Reinforcement Learning: Basic concepts

Joelle Pineau
School of Computer Science, McGill University
Facebook AI Research (FAIR)

CIFAR Reinforcement Learning Summer School
July 3 2017
Reinforcement learning

• Learning by trial-and-error, in real-time.

• Improves with experience

• Inspired by psychology
  – Agent + Environment
  – Agent selects actions to maximize utility function.
RL system circa 1990’s: TD-Gammon

Reward function:

+100 if win
- 100 if lose
0 for all other states

Trained by playing $1.5 \times 10^6$ million games against itself.

Enough to beat the best human player.
2016: World Go Champion Beaten by Deep Learning
RL applications at RLDM 2017

- Robotics
- Video games
- Conversational systems
- Medical intervention
- Algorithm improvement
- Improvisational theatre
- Autonomous driving
- Prosthetic arm control
- Financial trading
- Query completion
When to use RL?

- Data in the form of trajectories.
- Need to make a sequence of (related) decisions.
- Observe (partial, noisy) feedback to choice of actions.
- Tasks that require both learning and planning.
RL vs supervised learning

Training signal = desired (target outputs), e.g. class

Supervised Learning

Inputs ➔ Supervised Learning ➔ Outputs

Training signal = “rewards”

Reinforcement Learning

Inputs ("states") ➔ Reinforcement Learning ➔ Outputs ("actions")
RL vs supervised learning

Training signal = desired (target outputs), e.g. class

Inputs → Supervised Learning → Outputs

Training signal = “rewards”

Inputs ("states") → Reinforcement Learning → Outputs ("actions")

Environment
Supervised Learning

Inputs → Outputs

Inputs: (states)

Training signal = desired (target outputs), e.g. class

Reinforcement Learning

Inputs: (“states”)

Training signal = “rewards”

Environment

Outputs: (“actions”)

Practical & technical challenges:

1. Need access to the environment.
2. Jointly learning AND planning from correlated samples.
3. Data distribution changes with action choice.
Markov Decision Process (MDP)

Defined by:

\[ S = \{s_1, s_2, \ldots, s_n\} \], the set of states *(can be infinite/continuous)*

\[ A = \{a_1, a_2, \ldots, a_m\} \], the set of actions *(can be infinite/continuous)*

\[ T(s,a,s') := Pr(s'|s,a) \], the dynamics of the environment

\[ R(s,a) \]: Reward function

\[ \mu(s) \]: Initial state distribution
The **Markov** property

The distribution over future states depends only on the present state and action, not on any other previous event.

\[ Pr(s_{t+1} \mid s_0, \ldots, s_t, a_0, \ldots, a_t) = Pr(s_{t+1} \mid s_t, a_t) \]
The **Markov** property

- Traffic lights?

- Chess?
The Markov property

- Traffic lights?
- Chess?
- Poker?

**Tip**: Incorporate past observations in the state to have sufficient information to predict next state.
The goal of RL? Maximize return!

- Return, $U_t$ of a trajectory, is the sum of rewards starting from step $t$. 
The goal of RL? Maximize return!

• **Return**, $U_t$ of a trajectory, is the sum of rewards starting from step $t$.

$$U_t = r_t + r_{t+1} + r_{t+2} + \ldots + r_T$$

• **Episodic task**: consider return over finite horizon (e.g. games, maze).

• **Continuing task**: consider return over infinite horizon (e.g. juggling, balancing).

$$U_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \gamma^3 r_{t+3} \ldots = \sum_{k=0}^{\infty} \gamma^k r_{t+k}$$
The discount factor, $\gamma$

- **Discount factor**, $\gamma \in [0, 1)$ (usually close to 1).

- **Intuition:**
  - Receiving $80$ today is worth the same as $100$ tomorrow (assuming a discount factor of factor of $\gamma = 0.8$).
  - At each time step, there is a $1-\gamma$ chance that the agent dies, and does not receive rewards afterwards.
Defining behavior: The policy

- **Policy**, \( \pi \) defines the action-selection strategy at every state:

\[
\pi(s,a) = P(a_t=a \mid s_t=s)
\]

\( \pi : S \rightarrow A \)

**Goal:** Find the policy that maximizes expected total reward.

*(But there are many policies!)*

\[
\argmax_{\pi} E_{\pi} \left[ r_0 + r_1 + \ldots + r_T \mid s_0 \right]
\]
Example: Career Options

What is the best policy?

n = Do Nothing
i = Apply to industry
g = Apply to grad school
a = Apply to academia
Example: Career Options

What is the best policy?

