Reinforcement Learning

Typical Machine Learning
- Lots of Data
- Passive Learner
- Learning = pattern analysis, function approx.

Examples
* Find photos of cats.
* Product recommendation
* Text Translation

Robots are Not Like This!

Environment

1. Robot senses world
2. Robot decides and executes action
3. World state changes
4. Robot receives a reward
5. Robot updates its "strategy"
6. Go to 1

RL:
RL is a process of modifying behavior by rewarding desired outcomes.

- Dogs - treats, learn tricks
  - Dogs - punishment → negative reward
- Kids - grades
  - Joy, satisfaction
  - Starting salary

RL Process:
1. Robot senses world
2. Robot decides and executes action
3. World state changes
4. Robot receives a reward
5. Robot updates its "strategy"
6. Go to 1
**Questions:**
- Mathematical Model
  - states
  - actions
  - uncertainty (!)
- Mathematical formalism for reward
- Given the above
  How to compute an optimal strategy/policy
- Now... suppose we don't know the models for world, actions... how can we learn them?

**Markov Processes**

**Simplest case:**
- discrete time, \( k = 0, 1, 2, \ldots \)
- discrete states, \( S \) set of states.
- A set of transition probabilities

\[
T : S \times S \rightarrow [0, 1]
\]

probability

\[
P \{ S_{kn} | S_k \} = T (s, s')
= T (S_k, S_{kn})
\]

\[
T (s, s') = \text{prob} \{ \text{arriving to state } s', \text{ given we are now in state } s \}
\]

\[\rightarrow \text{ M. P. evolves "autonomously" }\]
Example:
Sisyphus has the job of throwing a large rock on a circular track.

\[ S = \{ A, B, \ldots, L \} \]
\[ T(\{ A, B \}) = 0.25 \quad T(\{ A, C \}) = 0.5 \quad T(A, D) = 0.25 \]
\[ T(\{ J, K \}) = 0.25 \quad T(\{ J, L \}) = 0.5 \quad T(J, A) = 0.25 \]
\[ T(K, L) = 0.25 \quad T(K, A) = 0.5 \quad T(K, B) = 0.25 \]
\[ T(s, s') = 0 \text{ otherwise} \]

Properties:
- Stationary: \( T \) does not change over time

\[ P\{ L_k = 1 \} = 0.25 \]
\[ P\{ L_k = 2 \} = 0.5 \]
\[ P\{ L_k = 3 \} = 0.25 \]

Markov Property
\[ P\{ s_{k+1} = E \mid s_0 = A, s_1 = C, s_2 = D \} = P\{ s_{k+1} = E \mid s_k \} \]
Markov Decision Processes (MDPs)

Markov Processes evolve autonomously.
Let's give Sisyphus a tiny bit of free will:

- R: Throw rock counterclockwise
- L: Throw rock clockwise

Same throwing abilities for L & R

\[ P_{s'} = P_{s} \] unchanged

\[ T(A, L, B) = 0.25 \quad T(A, L, C) = 0.5 \quad T(A, L, D) = 0.25 \]

\[ T(s, a, s') = P\{s' | s, a\} \]

\[ L = \text{action} \]

\[ T(A, R, L) = 0.25 \quad T(A, R, K) = 0.5 \quad T(A, R, J) = 0.25 \]

Rewards: \[ R: S^* \rightarrow \mathbb{R} \]
For Sisyphus: \[ R(E) = +1 \]
\[ R(s) = -0.2 \text{ for } s \neq E \]

Suppose Sisyphus executes two actions:
\[ a_1 = L, \quad a_2 = L \]
\[ c_{k=1} \quad c_{k=2} \]
Expectation

Suppose a random variable $X$ takes values from the set \{ $c_1, c_2, \ldots, c_n$ \}. The expected value of $X$ is defined as

$$E[X] = \sum_{i=1}^{n} P\{X = c_i\} \cdot c_i$$

Example: roll one die, $X$ shows

$$E[X] = \sum_{i=1}^{6} \frac{1}{6} \cdot x_i = 3.5$$

down values equally likely.

Intuition: Perform this many times

Average of $X$'s $\rightarrow E[X]$
**Expected Return**

Define return $r = \sum_{i=0}^{n} R(s_i)$. 

