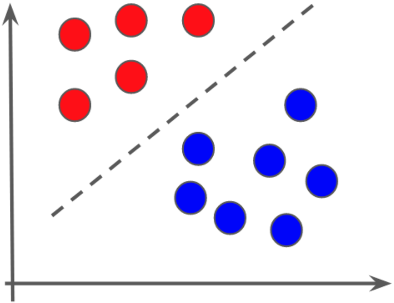
Machine Learning
Reinforcement learning

S. Herbin

stephane.herbin@onera.fr

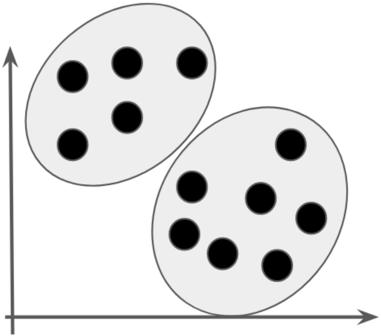
Introduction

Supervised Learning



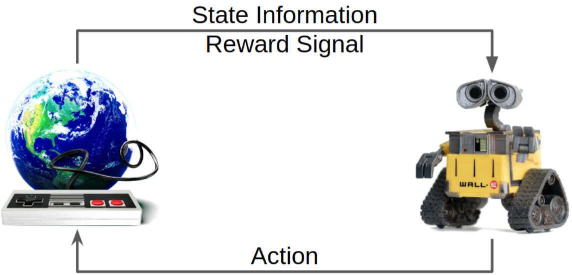
- Data: Features + Labels
- Task: Discriminate classes based on features
- Learning objective: find the good decision that minimizes the expected risk
- Problem: the expected risk is only available through samples (Generalization issue)

Unsupervised Learning



- Data: Samples
- Task: Find structure or good latent representation in samples
- Learning objective: gather or embed data that share common objective (similarity, pretext task)
- Problem: what is a good objective?

Reinforcement Learning



- Data: samples + rewards
- Task: Learn how to behave s.t. reward is maximized
- Learning objective: generate **sequences** of good decisions
- Problem: value of decisions is only known by acting

⇝ need to generate and analyze a large space of possible sequences (exploration vs. exploitation dilemma).

Vocabulary of RL

Action space
Observation space
State space
Reward & return
Policy
Episode, Trajectory
Horizon
Value function
Off policy / In policy
...

RL basics

1. Modeling sequential dependencies, actions and observations
   - The Markov family
2. Optimizing actions: model based
   - Dynamic programming
   - Policy iteration vs. Value iteration
3. Optimizing actions: Reinforcement Learning
   - Exploration/Exploitation
   - Fundamental algorithms: SARSA and Q-learning
4. What about Deep learning?
   - Function approximation
   - Deep Q-learning: DQN
   - PPO: Proximal Policy Optimization

Modeling sequential dependencies, actions and observations

Markov Process or Markov Chain

- Information state: sufficient statistic of history
- State s_t has Markov property if and only if:

$$p(s_{t+1} \mid s_t) = p(s_{t+1} \mid h_t)$$

where $h_t = (s_0, s_1, \dots s_t)$ is the history and p is a probability.

$\rightsquigarrow$ Memory-less random process (/walk)

- Definition of Markov Process $M = (S, P)$
  - S is a (finite) set of states ($s \in S$)
  - P is a (stationary) transition model that specifies $p(s_{t+1} = s' \mid s_t = s)$
- If finite number (N) of states, P is as a matrix $P_{i,j} = p(s_{t+1} = j \mid s_t = i)$. Many theoretical results!!

Markov Reward Process (MRP)

- Extend Markov Process by rewards
- Definition of Markov Reward Process (MRP) $M = (S, P, R, \gamma)$
  - S is a (finite) set of states ($s \in S$)
  - P is dynamics/transition model that specifies $P(s_{t+1} = s' \mid s_t = s)$
  - R is a reward function $R(s_t = s)$
- Note: still no actions
- If finite number (N) of states, we can express R as a vector

Return & Value Function

- Episode:
  - Sequence of states until termination
  - A “sample”
- Horizon:
  - Number of time steps in each episode
  - Can be finite or infinite

Return & Value Function

- Episode:
  - Sequence of states until termination
  - A "sample"
- Horizon:
  - Number of time steps in each episode
  - Can be finite or infinite
- Return: G_t (for a MRP)
  - Discounted sum of rewards from time step t to horizon with factor $\gamma \in [0, 1]$

$$G_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \gamma^3 r_{t+3} + \dots$$

Return & Value Function

- Episode:
  - Sequence of states until termination
  - A "sample"
- Horizon:
  - Number of time steps in each episode
  - Can be finite or infinite
- Return: G_t (for a MRP)
  - Discounted sum of rewards from time step t to horizon with factor $\gamma \in [0, 1]$

$$G_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \gamma^3 r_{t+3} + \dots$$

- State Value Function: $V(s)$ (for a MRP)
  - Expected return from starting in state s

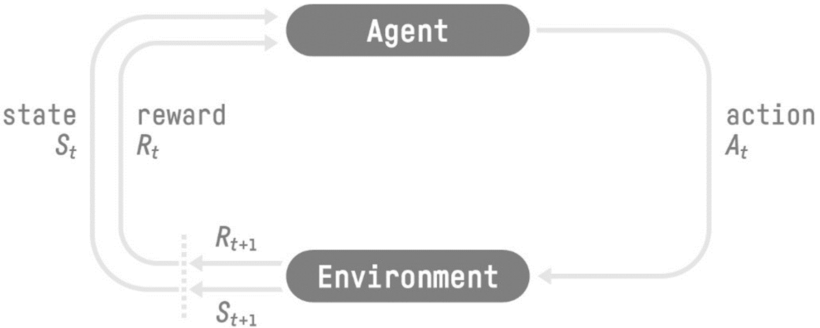
$$V(s) = \mathbb{E}[G_t \mid s_t = s] = \mathbb{E}[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \gamma^3 r_{t+3} + \dots \mid s_t = s]$$

