langle \mathcal{S}, \mathcal{P}, \mathcal{R}, \gamma \rangle$ where

- $\mathcal{S}$ is a (finite) set of states.
- $\mathcal{P}$ is a state transition probability matrix:
  $\mathcal{P}_{ss'} = \mathbb{P}\left[S_{t+1} = s' \mid S_t = s\right]$
- $\mathcal{R}$ is a reward function: $\mathcal{R}_s = \mathbb{E}[R_{t+1} | S_t = s]$
- $\gamma$ is discount factor, $\gamma \in [0,1]$

# Return

## Definition

The Return $\mathcal{G}_t$ is the total discounted reward from time-step t

- $\mathcal{G}_t = R_t + \gamma R_{t+2} + ... = \sum_{k=1}^{\infty} \gamma^k R_{t+k+1}$

- $\gamma$ is the discount rate $\gamma \in$ [0,1]
- value of receiving reward R after k+1 step is $\gamma^k R$
- This value immediate reward above delayed reward.
  - $\gamma$ close to 0 leads to "myopic" evaluation.
  - $\gamma$ close to 1 leads to "far-sighted" evaluation.

# Bellman equation for Markov Reward Process (MRP)

$$
\begin{aligned}
\text{v(s)} &= \mathbb{E}[G_t | S_t = s] \\
&= \mathbb{E}[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ... | S_t = s] \\
&= \mathbb{E}[R_{t+1} + \gamma (R_{t+2} + \gamma R_{t+3} + ...) | S_t = s] \\
&= \mathbb{E}[R_{t+1} + \gamma (\mathcal{G}_{t+1}) | S_t = s] \\
&= \mathbb{E}[R_{t+1} + \gamma v(S_{t+1}) | S_t = s]
\end{aligned}
$$

# Markov Decision Process

A Markov Decision process (MDP) is a Markov reward process with decisions. It is an environment in which all states are Markov.

## Definition

A Markov Decision process is a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$ where

- $\mathcal{S}$ is a (finite) set of states.
- *$\mathcal{A}$ is a finite set of actions*
- $\mathcal{P}$ is a state transition probability matrix:
  $\mathcal{P}_{ss'}^{a} = \mathbb{P}\left[S_{t+1} = s' \mid S_t = s, A_t = A\right]$
- $\mathcal{R}$ is a reward function: $\mathcal{R}_s^a = \mathbb{E}[R_{t+1} | S_t = s, A_t]$
- $\gamma$ is discount factor, $\gamma \in [0{,}1]$

# Value Function

## state-value function

A state-value function $v_\pi(s)$ of an MDP is the expected return starting from state s, and then following a policy $\pi$

$$v_\pi(s) = \mathbb{E}_\pi \; [ \; \mathcal{G}_t | S_t = s \; ]$$

The Bellman equation for $v_\pi(s)$ is:

$$v_\pi(s) = \mathbb{E}_\pi \; [ \; R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s]$$

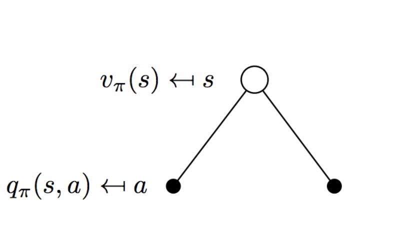
## Action-Value Function

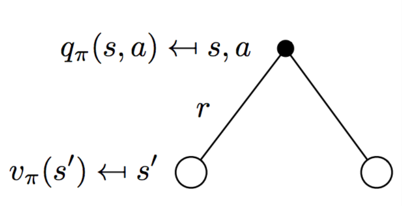
The action-value function $q_\pi$(s,a) is the expected return starting from state s, taking action a, and then following policy $\pi$.