$R_2$ for surplus = $R(0) + R(s_1) + R(s_2)$

$E[R_2] = E[R(s_1) + R(s_2)]$

$P[R_3 = E^3] = P \left[ \frac{r_h}{0.6} \times 0.6 + P \left[ \frac{r_h}{-0.6} \times -0.6 \right] \right]$

because $r_h$ has only two possible values:

- $0.2 + 0.2 + 0.2 = 0.6 \Rightarrow S_2 \neq E$
- $0.2 + 0.2 + 1 = 0.6 \Rightarrow S_2 = E$

To arrive at $S_2 = E$

<table>
<thead>
<tr>
<th>$S_0$</th>
<th>$S_1$</th>
<th>$S_2$</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
<td>E</td>
<td>(0.25) x (0.25) = 0.0625</td>
</tr>
<tr>
<td>A</td>
<td>C</td>
<td>E</td>
<td>(0.5) x (0.5) = 0.25</td>
</tr>
<tr>
<td>A</td>
<td>D</td>
<td>E</td>
<td>(0.25) x (0.25) = 0.0625</td>
</tr>
</tbody>
</table>

$P(S_3 = E) = 0.375$

$P(S_3 \neq E) = 1 - P[S_3 = E]$

$= 0.625$

This is great for finite horizons (i.e., only consider a finite # of stages).

Suppose $h \to \infty$??

To deal with this, use discounted rewards.

$r_h = \sum_{i=0}^{n} \gamma^i R(s_i)$

for $0 \leq \gamma < 1$.

$$\lim_{h \to \infty} \sum_{i=0}^{t} \gamma R(s_i) \leq \frac{R_{max}}{1 - \gamma}$$
Policies

\[ E[\tau_n] = E \left[ \sum_{i=0}^{n} \gamma^i R(s_i) \mid a, \ldots, a_n \right] \]

Def: A policy \( \pi: S \rightarrow A \) specifies \( a = \pi(s) \), the action to take in states.

For a policy \( \pi \)

\[ V^\pi(s) = E[\gamma_0(s) \mid \pi] \]

\[ = E \left[ \sum_{i=0}^{\infty} \gamma^i R(s_i) \mid s_0 = s, \pi \right] \]

\[ = E \left[ R(s_0) + \sum_{i=1}^{\infty} \gamma^i R(s_i) \mid s_0 = s, \pi \right] \]

\[ = R(s) + E \left[ \gamma \sum_{i=1}^{\infty} \gamma^{i-1} R(s_i) \mid \pi \right] \]

\[ = R(s) + \gamma E \left[ \sum_{i=0}^{\infty} \gamma^i R(s_{j+1}) \mid \pi \right] \]

Expected return under \( \pi \) from state \( s_j+1 \)

\[ V^\pi(s) = R(s) + \gamma \sum_{s'} T(s, \pi(s), s') V^\pi(s') \]

Expected return for executing \( a = \pi(s) \) from state \( s \).

We want the optimal policy, \( \pi^* \)

\[ \pi^* = \arg \max_{\pi} V^\pi(s) \]

Def: The value function

\[ V^*(s) = V^{\pi^*}(s) \]

\[ V^*: S \rightarrow \mathbb{R} \]

is the value for \( \pi^* \)

\[ \pi^*(s) = \arg \max_{a \in A} \sum_{s'} T(s, a, s') V^*(s') \]

Expected return for executing action \( a \) in state \( s \).
Bellman Equation

\[ \pi^* = \arg \max_a \sum_s T(s,a,s') V^*(s') \]

\[ V^*(s) = R(s) + \gamma \max_a \sum_{s'} T(s,a,s') V^*(s') \]

**Bellman Eqn**

**Value Iteration**

How to find \( V^* \)?