Discount Factor

$$V(s) = \mathbb{E}[G_t \mid s_t = s] = \mathbb{E}[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \gamma^3 r_{t+3} + \dots \mid s_t = s]$$

- Mathematically convenient (avoid infinite returns and values)
- Humans often act as if there's a discount factor $\gamma < 1$ on impact to future
- $\gamma = 0$: Only care about immediate reward
- $\gamma = 1$: Future reward is as beneficial as immediate reward
- If episode lengths are always finite, can use $\gamma = 1$ (but don't have to)

The Reinforcement loop



Markov Decision Process (MDP)

- Markov Decision Process is Markov Reward Process + actions
- Definition of MDP
  - S is a (finite) set of Markov states $s \in S$
  - A is a (finite) set of actions $a \in A$
  - P is dynamics/transition model for each action, that specifies $P(s_{t+1} = s' \mid s_t = s, a_t = a)$
  - R is a reward function $R(s_t = s, a_t = a)$ possibly random
    - Sometimes R is also defined based on (s) or on (s, a, s')
  - Discount factor $\gamma \in [0, 1]$
- MDP is tuple (S, A, P, R, γ)

MDP (cont'd)

- MDP is tuple (S, A, P, R, γ)
- Optional components:
  - $\rho_0 : S \rightarrow \mathbb{R}^+$: a distribution of start states
    - uniform distribution: the agent can start in any state – implicit assumption of MDP definition above
    - non-uniform distribution: the agent starts its episodes in only some of the states; e.g., it's unlikely that a game will start in a terminal state

MDP (cont'd)

- MDP is tuple (S, A, P, R, γ)
- Optional components:
  - $\rho_0 : S \to \mathbb{R}^+$: a distribution of start states
    - uniform distribution: the agent can start in any state – implicit assumption of MDP definition above
    - non-uniform distribution: the agent starts its episodes in only some of the states; e.g., it's unlikely that a game will start in a terminal state
  - $T \subset S$: set of terminal states
    - important for episodic MDPs
    - or if there is not fixed horizon, but the episodes should be finite

MDP (cont'd)

- MDP is tuple (S, A, P, R, γ)
- Optional components:
  - $\rho_0 : S \to \mathbb{R}^+$: a distribution of start states
    - uniform distribution: the agent can start in any state – implicit assumption of MDP definition above
    - non-uniform distribution: the agent starts its episodes in only some of the states; e.g., it's unlikely that a game will start in a terminal state
  - $T \subset S$: set of terminal states
    - important for episodic MDPs
    - or if there is not fixed horizon, but the episodes should be finite
  - γ: discount factor
    - important to quantify the importance of future
    - some treat γ as a hyperparameter and not part of the definition

    $\rightsquigarrow$ different optimal policies can be found

    $\rightsquigarrow$ depends on how the optimal policy is defined

MDP Policy

- Policy specifies what action to take in each state
  - Can be deterministic or stochastic
- For generality, considered as a stationary conditional distribution

$$\pi(a \mid s) = P(a_t = a | s_t = s)$$

- Other name for a control law.
- This is what is expected to be learned!

MDP + Policy = MRP

- MDP + Policy $\pi(a \mid s)$ = Markov Reward Process
- $[s_t, a_{t-1}]$ is Markov
- Precisely, it is the MRP $(S, R^\pi, P^\pi, \gamma)$ where

$$R^\pi(s) = \sum_{a \in A} \pi(a \mid s) R(s, a)$$

$$P^\pi(s' \mid s) = \sum_{a \in A} \pi(a \mid s) P(s' \mid s, a)$$

- Implies we can use same techniques to evaluate the value of a policy for an MDP as we could to compute the value of a MRP, by defining a MRP with R^π and P^π

MDP and Value functions

- Definition of Return G_t (same as MRP)
  - Discounted sum of rewards from time step t to horizon

$$G_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \gamma^3 r_{t+3} + \dots$$

MDP and Value functions

- Definition of Return G_t (same as MRP)
  - Discounted sum of rewards from time step t to horizon

$$G_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \gamma^3 r_{t+3} + \dots$$

- Definition of *On-Policy* State Value Function $V^\pi(s)$: depends on policy π!
  - Expected return from starting in state s under policy π

$$\begin{aligned} V^\pi(s) &= \mathbb{E}_\pi[G_t \mid s_t = s] && (1) \\ &= \mathbb{E}_\pi[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \gamma^3 r_{t+3} + \dots \mid s_t = s] && (2) \end{aligned}$$

MDP and Value functions

- Definition of Return G_t (same as MRP)
  - Discounted sum of rewards from time step t to horizon

$$G_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \gamma^3 r_{t+3} + \dots$$

- Definition of *On-Policy* State Value Function $V^\pi(s)$: depends on policy π!
  - Expected return from starting in state s under policy π

$$\begin{aligned} V^\pi(s) &= \mathbb{E}_\pi[G_t \mid s_t = s] && (1) \\ &= \mathbb{E}_\pi[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \gamma^3 r_{t+3} + \dots \mid s_t = s] && (2) \end{aligned}$$

- Definition of *On-Policy* State-Action Value Function $Q^\pi(s, a)$
  - Expected return from starting in state s, taking action a and then following policy π

$$\begin{aligned} Q^\pi(s, a) &= \mathbb{E}_\pi[G_t \mid s_t = s, a_t = a] \\ &= \mathbb{E}_\pi[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \gamma^3 r_{t+3} + \dots \mid s_t = s, a_t = a] \end{aligned}$$