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

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

# Bellman Equation for $V^\pi$ and $Q^\pi$




$$v_\pi(s) = \sum_{a \in A} \pi(a|s) q_\pi(s,a) \qquad q_\pi(s,a) = \mathcal{R}_s^a + \gamma \sum_{s' \in S} \mathcal{P}_{ss'}^a (a|s) v_\pi(s')$$

# Value Function Prediction:Temporal Difference Learning

- Temporal difference learning can be used to estimate the value function of an unknown MDP. TD is a combination of MC and dynamic programming ideas.
- TD methods can learn directly from raw experience without a model of the environment dynamics.
- TD learns from incomplete episodes , by bootstrapping.
- Updates a guess towards a guess

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

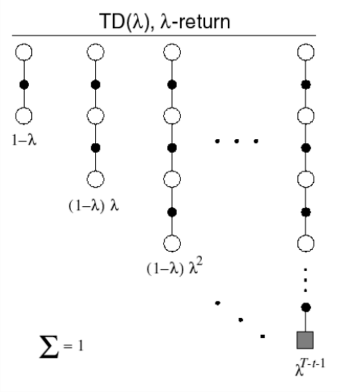
# TD Learning

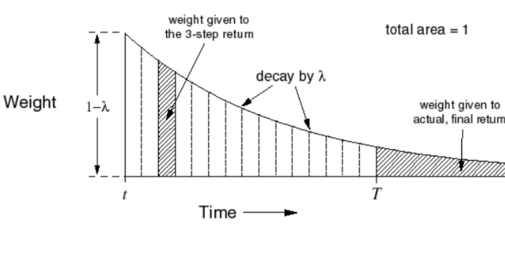
- Goal: Lean $v_\pi$ online under policy $\pi$.
- Simplest temporal difference learning : TD(0)
  - Update value $V(S_t)$ toward estimated return: $R_{t+1} + \gamma V(S_{t+1})$

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

  - $R_{t+1} + \gamma V(S_{t+1})$ is called the TD target
  - $\delta_t = R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$ is called the TD error
  - TD works in continuing environment
  - TD has low variance , some bias
  - TD can learn online after every step

# TD ($\lambda$) and Weighting Function




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

$$V(S_t) \leftarrow V(S_t) + \alpha\left(G_t^{\lambda} - V(S_t)\right)$$

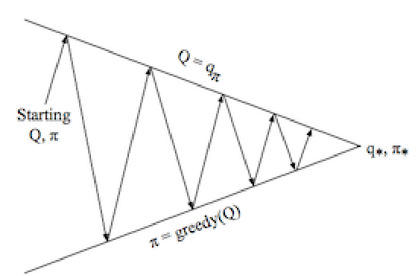
# Model Free Control Introduction

Goal: Optimize the value function of an unknown MDP.
Examples: Robot Walking, Ship Steering, Helipcopter, Aeroplane logistics, Portfolio Management. etc

- On policy Learning: learns action value relative to the policy it follows. Or learning about policy $\pi$ from experience sampled from policy $\pi$ (sequence of actions) Example SARSA
- Off-policy learning: Learn about policy $\pi$ from experience sampled from $\mu$. The value assigned to a given state (or state-action pair) is a function of the immediate reward and of the maximum rewards received in the subsequent states during the episode. E.g Q learning.

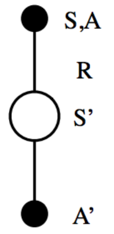
# Action-value Function

- Greedy policy improvement over Q(s,a) $\pi'(s) = \underset{a \in A}{\text{argmax}}\ Q(s,a)$
- Can be reduced to "Every episode" and then use $\epsilon$-greedy policy per episode




- Policy evaluation: Monte carlo evaluation , Q $= q_\pi$
- Greedy policy vs exploration.

# SARSA And TD control




- use TD learning
- use $\epsilon$-greedy policy improvement
- $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(q_t^\lambda - Q(S_t, A_t))$

- Every time step
- Policy evaluation, SARSA
- Policy improvement $\epsilon$-greedy.