Suppose we have an estimate \( V^k \) of \( V^* \) and we want to iteratively improve

\[ V^k \rightarrow V^* \]

the unique solution to Bellman eqn.

\[ V^{k+1}(s) = R(s) + \gamma \max_a \sum_{s'} T(s,a,s') V^k(s') \]

Best guess at \( V^* \) at \( k^{th} \) iteration

If it works

if \( V^k = V^* \)

what we would have
<table>
<thead>
<tr>
<th></th>
<th>V̄</th>
<th>V̄</th>
<th>V₂(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>+1</td>
<td>0.3</td>
<td>-0.05</td>
</tr>
<tr>
<td>B</td>
<td>+1</td>
<td>0.3</td>
<td>0.1</td>
</tr>
<tr>
<td>C</td>
<td>+1</td>
<td>0.3</td>
<td>0.25</td>
</tr>
<tr>
<td>D</td>
<td>+1</td>
<td>0.3</td>
<td>0.1</td>
</tr>
<tr>
<td>E</td>
<td>+1</td>
<td>1.5</td>
<td>1.15</td>
</tr>
<tr>
<td>F</td>
<td>+1</td>
<td>0.3</td>
<td>0.1</td>
</tr>
<tr>
<td>G</td>
<td>+1</td>
<td>0.3</td>
<td>0.25</td>
</tr>
<tr>
<td>H</td>
<td>+1</td>
<td>0.3</td>
<td>0.1</td>
</tr>
<tr>
<td>I</td>
<td>+1</td>
<td>-0.3</td>
<td>-0.05</td>
</tr>
<tr>
<td>J</td>
<td>+1</td>
<td>-0.3</td>
<td>-0.05</td>
</tr>
<tr>
<td>K</td>
<td>+1</td>
<td>0.3</td>
<td>-0.05</td>
</tr>
</tbody>
</table>

### V₂(s)

For A, I, S, K, L

\[ V₂(s) = R(s) + 0.5 \max_{a \in \{R, L\}} T(s, a, s') V₁(s') \]

\[ V₂(s) = -0.2 + 0.5 \max \{0.25 \times 3 + 0.5 \times 3 + 0.25 \times 3, 0.25 \times 3 + 0.5 \times 3 + 0.25 \times 3\} \]

= -0.05

\[ S = R \]

\[ V₂(B) = -0.2 + 0.5 \max \{0.3, 0.25 \times 3 + 0.5 \times 1.5 + 0.25 \times 3 \} \]

= -0.2 + 0.5(0.6)

= -0.2 + 0.3

= -0.1

\[ \gamma = 0.5 \]

\[ V^{k+1}(s) = R(s) + \gamma \max_{a \in \{R, L\}} \sum_{s'} T(s, a, s') V^k(s) \]

For S ≠ E, R(s) = -0.2

\[ V₁(s) = -0.2 + 0.5 \max \{0.25 \times 1 + 0.25 \times 1 + 0.25 \times 1, 0.25 \times 1 + 0.5 \times 1 \times 0.25 \times 1\} \]

= -0.2 + 0.5(0.3)

= -0.2 + 0.15 = 0.3
Policy Iteration

Let $\Pi^i$ be approximation to $\Pi^*$ at $i$th iteration.

$$
\Pi^{i+1}(s) = \arg \max_{a \in A} \sum_{s' \in S} T(s, \Pi^i(a), s') V^i(s')
$$

- $V^i(s)$ is current guess for $V^*$
- $\Pi^i$ is current guess for $\Pi^*$

$$
V^{i+1}(s) = R(s) + \gamma \sum_{s' \in S} T(s, \Pi^{i+1}(s), s') V^i(s')
$$

Looks like expected return under $\Pi^{i+1}$
Reinforcement Learning

we have done a lot of work:

$\rightarrow$ MDP = ($S, A, T, R, \pi$)

$\rightarrow$ Bellman Eqn + Value Function

$V^*(s) = R(s) + \gamma \max_a \sum_s P(s, a, s') V^*(s')$

$\rightarrow$ Optimal Policy

$\pi^*(s) = \arg \max_a \sum_s P(s, a, s') V^*(s')$

For Machine Learning

* We don't know $T$.
* We don't know $R(s)$

R.L. Two Approaches

Passive: The agent has a fixed policy $\pi$, and it can learn about $T$ and/or $V^\pi$ by executing $\pi$.

Active: The $\pi$ policy is not fixed. The agent learns about $V^*$ and/or $T$, while learning to act optimally.
Direct Utility Estimation

\[ V^\pi(s) = E \left[ \sum_{i=0}^{\infty} \gamma^i R(s_i) \mid \pi, s_0 = s \right] \]

Idea: execute \( \pi \) from \( s \) many times. The average of the returns is a good approximation for \( V^\pi(s) \).