Other Markov models

- In MRP and MDP, the state (i.e. the environment) is fully **observable**. In many problems, this is not possible.
- HMM: Hidden Markov Model
  - The state s is partially observable from o with conditional probability: $p(o \mid s)$
- POMDP: Partially Observable Markov Decision Process
  - MDP + HMM

Optimizing actions: model based approach

Optimal value functions

- The Optimal Value Function, $V^*(s)$ gives the expected return if you start in state s and always act according to the optimal policy in the environment:

$$V^*(s) = \max_\pi \mathbb{E}_\pi[G_t \mid s_t = s]$$

- The Optimal Action-Value Function, $Q^*(s,a)$ gives the expected return if you start in state s, take an arbitrary action a, and then forever after act according to the optimal policy in the environment:

$$Q^*(s,a) = \max_\pi \mathbb{E}_\pi[G_t \mid s_t = s, a_t = a]$$

- The goal is to compute the optimal policy π^* that maximizes the expected return:

$$\pi^*(s) \in \arg\max_\pi V^\pi(s)$$

MDP optimality

It can be shown that:

- There exists a unique optimal value function $V^*(s)$
- In an infinite horizon problem (i.e. agents act forever), there exists an optimal policy π^* for an MDP, that is
  - deterministic
  - stationary (does not depend on time step)
  - unique? $\rightsquigarrow$ Not necessarily, may have state-actions with identical optimal values.
  - The optimal policy can be expressed as:

$$\pi^*(s) \in \arg\max_a Q^*(s, a)$$

- In an finite horizon problem undiscounted return ($\gamma = 1$) needs to make time as an argument $\rightsquigarrow$ trade-off between time and impact of actions

Bellman Equation and Bellman Backup Operators

The fundamental equations!

- Value functions of a policy must satisfy the Bellman equations

$$V^\pi(s) = \sum_a \pi(a \mid s) \left[R(s,a) + \gamma \sum_{s' \in S} P(s' \mid s,a) V^\pi(s') \right]$$

$$Q^\pi(s,a) = R(s,a) + \gamma \sum_{s' \in S} P(s' \mid s,a) \sum_{a'} \pi(a' \mid s') Q^\pi(s',a')$$

- "*The value of your starting point is the reward you expect to get from being there, plus the value of wherever you land next.*"
- Optimal equations:

$$V^*(s) = \max_a \left[R(s,a) + \gamma \sum_{s' \in S} P^\pi(s' \mid s,a) V^*(s') \right]$$

$$Q^*(s,a) = R(s,a) + \gamma \sum_{s' \in S} P(s' \mid s,a) \max_{a'} Q^*(s',a')$$

Bellman Backup Operators

Bellman backup operator B^π

$$B^\pi V(s) = \sum_a \pi(a \mid s) \left[R(s,a) + \gamma \sum_{s' \in S} P(s' \mid s,a) V(s') \right]$$

Bellman optimal backup operator B^*

$$B^* V(s) = \max_a \left[R(s,a) + \gamma \sum_{s' \in S} p(s' \mid s,a) V(s') \right]$$

- Read B^* and B^π as operators applied to V and yielding a value function over all states s
- Need to know transition probability $p(s' \mid s, a)$ and rewards function $R(s,a)$ to compute it.

MDP: Computing Optimal Policy and Optimal Value I

Two strategies

1. Learn the Policy by iteration
   - Start with an approximate or random policy
   - Evaluate it
   - Improve it
2. Learn the Value function by iteration
   - Idea: Maintain optimal value of starting in a state s if we have a finite number of steps k left in the episode
   - Iterate to consider longer and longer episodes
   - Define the policy from the Value function V
3. Estimate the value by random sampling (Monte-Carlo)
   - Idea (simple): generate episodes, collect rewards and returns, compute average return.

MDP Policy Evaluation, Iterative Algorithm

- Goal: For a given π, determine V^π
- iterative approach:
  - Initialize $V_0(s) = 0$ for all s
  - For $k = 1$ until convergence
    - For all s in S:

$$V_k^\pi(s) = r(s, \pi(s)) + \gamma \sum_{s' \in S} p(s' \mid s, \pi(s)) V_{k-1}^\pi(s')$$

- This is a Bellman backup for a particular policy

MDP Policy Evaluation

Algorithm 1: Policy Evaluation

Input: MDP, policy π, small positive number θ

Output: $V \approx v_\pi$

Initialize V arbitrarily (e.g., $V(s) = 0$ for all $s \in \mathcal{S}^+$)

repeat

  $\Delta \leftarrow 0$

  for $s \in \mathcal{S}$ **do**

    $v \leftarrow V(s)$

    $V(s) \leftarrow \sum_a \pi(a \mid s) \sum_{s'} p(s' \mid s, a)(r + \gamma V(s'))$

    $\Delta \leftarrow \max(\Delta, |v - V(s)|)$

  end

until $\Delta < \theta$;

return V

Policy Evaluation as Bellman Operations

- Bellman backup operator B^π for a particular policy is defined as

$$B^\pi V(s) = R^\pi(s) + \gamma \sum_{s' \in S} P^\pi(s' \mid s) V(s')$$

- Policy evaluation amounts to computing the fixed point of B^π
- To do policy evaluation, repeatedly apply operator until V stops changing

$$V^\pi = B^\pi B^\pi B^\pi B^\pi \dots B^\pi V$$

- It converges because B^π is contractive for $\gamma < 1$.