$R(s) = -1$ for Unemployed
$R(s) = 0$ for Grad School
$R(s) = +5$ for Academia
$R(s) = +10$ for Industry

n=Do Nothing
i = Apply to industry
g = Apply to grad school
a = Apply to academia
Value functions

The expected return of a policy (for every state) is called the value function: \( V^\pi(s) = E^\pi [r_t + r_{t+1} + \ldots + r_T | s_t = s] \)

Simple strategy to find the best policy:

1. Enumerate the space of all possible policies.
2. Estimate the expected return of each one.
3. Keep the policy that has maximum expected return.
Getting confused with terminology?

- Reward?
- Return?
- Value?
- Utility?
Getting confused with terminology?

- **Reward**: 1 step numerical feedback
- **Return**: Sum of rewards over the agent’s trajectory.
- **Value**: Expected sum of rewards over the agent’s trajectory.
- **Utility**: Numerical function representing preferences.

- In RL, we assume $\text{Utility} = \text{Return}$. 
The value of a policy

\[ V^\pi(s) = E_\pi [ r_t + r_{t+1} + \ldots + r_T | s_t = s ] \]

\[ V^\pi(s) = E_\pi [ r_t ] + E_\pi [ r_{t+1} + \ldots + r_T | s_t = s ] \]

\[ V^\pi(s) = \sum_{a \in A} \pi(s,a) R(s,a) + E_\pi [ r_{t+1} + \ldots + r_T | s_t = s ] \]

Immediate reward  \hspace{1cm} Future expected sum of rewards
The value of a policy

\[ V_\pi(s) = E_\pi [r_t + r_{t+1} + \ldots + r_T \mid s_t = s] \]

\[ V_\pi(s) = E_\pi [r_t] + E_\pi [r_{t+1} + \ldots + r_T \mid s_t = s] \]

\[ V_\pi(s) = \sum_{a \in A} \pi(s,a)R(s,a) + E_\pi [r_{t+1} + \ldots + r_T \mid s_t = s] \]

\[ V_\pi(s) = \sum_{a \in A} \pi(s,a)R(s,a) + \sum_{a \in A} \pi(s,a)\sum_{s' \in S} T(s,a,s')E_\pi [r_{t+1} + \ldots + r_T \mid s_{t+1} = s'] \]

*Expectation over 1-step transition*
The value of a policy

\[ V^\pi(s) = E_\pi [ r_t + r_{t+1} + \ldots + r_T | s_t = s ] \]

\[ V^\pi(s) = E_\pi [ r_t ] + E_\pi [ r_{t+1} + \ldots + r_T | s_t = s ] \]

\[ V^\pi(s) = \sum_{a \in A} \pi(s,a)R(s,a) + E_\pi [ r_{t+1} + \ldots + r_T | s_t = s ] \]

\[ V^\pi(s) = \sum_{a \in A} \pi(s,a)R(s,a) + \sum_{a \in A} \pi(s,a)\sum_{s' \in S} T(s,a,s')E_\pi [ r_{t+1} + \ldots + r_T | s_{t+1} = s' ] \]

\[ V^\pi(s) = \sum_{a \in A} \pi(s,a)R(s,a) + \sum_{a \in A} \pi(s,a)\sum_{s' \in S} T(s,a,s') V^\pi(s') \quad \text{By definition} \]

This is a **dynamic programming** algorithm.
The value of a policy

State value function (for a **fixed** policy):

\[ V^\pi(s) = \sum_{a \in A} \pi(s,a) \left[ R(s,a) + \gamma \sum_{s' \in S} T(s,a,s') V^\pi(s') \right] \]

*Immediate Future expected sum of rewards*

State-action value function:

\[ Q^\pi(s,a) = R(s,a) + \gamma \sum_{s'} T(s,a,s') \left[ \sum_{a' \in A} \pi(s',a') Q^\pi(s',a') \right] \]

These are two forms of **Bellman’s equation**.
The value of a policy

State value function:

\[ V_\pi(s) = \sum_{a \in A} \pi(s,a) \left( R(s,a) + \gamma \sum_{s' \in S} T(s,a,s') V_\pi(s') \right) \]

When \( S \) is a **finite set of states**, this is a **system of linear equations** (one per state) with a unique solution \( V_\pi \).