Can't execute \( \infty \) actions, so execute over a finite horizon, length \( h \).

\[ \Gamma_h(s_0, \ldots) = \sum_{i=0}^{h} \gamma^i R(s_i) \]

\[ = \left( \sum_{i=0}^{h} \gamma^i R(s_i) \right) + \sum_{i=h+1}^{\infty} \gamma^i R(s_i) \]

**Error in approximation**

Error = \[ \sum_{i=h+1}^{\infty} \gamma^i R(s_i) \leq \sum_{i=h+1}^{\infty} \gamma^i R_{\text{max}} = \sum_{i=0}^{\infty} \gamma^{i+h+1} \]

\[ R_{\text{max}} = \gamma \sum_{i=0}^{\infty} \gamma^i \]

\[ = \gamma^{h+1} \sum_{i=0}^{\infty} \gamma^i = \frac{\gamma^{h+1}}{1-\gamma} \leq \text{Bound on the error} \]
Adaptive Dynamic Programming (ADP)

Recall:
\[ V^\pi(s) = R(s) + \gamma \sum_{s' \in S'} T(s, \pi(m), s') V^\pi(s') \] ★

Idea: at each stage of execution, execute \( a = \pi(s) \) in state \( s \) and arrive to state \( s' \Rightarrow s, a, s' \), we observe \( R(s') \)

At each stage:
- For each \( s \), update \( \hat{T}(s, a, s) = \frac{N_{sas} \hat{N}_{s}}{N_{sa}} \)
- Update \( \hat{V}^\pi(s) \leftarrow R(s) + \gamma \sum_{s' \in S'} \hat{T}(s, a, s') \hat{V}^\pi(s') \)

\( N_{sas} \) = # of times we experience \( s, a, s' \)
\( N_{sa} \) = # of times we execute action \( a \) from state \( s \)

Keep in a table \( \hat{T} \) and use D.P. to implement this.

Execute $a$ in states $s$ and reach $s'$: $\Rightarrow s, a, s'$

$R(s) + \gamma V'_{\pi}(s') \Rightarrow$ after experience $s, a, s'$

Before experiencing $s, a, s'$, the best guess for return is $V_{\pi}(s)$

$[R(s) + \gamma V'_{\pi}(s')] - V_{\pi}(s)$

Temporal Difference.

No Model of $T$

Updating Scheme:

$V_{\pi}(s) \leftarrow V_{\pi}(s) + \alpha [R(s) + \gamma V_{\pi}(s') - V_{\pi}(s)]$

TD equation

Learning rate

ADP vs. TD

ADP: $V_{\pi}$ is made to agree with all past experience.

TD: $V_{\pi}$ is made to agree only with current experience.
Active Learning

- If we have a model $T$, we can use $T$ to build an approximation of $V^*$, and thus $\hat{\Pi}^*$. [e.g., using Value or Policy Iteration]

Q Given $\hat{T} \Rightarrow \hat{V}^*$, $\hat{\Pi}^*$ should we

a. execute $\hat{\Pi}^* (s)$ —— Exploitation

b. Try some other action, $a$, —— Exploration
   maybe finding better $\hat{V}^*$

→ key problem: Balancing Exploration vs. Exploitation

Many available Algorithms to do this——
Q-Learning: For active learning, we need to explicitly consider the action \( a \) when deciding what to do. \( V^*(s) \) does not explicitly consider the action to be performed.

\[
V^*(s) = R(s) + \gamma \max_a \sum_{s'} T(s,a,s') V^*(s')
\]

Constraint eqn for Q-function

\[
Q(a,s) = R(s) + \gamma \sum_{s'} T(s,a,s') V^*(s')
\]

\[
Q(a,s) = R(s) + \gamma \sum_{s'} T(s,a,s') \max_{a'} Q(a',s')
\]
TD Q-Learning

Given Q at some stage of learning.
We execute action a in states, arrive to s'
⇒ s, a, s'

\[
Q(a, s) \leftarrow Q(a, s) + \alpha \left[ R(s') + \gamma \max_{a'} Q(a', s') - Q(a, s) \right]
\]

- Current estimate of Q
- Estimate of Q based on having seen s, a, s'
- Learning rate
- Estimate of Q before seeing s, a, s'

Note: We never build a model for T ⇒ Model-Free.