Policy Improvement

- Act greedily with respect to V^{π} to choose the action.
- Compute state-action value of a policy π_k
  - For s in S and a in A:

$$Q^{\pi}(s,a) = R(s,a) + \gamma \sum_{s' \in S} P(s' \mid s,a) V^{\pi}(s')$$

- Compute new policy π_{k+1} for all $s \in S$

$$\pi_{k+1}(s) \in \arg\max_{a \in A} Q^{\pi_k}(s,a)$$

Policy Improvement

Algorithm 2: Policy Improvement

Input: MDP, value function V

Output: policy π'

for $s \in S$ **do**

  for $a \in A$ **do**

    $Q(s,a) \leftarrow \sum_{s'} p(s' \mid s,a)(r + \gamma V(s'))$

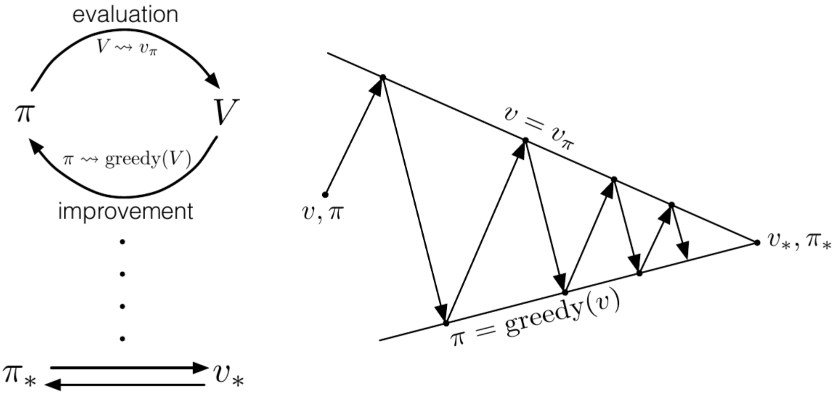
  end

  $\pi'(s) \leftarrow \arg\max_{a \in \mathcal{A}(s)} Q(s,a)$

end

return π'

MDP Policy Iteration



MDP Policy Iteration

Algorithm 3: Policy Iteration

Input: MDP, small positive number θ

Output: policy $\pi \approx \pi^*$

Initialize π arbitrarily (e.g., $\pi(a \mid s) = \frac{1}{|A|}$ for all $s \in S$ and $a \in A$)

policy-stable $\leftarrow$ *false*

repeat

  $V \leftarrow$ **Policy_Evaluation**(MDP, π, θ)

  $\pi' \leftarrow$ **Policy_Improvement**(MDP, V)

  if $\pi = \pi'$ **then**

    policy-stable $\leftarrow$ *true*

  end

  $\pi \leftarrow \pi'$

until *policy-stable* $=$ *true*;

return π

Policy iteration stopping condition

- Since $\pi_{k+1}(s) \in \arg\max_{a \in A} Q^{\pi_k}(s,a)$ we can ensure that

$$\forall s \in S, V^{\pi_{k+1}}(s) \geq V^{\pi_k}(s)$$

- Iterations stop for a k where

$$Q^{\pi}(s, \pi_{k+1}(s)) = \max_a Q^{\pi}(s,a) = Q^{\pi}(s, \pi_k(s)) = V^{\pi_k}(s)$$

- The iterations stop when Bellman optimality condition is verified

$$\forall s \in S, V^{\pi}(s) = \max_a Q^{\pi}(s,a)$$

- The output policy π_k is optimal ($\pi_k = \pi^*$)

Value Iteration (VI)

- Set $k = 1$
- Initialize $V_0(s) = 0$ for all states s
- Loop until convergence
  - For each state s

$$V_{k+1}(s) = \max_{a \in A} R(s,a) + \gamma \sum_{s' \in S} P(s' \mid s, a) V_k(s')$$

  - Interpreted as Bellman backup on value function (fixed point iteration)

$$V_{k+1} = B^{\pi} V_k$$

- The final policy is defined as:

$$\pi(s) \in \arg\max_{a \in A} \left[R(s,a) + \gamma \sum_{s' \in S} P(s' \mid s, a) V(s') \right]$$

Value Iration

Algorithm 4: Value Iteration

Input: MDP, small positive number θ

Output: policy $\pi \approx \pi_*$

Initialize V arbitrarily (e.g., $V(s) = 0$ for all $s \in \mathcal{S}^+$)

repeat

  $\Delta \leftarrow 0$

  for $s \in \mathcal{S}$ **do**

    $v \leftarrow V(s)$

    $V(s) \leftarrow \max_{a \in A} \sum_{s'} p(s' \mid s, a)(r + \gamma V(s'))$

    $\Delta \leftarrow \max(\Delta, |v - V(s)|)$

  end

until $\Delta < \theta$;

$\pi \leftarrow$ **Policy_Improvement**(MDP, V)

return π

Monte-Carlo RL

Evaluate the Policy by random sampling + iteration (Monte-Carlo)

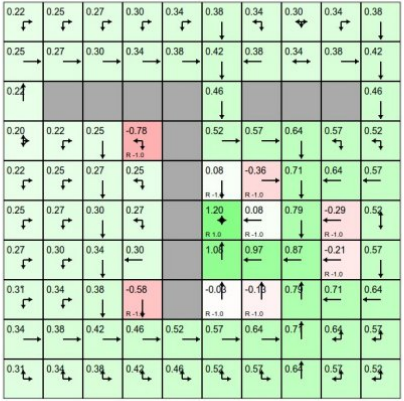
- Generate a series of episodes $[s_0, a_1, s_1, r_1, \ldots s_T, a_T, r_T]$ using current policy π_k, and estimate the partial return $G_t = \sum_{j=t}^{T} \gamma^{j-t} r_j$
- Value = mean return on each state (Law of Large Numbers) for first-visit or every-visit state
- Improve policy

Monte-Carlo policy evaluation

Algorithm 5: First-Visit MC Prediction (*for action values*)