Bellman’s equation in matrix form:

\[ V_\pi = R_\pi + \gamma T_\pi V_\pi \]

Which can solved exactly:

\[ V_\pi = ( I - \gamma T_\pi )^{-1} R_\pi \]
Iterative Policy Evaluation: Fixed policy

Main idea: turn Bellman equations into update rules.

1. Start with some initial guess $V_0(s), \forall s$. (Can be 0, or $r(s, \cdot)$.)
Iterative Policy Evaluation: Fixed policy

Main idea: turn Bellman equations into update rules.

1. Start with some initial guess $V_0(s), \forall s$. (Can be 0, or $r(s, \cdot)$.)

2. During every iteration $k$, update the value function for all states:

$$V_{k+1}(s) \leftarrow \left( R(s, \pi(s)) + \gamma \sum_{s' \in S} T(s, \pi(s), s') V_k(s') \right)$$
Iterative Policy Evaluation: Fixed policy

Main idea: turn Bellman equations into update rules.

1. Start with some initial guess $V_0(s), \forall s$. (Can be 0, or $r(s, \cdot)$.)

2. During every iteration $k$, update the value function for all states:

   $$V_{k+1}(s) \leftarrow \left( R(s, \pi(s)) + \gamma \sum_{s' \in S} T(s, \pi(s), s') V_k(s') \right)$$

3. Stop when the maximum changes between two iterations is smaller than a desired threshold (the values stop changing.)

   This is a dynamic programming algorithm. Guaranteed to converge!
Convergence of Iterative Policy Evaluation

- Consider the absolute error in our estimate $V_{k+1}(s)$:

\[
|V_{k+1}(s) - V^\pi(s)| = \left| \sum_a \pi(s, a)(R(s, a) + \gamma \sum_{s'} T(s, a, s')V_k(s')) - \sum_{a} \pi(s, a)(R(s, a) + \gamma \sum_{s'} T(s, a, s')V^\pi(s')) \right|
\]

\[
= \gamma \left| \sum_{a} \pi(s, a) \sum_{s'} T(s, a, s')(V_k(s') - V^\pi(s')) \right|
\]

\[
\leq \gamma \sum_{a} \pi(s, a) \sum_{s'} T(s, a, s')|V_k(s') - V^\pi(s')|
\]

- As long as $\gamma < 1$, the error contracts and eventually goes to 0.
Optimal policies and optimal value functions

• **Optimal value function**, $V^*$ is the highest value that can be achieved for each state:

$$V^*(s) = \max_{\pi} V^\pi(s)$$

• Any policy that achieves $V^*$ is called an **optimal policy**, $\pi^*$. 
Optimal policies and optimal value functions

• **Optimal value function**, $V^*$ is the highest value that can be achieved for each state:

$$V^*(s) = \max_\pi V^\pi(s)$$

• Any policy that achieves $V^*$ is called an **optimal policy**, $\pi^*$.

• For each MDP there is a **unique optimal value function** (

  *Bellman, 1957*).

• The optimal policy is not necessarily unique.
Optimal policies and optimal value functions

• If we know $V^*$ (and $R$, $T$, $\gamma$), then we can compute $\pi^*$ easily.

$$\pi^*(s) = \arg \max_{a \in A} \left( R(s, a) + \gamma \sum_{s' \in S} T(s, a, s') V^*(s') \right)$$

• If we know $\pi^*$ (and $R$, $T$, $\gamma$), then we can compute $V^*$ easily.

$$V^*(s) = \sum_{a \in A} \pi^*(s, a) \left( R(s, a) + \gamma \sum_{s' \in S} T(s, a, s') V^*(s') \right)$$

$$V^*(s) = R(s, \pi(s)) + \gamma \sum_{s' \in S} T(s, \pi(s), s') V^*(s')$$

Take-home: Both $V^*$ and $\pi^*$ are “solutions” to the MDP.
Finding a good policy: **Policy Iteration**