# Q-Learning: Off Policy Learning

We now consider off-policy learning of action-values Q(s,a).

- Next action is chosen based on behaviour policy $A_{t+1} \sim \mu(\cdot|S_t)$
- We update Q($S_t$,$A_t$ ) towards alternative action:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(R_{t+1} + \gamma Q(S_{t+1}, A') - Q(S_t, A_t))$$

- Note that in off-policy learning we evaluate the target policy $\pi(a|s)$to compute $V_\pi$ or q$\pi(s, a)$ while following the behavior policy $\mu$(a—s) :

$$\{S_1, A_1, R_2 ... S_T\} \sim \mu$$

- This might be important in order for agents to learn from other agents, or reuse old policies, or learn more optimal policies, or learn about multiple policies.

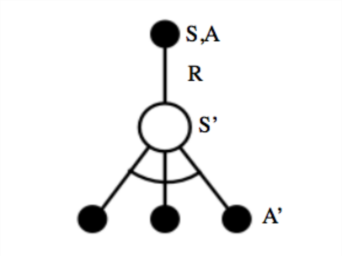
ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

# Off Policy control with Q-learning

Here we allow both behavior and target policies to improve

- Target policy is greedy w.r.t Q(S,a) $\pi(S_{t+1}) = \underset{a'}{\text{argmax}}\ Q(S_{t+1}, a')$
- The behavior policy $\mu$ is e.g $\epsilon$-greedy w.r.t Q(s,a)
- Sarsa control algorithm :

$$Q(S, A) \leftarrow Q(S, A) + \alpha(R + \gamma \max'_a\ Q(S', A') - Q(S, A))$$



# Why Value Functions Approximation ?

- Real world problems can present a very large number of states. Thus it impractical to evaluate the value of each state ( computational and memory requirement)
- The idea of function approximation is to estimate the value of path visited in the state in the state space and generalize this thus estimate across state space.
- How do we solve large MDP:
    - $\hat{v}(s, \mathbf{w}) \approx v_\pi(s)$
    - $\hat{q}(s, a, \mathbf{w}) \approx q_\pi(s,a)$
- We can now Generalise to unseen states and update parameter w using TD learning or MC learning.
- We consider only differential function approximator: Neural nets and Linear Function approximators.

# Stochastic Gradient descent

- Define J($\mathbf{w}$) as our objective function. Its gradient is defined as $\Delta_{\mathbf{w}}$J($\mathbf{w}$). Then we can adjust $\mathbf{w}$ in direction of negative gradient: $\Delta\mathbf{w}{=}{-}\frac{1}{2}\alpha\Delta_{\mathbf{w}}J(\mathbf{w})$
- Define J($\mathbf{w}$) $= \mathbb{E}_{\pi}[(v_{\pi}(s) - \hat{v}(s, \mathbf{w}))^2])$ . Then $\Delta\mathbf{w}{=}\alpha\mathbb{E}_{\pi}[(v_{\pi}(s) - \hat{v}_{\pi}(s, \mathbf{w}))]\Delta_{\mathbf{w}}\hat{v}_{\pi}(s, \mathbf{w})$
- For stochastic gradient we have that: $\Delta\mathbf{w}{=}\alpha\mathbb{E}_{\pi}[(v_{\pi}(s) - \hat{v}_{\pi}(s, \mathbf{w}))\Delta_{\mathbf{w}}\hat{v}_{\pi}(s, \mathbf{w})$
- For a linear approximator we have that: $\hat{v} = \mathbf{x}(S)^T\mathbf{w} = \sum_{j=1}^{n}\mathbf{x}_j(S)\mathbf{w}_j$
- We can substitute this into the equation above.

# Conclusions

- Agents learns from the environment following a policy and takes action each time towards attaining a goal
- This can be extended to POMDP.
- Use functional approximation to evaluate state values.
- Can all goals be really defined by the maximization of expected cumulative reward (philosophical??)