Input: policy π, positive integer `num_episodes`
Output: value function Q ($\approx q_\pi$ if `num_episodes` is large enough)
Initialize $N(s, a) = 0$ for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$
Initialize $\texttt{returns\_sum}(s, a) = 0$ for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$
for $i \leftarrow 1$ ***to*** `num_episodes` **do**
    Generate an episode $S_0, A_0, R_1, \ldots, S_T$ using π
    for $t \leftarrow 0$ ***to*** $T - 1$ **do**
        if (S_t, A_t) *is a first visit (with return G_t)* **then**
            $N(S_t, A_t) \leftarrow N(S_t, A_t) + 1$
            $\texttt{returns\_sum}(S_t, A_t) \leftarrow \texttt{returns\_sum}(S_t, A_t) + G_t$
    end
end
$Q(s, a) \leftarrow \texttt{returns\_sum}(s, a)/N(s, a)$ for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$
return Q

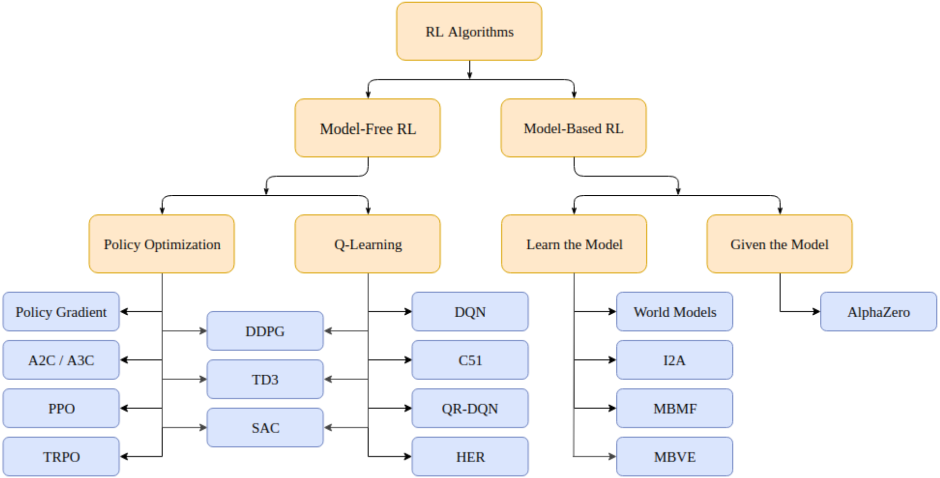
Small demo



Grid World

Learning policy

The huge family of RL algorithms



Learning without models

- Assumption: We don't have the exact model of the environment (i.e., model-free), but we can query the environment ("playing roll-outs")
  - state space and action space are in principle known
  - we don't know the transition probabilities beforehand
  - we don't know the reward distribution beforehand

Learning without models

- Assumption: We don't have the exact model of the environment (i.e., model-free), but we can query the environment ("playing roll-outs")
  - state space and action space are in principle known
  - we don't know the transition probabilities beforehand
  - we don't know the reward distribution beforehand

- Remarks:
  - If we would know the MDP, we only have to do "planning" to find the optimal policy
  - If we first learn the MDP and then apply planning to the learned MDP, we do "model-based" RL

Learning without models

- Assumption: We don't have the exact model of the environment (i.e., model-free), but we can query the environment ("playing roll-outs")
  - state space and action space are in principle known
  - we don't know the transition probabilities beforehand
  - we don't know the reward distribution beforehand

- Remarks:
  - If we would know the MDP, we only have to do "planning" to find the optimal policy
  - If we first learn the MDP and then apply planning to the learned MDP, we do "model-based" RL

- Goal: We want to learn $V^{\pi}(s)$ or $Q^{\pi}(s, a)$ (depending on the RL algorithm we want to use) by only querying the unknown MDP

Exploration & exploitation

When the model + environment are unknown, we need to find ways to understand how they behave.

Reinforcement learning is like trial-and-error learning

- The agent should discover a good policy
- From its experiences of the environment
- Without losing too much reward along the way

Two design principles:

- exploration = generate relevant experience of unknown environment
- exploitation = use experience to efficiently improve the policy π (= maximize the reward)

RL first basic ideas

Greedy Policy iteration may be inefficient:

- If π is deterministic, we may not observe all possible actions $a \in A$ in a state s
- So, we cannot compute $Q(s, a)$ for any $a \neq \pi(s)$

$\rightsquigarrow$ How to interleave policy evaluation and improvement?

Two tools

- Randomness in decision: ϵ-greediness
- Sequential stochastic approximation

ϵ-greedy Policies

Simplest idea for ensuring continual exploration

- All m actions are tried with non-zero probability
- With probability $1 - \epsilon$ choose the greedy action
- With probability ϵ choose an action at random

$$\pi(a \mid s) = \begin{cases} 1 - \epsilon + \epsilon/m & \text{if } a = \arg\max_{a' \in A} Q(s, a') \\ \epsilon/m & \text{otherwise} \end{cases}$$

One can show that

- For any ϵ-greedy policy π', the ϵ-greedy policy wrt Q^π is a monotonic improvement of the value function

$$V^{\pi'} \geq V^\pi$$

Greedy in the Limit of Infinite Exploration (GLIE)

Definition of GLIE:

- All state-action pairs are visited an infinite number of times
- Behavior policy (policy used to act in the world) converges to greedy policy

$$\lim_{i \to \infty} \pi(a \mid s) \to \arg\max_{a \in A} Q(s, a)$$

with probability 1

Simple Strategy:

- ϵ-greedy where ϵ is annealed close to 0 with $\epsilon_i = 1/i$

Theorem:

- GLIE Monte-Carlo control converges to the optimal state-action value function $Q(s, a) \to Q^*(s, a)$