- Start with an initial policy \( \pi_0 \) (e.g. random)

- Repeat:
  - Compute \( V^\pi \), using iterative policy evaluation.
  - Compute a new policy \( \pi' \) that is greedy with respect to \( V^\pi \)

- Terminate when \( \pi = \pi' \)
Main idea: Turn the Bellman optimality equation into an iterative update rule (same as done in policy evaluation):

1. Start with an arbitrary initial approximation $V_0(s)$

2. On each iteration, update the value function estimate:
   \[ V_k(s) = \max_{a \in A} \left( R(s,a) + \gamma \sum_{s' \in S} T(s,a,s')V_{k-1}(s') \right) \]

3. Stop when max value change between iterations is below threshold.

The algorithm converges (in the limit) to the true $V^*$. 
Three related algorithms

1. **Policy evaluation:** Fix the policy, estimate its value.

2. **Policy iteration:** Find the best policy at each state.
   » Policy evaluation + greedy improvement.

3. **Value iteration:** Find the optimal value function.
Three related algorithms

1. **Policy evaluation**: Fix the policy, estimate its value.
   - \( O(S^3) \)

2. **Policy iteration**: Find the best policy at each state.
   » Policy evaluation + greedy improvement.
   - \( O(S^3 + S^2A) \) per iteration

3. **Value iteration**: Find the optimal value function.
   - \( O(S^2A) \) per iteration
A 4x3 gridworld example

- 11 discrete states, 4 motion actions (N, S, E, W) in each state.
- Transitions are mildly stochastic.
- Reward is +1 in top right state, -10 in state directly below, -0 elsewhere.
- Episode terminates when the agent reaches +1 or -10 state.
- Discount factor $\gamma = 0.99$. 

![Intended direction](image_url)
# Value Iteration (1)

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>+1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>-10</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Value Iteration (2)

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0.69</td>
<td>+1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>-0.99</td>
<td>-10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>-0.99</td>
<td></td>
</tr>
</tbody>
</table>

Bellman residual: \[ |V_2(s) - V_1(s)| = 0.99 \]
Value Iteration (5)

Bellman residual: \[ |V_5(s) - V_4(s)| = 0.23 \]
Value Iteration (20)

<table>
<thead>
<tr>
<th></th>
<th>0.78</th>
<th>0.80</th>
<th>0.81</th>
<th>+1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.77</td>
<td>-0.44</td>
<td>-10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.75</td>
<td>0.69</td>
<td>0.37</td>
<td>-0.92</td>
<td></td>
</tr>
</tbody>
</table>

Bellman residual: $|V_5(s) - V_4(s)| = 0.008$
Another example: Four Rooms

- Four actions, fail 30% of the time.
- No rewards until the goal is reached, $\gamma = 0.9$.
- Values propagate backwards from the goal.
Asynchronous value iteration

• Instead of updating all states on every iteration, focus on *important states*.
  – E.g., board positions that occur on every game, rather than just once in 100 games.

• **Asynchronous dynamic programming algorithm:**
  – Generate trajectories through the MDP.
  – Update states whenever they appear on such a trajectory.

• Focuses the updates on states that are actually possible.
Generalized Policy Iteration

- Any combination of policy evaluation and policy improvement steps.
  e.g. only update value of one state and improve policy at that state.
Key challenges in RL

• Designing the problem domain
  – State representation
  – Action choice
  – Cost/reward signal

• Acquiring data for training
  – Exploration / exploitation
  – High cost actions
  – Time-delayed cost/reward signal

• Function approximation

• Validation / confidence measures
Learning online from trial & error

\[ Q, \pi \Rightarrow a_t \Rightarrow S_t \rightarrow_{a} R_{t} \Rightarrow S_{t+1} \]
Online reinforcement learning

- **Monte-Carlo** value estimate: Use the empirical return, $U(s_t)$ as a target estimate for the actual value function:

\[
V(s_t) \leftarrow V(s_t) + \alpha(U(s_t) - V(s_t))
\]

*Not a Bellman equation. More like a gradient equation.*

- Here $\alpha$ is the learning rate (a parameter).