Monte-Carlo policy optimization

Algorithm 6: First-Visit GLIE MC Control

Input: positive integer `num_episodes`, GLIE $\{\epsilon_i\}$
Output: policy π ($\approx \pi_*$ if `num_episodes` is large enough)

Initialize $Q(s, a) = 0$ for all $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$
Initialize $N(s, a) = 0$ for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$
for $i \leftarrow 1$ ***to*** `num_episodes` **do**
  $\epsilon \leftarrow \epsilon_i$
  $\pi \leftarrow \epsilon$-greedy(Q)
  Generate an episode $S_0, A_0, R_1, \ldots, S_T$ using π
  for $t \leftarrow 0$ ***to*** $T - 1$ **do**
    if (S_t, A_t) *is a first visit (with return* G_t*)* **then**
      $N(S_t, A_t) \leftarrow N(S_t, A_t) + 1$
      $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{1}{N(S_t, A_t)}(G_t - Q(S_t, A_t))$
  end
end
return π

Model-free Policy Iteration with TD Methods

- Monte-Carlo approach requires to sample full episodes before updating the policy using the return G_t
- One idea: use the current estimate of the return $r_{t+1} + \gamma Q(s_{t+1}, a_{t+1})$ instead of G_t and update at each time step
  $=$ this is *Temporal Difference* learning
- It allows the exploitation of incomplete sequences

This gives the *SARSA* algorithm.

Algorithm 7: Sarsa

Input: policy π, positive integer `num_episodes`, small positive fraction α, GLIE $\{\epsilon_i\}$

Output: value function Q ($\approx q_\pi$ if `num_episodes` is large enough)

Initialize Q arbitrarily (e.g., $Q(s,a) = 0$ for all $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$, and $Q(\textit{terminal-state}, \cdot) = 0$)

for $i \leftarrow 1$ ***to*** `num_episodes` **do**

    $\epsilon \leftarrow \epsilon_i$

    Observe S_0

    Choose action A_0 using policy derived from Q (e.g., ϵ-greedy)

    $t \leftarrow 0$

    repeat

        Take action A_t and observe R_{t+1}, S_{t+1}

        Choose action A_{t+1} using policy derived from Q (e.g., ϵ-greedy)

        $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t))$

        $t \leftarrow t + 1$

    until S_t *is terminal*;

end

return Q

Convergence Properties of SARSA

- Theorem:
  SARSA for finite-state and finite-action MDPs converges to the optimal action-value, $Q(s, a) \to Q^*(s, a)$, under the following conditions:
  1. The policy sequence $\pi_t(a \mid s)$ satisfies the condition of GLIE
  2. The step-sizes α_t satisfy the Robbins-Munro sequence such that

$$\sum_{t=1}^{\infty} \alpha_t = \infty$$

$$\sum_{t=1}^{\infty} \alpha_t^2 < \infty$$

- For example $\alpha_t = \frac{1}{t}$ satisfies the above condition
- In practice, α is often kept constant.

Q-Learning: Learning the Optimal State-Action Value

- A possible alternative is to directly estimate the value of π^* while acting with another behavior policy π_b?
- This is called an *off-policy* RL algorithm
- Key idea: Maintain state-action Q estimates and use it to bootstrap: use the value of the best future action
- This gives the *Q-Learning* updating step:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha((r_{t+1} + \gamma \max_{a' \in A} Q(s_{t+1}, a')) - Q(s_t, a_t))$$

- Same convergence property to optimal policy as SARSA (GLIE + Robbins-Munro)

Algorithm 8: Q-Learning (Sarsamax)

Input: policy π, positive integer `num_episodes`, small positive fraction α, GLIE $\{\epsilon_i\}$

Output: value function Q ($\approx q_\pi$ if `num_episodes` is large enough)

Initialize Q arbitrarily (e.g., $Q(s,a) = 0$ for all $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$, and $Q(\textit{terminal-state}, \cdot) = 0$)

for $i \leftarrow 1$ ***to*** `num_episodes` **do**

  $\epsilon \leftarrow \epsilon_i$

  Observe S_0

  $t \leftarrow 0$

  repeat

    Choose action A_t using policy derived from Q (e.g., ϵ-greedy)

    Take action A_t and observe R_{t+1}, S_{t+1}

    $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t))$

    $t \leftarrow t + 1$

  until S_t *is terminal*;

end

return Q

Policies and experience

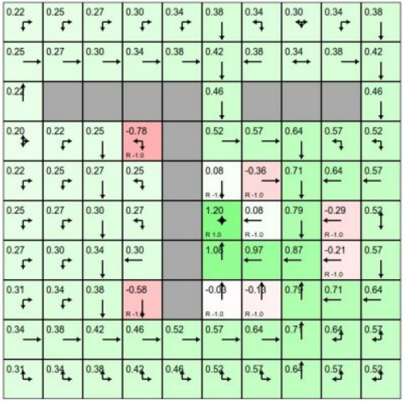
- On-policy learning
  - "Learn on the job"
  - Learn about policy π from experience sampled from π
  - SARSA is one example
- Off-policy learning
  - "Look over someone's shoulder"
  - Learn about policy π from experience sampled from another one (μ)
  - Q-learning is one example

RL summary

Full Backup (DP)	*Sample Backup (TD)*
Iterative Policy Evaluation	TD Learning
$V(s) \leftarrow \mathbb{E}\left[R + \gamma V(S') \mid s\right]$	$V(S) \overset{\alpha}{\leftarrow} R + \gamma V(S')$
Q-Policy Iteration	Sarsa
$Q(s,a) \leftarrow \mathbb{E}\left[R + \gamma Q(S',A') \mid s,a\right]$	$Q(S,A) \overset{\alpha}{\leftarrow} R + \gamma Q(S',A')$
Q-Value Iteration	Q-Learning
$Q(s,a) \leftarrow \mathbb{E}\left[R + \gamma \max_{a' \in \mathcal{A}} Q(S',a') \mid s,a\right]$	$Q(S,A) \overset{\alpha}{\leftarrow} R + \gamma \max_{a' \in \mathcal{A}} Q(S',a')$

(from D. Silver slides)

Small demo - RL



Grid World

Deep RL

Deep Learning in RL

- High dimensional (continuous) states: e.g. images, kinematics of articulated structures
- Parametric representation of the policy

$$\hat{Q}(s, a; \mathrm{w}) \approx Q(s, a)$$

- Function Approximations techniques (gradient descent on relevant MSE loss) and generalization issues

Incremental Model-Free Control Approaches

- Similar to policy evaluation, true state-action value function for a state is unknown and we substitute an estimate
- In Monte Carlo methods, use a return G_t as a substitute target

$$\Delta \mathrm{w} = \alpha(G_t - \hat{Q}(s_t, a_t; \mathrm{w}))\nabla_{\mathrm{w}} \hat{Q}(s_t, a_t; \mathrm{w})$$

- For SARSA instead use a TD target $r + \gamma \hat{Q}(s', a'; \mathrm{w})$ which leverages the current function approximations value

$$\Delta \mathrm{w} = \alpha(r + \gamma \hat{Q}(s', a'; \mathrm{w}) - \hat{Q}(s, a; \mathrm{w}))\nabla_{\mathrm{w}} \hat{Q}(s, a; \mathrm{w})$$

- For Q-learning instead use a TD target $r + \gamma \max_{a'} \hat{Q}(s', a'; \mathrm{w})$ which leverages the max of the current function approximations value

$$\Delta \mathrm{w} = \alpha(r + \gamma \max_{a'} \hat{Q}(s', a'; \mathrm{w}) - \hat{Q}(s, a; \mathrm{w}))\nabla_{\mathrm{w}} \hat{Q}(s, a; \mathrm{w})$$

Using these Ideas to do Deep RL for the Atari challenge

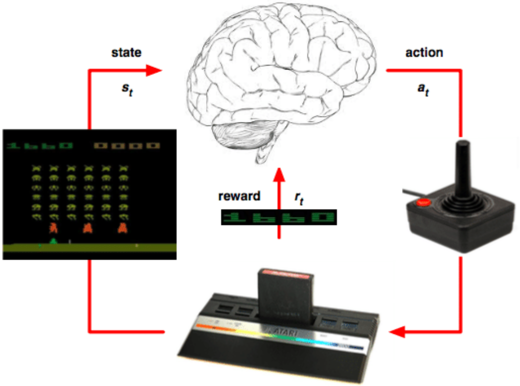
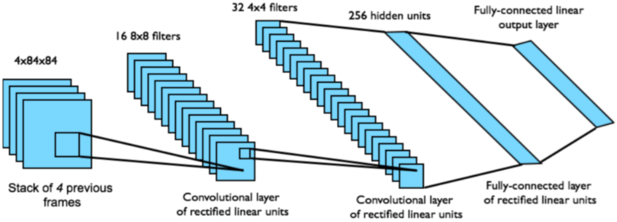


Image by David Silver

Using these Ideas to do Deep RL in Atari

- End-to-end learning of values $Q(s, a)$ from pixels s
- Input state s is stack of raw pixels from last 4 frames
- Output is $Q(s, a)$ for 18 joystick/button positions
- Reward is change in score for that step
- Network architecture and hyperparameters fixed across all games



DQN source code: https://github.com/deepmind/dqn

Q-Learning with Value Function Approximation

- Minimize MSE loss by stochastic gradient descent
- Converges to the optimal $Q^*(s, a)$ using table lookup representation
- However Q-learning with Value Function Approximation is unstable
- Two of the issues causing problems:
  - Correlations between samples violates i.i.d assumption of DNNs
  - Non-stationary targets
- Deep Q-learning (DQN) addresses both of these challenges by
  - Experience replay
  - Fixed Q-targets

DQNs: Replay Buffer

- To help remove correlations, store dataset (called a replay buffer) $\mathcal{D}$ from prior experience
- To perform experience replay, repeat the following:
  1. $(s, a, r, s') \sim \mathcal{D}$: sample experience tuple from the dataset
  2. Compute the target value for the sampled s: $r + \gamma \max_{a'} \hat{Q}(s', a'; \mathrm{w})$
  3. Use stochastic gradient descent to update the network weights

$$\Delta \mathrm{w} = \alpha(r + \gamma \max_{a'} \hat{Q}(s', a'; \mathrm{w}) - \hat{Q}(s, a; \mathrm{w})) \nabla_{\mathrm{w}} \hat{Q}(s, a; \mathrm{w})$$

- Remarks:
  - Fixed sized buffer $\rightsquigarrow$ first-in–first-out scheme (as default implementation)
  - heuristic trade-off between performing new episodes and sampling from the replay buffer

$\rightsquigarrow$ Can treat the target as a scalar, but the weights will get updated on the next round, changing the target value

DQNs: Fixed Q-Targets

- To help improve stability, fix the target weights used in the target calculation for multiple updates
- Target network uses a different set of weights than the weights being updated
- Let parameters w^- be the set of weights used in the target and w be the weights that are being updated
- Slight change to computation of target value:
  - $(s, a, r, s') \sim \mathcal{D}$: sample experience tuple from the dataset
  - Compute the target value for the sampled s: $r + \gamma \max_{a'} \hat{Q}(s', a'; \mathrm{w}^-)$
  - Use stochastic gradient descent to update the network weights