- Need to wait until the end of the trajectory to compute $U(s_t)$. 
Temporal-Difference (TD) learning

- Monte-Carlo learning: \( V(s_t) \leftarrow V(s_t) + \alpha(U(s_t) - V(s_t)) \)

- TD-learning:

\[
V(s_t) \leftarrow V(s_t) + \alpha(r_{t+1} + \gamma V(s_{t+1}) - V(s_t)) \quad \forall t = 0, 1, 2, \ldots
\]
TD-Gammon (Tesauro, 1992)

Reward function:
+100 if win
- 100 if lose
0 for all other states

Trained by playing $1.5 \times 10^6$ million games against itself.

Enough to beat the best human player.
Several challenges in RL

- Designing the problem domain
  - State representation
  - Action choice
  - Cost/reward signal

- Acquiring data for training
  - Exploration / exploitation
  - High cost actions

- Time-delayed cost/reward signal

- Function approximation

- Validation / confidence measures
Tabular / Function approximation

- **Tabular**: Can store in memory a list of the states and their value.

- **Function approximation**: Too many states, continuous state spaces.

*Can prove many more theoretical properties in this case, about convergence, sample complexity.*
In large state spaces: Need approximation

\[ \hat{Q}^\pi (s, a) = \sum_{i=1}^{d} \theta_i \phi_i (s, a) \]

Challenge: finding good features
Learning representations for RL

Original state

Linear function

$Q_{\theta}(s,a)$
Deep Reinforcement Learning

Original state

Convolutional Neural Net

Deep Q-Network trained with stochastic gradient descent.

[DeepMind: Mnih et al., 2015]
Deep RL in Minecraft

Many possible architectures, incl. memory and context

Online videos: https://sites.google.com/a/umich.edu/junhyuk-oh/icml2016-minecraft

[U.Michigan: Oh et al., 2016].
The RL lingo

- Episodic / Continuing task
- Batch / Online
- On-policy / Off-policy
- Exploration / Exploitation
- Model-based / Model-free
- Policy optimization / Value function methods
On-policy / Off-policy

• Policy induces a distribution over the states (data).
  – Data distribution changes every time you change the policy!
On-policy / Off-policy

- Policy induces a distribution over the states (data).
  - Data distribution **changes** every time you change the policy!

- Evaluating several policies with the same batch:
  - Need very big batch!
  - Need policy to adequately cover all \((s,a)\) pairs.
On-policy / Off-policy

• Policy induces a distribution over the states (data).
  – Data distribution **changes** every time you change the policy!

• Evaluating several policies with the same batch:
  – Need very big batch!
  – Need policy to adequately cover all \((s,a)\) pairs.

• Use importance sampling to reweigh data samples to compute unbiased estimates of a new policy.

\[ \rho_t = \frac{\pi(s_t,a_t)}{b(s_t,a_t)} \]
Exploration / Exploitation
**Exploration / Exploitation**

**Exploration**: Increase knowledge for long-term gain, possibly at the expense of short-term gain.

**Exploitation**: Leverage current knowledge to maximize short-term gain.
Model-based vs Model-free RL

• **Option #1**: Collect large amounts of observed trajectories. Learn an approximate model of the dynamics (e.g. with supervised learning). Pretend the model is correct and apply value iteration.

• **Option #2**: Use data to directly learn the value function or optimal policy.
Policy Optimization / Value Function

Policy Optimization
- DFO / Evolution
- Policy Gradients
- Actor-Critic Methods

Dynamic Programming
- Policy Iteration
- Value Iteration
- Q-Learning
- TD-Learning
Quick summary

• RL problems are everywhere!
  – Games, text, robotics, medicine, …

• Need access to the “environment” to generate samples.
  – Most recent results make extensive use of a simulator.

• Feasible methods for large, complex tasks.

• Intuition about what is “easy”, “hard” is different than supervised learning.
RL resources

Comprehensive list of resources:
• https://github.com/aikorea/awesome-rl

Environments & algorithms:
• http://glue.rl-community.org/wiki/Main_Page
• https://gym.openai.com
• https://github.com/deepmind/lab