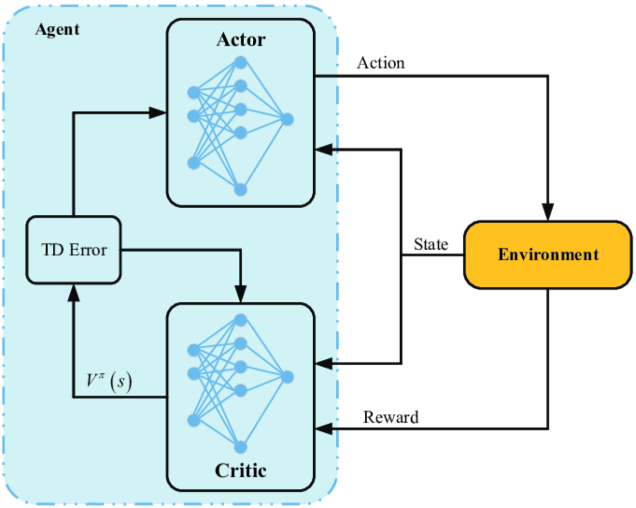
$$\Delta \mathrm{w} = \alpha(r + \gamma \max_{a'} \hat{Q}(s', a'; \mathrm{w}^-) - \hat{Q}(s, a; \mathrm{w}))\nabla_{\mathrm{w}} \hat{Q}(s, a; \mathrm{w})$$

- Remark:
  - Hyperparameter how often you update w^-
  - Trade-off between updating too often ($\rightsquigarrow$ instability) and too rarely ($\rightsquigarrow$ too old state information)

DQN Summary

- DQN uses experience replay and fixed Q-targets
- Store transition $(s_t, a_t, r_{t+1}, s_{t+1})$ in replay memory $\mathcal{D}$
- Sample random mini-batch of transitions (s, a, r, s') from $\mathcal{D}$
- Compute Q-learning targets wrt old, fixed parameters w^-
  - Update w^- from time to time
- Optimizes MSE between Q-network and Q-learning targets
- Uses stochastic gradient descent

Actor-Critic algorithms I



Actor-Critic algorithms II

- Goal: better balance between exploration and exploitation.
- **Actor (Policy)**: Maintains a policy $\pi(a|s;\theta)$.
- **Critic (Value Function)**: Learns a value function $V(s;\phi)$. Estimates the expected cumulative reward.
- **Interaction and Learning**:
  1. The Actor interacts with the environment.
  2. The Critic updates its value function.
  3. The Actor updates its policy.

PPO: a popular actor-critic strategy

- **Proximal Policy Optimization**.
- **On-Policy**: Learns the policy by interacting with the environment using that same policy.
- **Trust Region**: Improves the policy while staying *close* to the previous policy. Essential for stability.
- **Simple Implementation**: Relatively easy to implement and tune.

PPO Objective Function I

$$L(\theta) = \mathbb{E}_t \left[\min\left(r_t(\theta)A_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)A_t\right)\right]$$

- $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$ **(Probability Ratio):** Measures how much the probability of taking action a_t in state s_t has changed between the current policy π_θ and the old policy $\pi_{\theta_{old}}$. A value close to 1 indicates a small change.
- A_t **(Advantage Function):** Quantifies how much better taking action a_t in state s_t was compared to the average return expected in that state. It helps the algorithm learn which actions are truly beneficial. A common way to calculate it is: $A_t = R_t + \gamma V(s_{t+1}) - V(s_t)$, where R_t is the reward received at time t, γ is the discount factor, and $V(s)$ is the value function.

PPO Objective Function II

$$L(\theta) = \mathbb{E}_t \left[\min\left(r_t(\theta)A_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)A_t\right)\right]$$

- ϵ **(Clipping Parameter):** A hyperparameter that defines the size of the trust region. It limits how much the policy can change in a single update. Typical values are around 0.1 or 0.2. It prevents the policy from making overly large updates that could destabilize training.
- $\textbf{clip}(x, 1-\epsilon, 1+\epsilon)$ **(Clipping Function):** Limits the probability ratio $r_t(\theta)$ to the range $[1-\epsilon, 1+\epsilon]$. This is the key mechanism for enforcing the trust region.
- $\min(\cdot, \cdot)$ **(Minimum Function):** Takes the minimum of the two arguments. This ensures that the policy update is both beneficial (positive advantage) and within the trust region.

How the Critic Learns

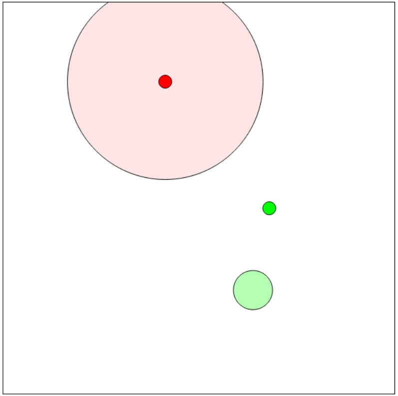
- **Goal**: Accurately estimate the value function $V(s)$ (expected cumulative reward from state s).
- **Method**: Typically Temporal Difference (TD) learning (e.g., TD(0)) combined with function approximation (often a neural network).
- **Process (Simplified)**:
  1. Collect experiences (s, a, r, s') from the Actor.
  2. Calculate a target value (e.g., using TD(0): $r + \gamma V(s')$).
  3. Update the Critic's parameters (e.g., neural network weights) to minimize the difference between the current value estimate $V(s)$ and the target.
- **Advantages**:
  - **Stability**: The clipping mechanism makes PPO more stable.
  - **Sample Efficiency**: Learns good policies with fewer interactions.
  - **Ease of Implementation**: Relatively simple to implement.

Other approaches

MANY!

(See references)

Small demo - Deep RL



Puck World

Références

Many of the slides come from these references

- https://www.davidsilver.uk/teaching/
- https://web.stanford.edu/class/cs234/modules.html
- https://github.com/automl-edu/RL_lecture