The basic background & assumptions

- Environment is a finite MDP (i.e. $A$ and $S$ are finite).
- MDP's dynamics defined by transition probabilities:
  $$P^a_{s's} = P(s_{t+1} = s'|s_t = s, a_t = a)$$
- and expected immediate rewards,
  $$R^a_{s's'} = E\{r_{t+1}|s_t = s, a_t = a, s_{t+1} = s'\}$$
- Goal: to search for good policies $\pi$
- Strategy: use value functions to structure search:
  $$V^*(s) = \max_a \sum_{s'} P^a_{s's'} [R^a_{s's'} + \gamma V^*(s')]$$
  or
  $$Q^*(s,a) = \sum_{s'} P^a_{s's'} [R^a_{s's'} + \gamma \max_{a'} Q^*(s',a')]$$

Overview of methods for solving Bellman’s equations

- **Dynamic programming**:
  - well-understood mathematical properties...
  - ...but require a complete and accurate model of the environment
- **Monte Carlo** (simulation methods):
  - conceptually simple
  - no model required...
  - ...but unsuitable for incremental computation
- **Temporal difference** methods
  - also require no model;
  - suitable for incremental computation...
  - ... but mathematically complex to analyse
Dynamic programming

- Basic Idea: “sweep” through $S$ performing a full backup operation on each $s$.
- A few different methods exist. E.g.:
  - Policy Iteration and
  - Value Iteration.
- The building blocks:
  - Policy Evaluation: how to compute $V^\pi$ for an arbitrary $\pi$.
  - Policy Improvement: how to compute an improved $\pi$ given $V^\pi$.

Policy Evaluation

- The task of computing $V^\pi$ for an arbitrary $\pi$ is known as the prediction problem.
- As we have seen, a state-value function is given by
  \[ V^\pi(s) = E_\pi\{ R_t | s_t = s \} = E_\pi\{ r_{t+1} + \gamma V^\pi(s_{t+1}) | s_t = s \} = \sum_a \pi(s,a) \sum_{s'} P^a_{ss'}[R^a_{ss'} + \gamma V^\pi(s')] \]

- a system of $|S|$ linear equations in $|S|$ unknowns (the state values $V^\pi(s)$)

Iterative Policy Evaluation

- Consider the sequence of approximations $V_0, \ldots V^\pi$.
- Choose $V_0$ arbitrarily and set each successive approximation to accommodate to the Bellman equation:
  \[ V_{k+1}(s) \leftarrow E_\pi\{ r_{t+1} + \gamma V_k(s_{t+1}) | s_t = s \} \leftarrow \sum_a \pi(s,a) \sum_{s'} P^a_{ss'}[R^a_{ss'} + \gamma V_k(s')] \]  \[ \Delta \leftarrow \max(\Delta, |V_{k+1}(s) - V_k(s)|) \]
- “Sweeps”: $V_0 \rightarrow V_1 \rightarrow \ldots \rightarrow V_k \rightarrow V_{k+1} \ldots \rightarrow V^\pi$

Iterative Policy Evaluation Algorithm

<table>
<thead>
<tr>
<th>I n i t i a l i z a t i o n :</th>
</tr>
</thead>
<tbody>
<tr>
<td>for ( each $s \in S$)</td>
</tr>
<tr>
<td>$V(s) \leftarrow 0$</td>
</tr>
</tbody>
</table>

| I P E ( $\pi$ ) | /* $\pi$: policy to be evaluated */ |
|-----------------|
| repeat |
| $\Delta \leftarrow 0$ |
| $V_k \leftarrow V$ |
| for ( each $s \in S/\{s_{\text{terminal}}\}$) |
| $V(s) \leftarrow \sum_a \pi(s,a) \sum_{s'} P^a_{ss'}[R^a_{ss'} + \gamma V_k(s')]$ |
| $\Delta \leftarrow \max(\Delta, |V_k(s) - V(s)|)$ |
| until $\Delta < \theta$ | /* $\theta > 0$: a small constant */ |
| return $V$ | /* $V \approx V^\pi$ */ |

- NB: alternatively one could evaluate $V^\pi$ in place (i.e using a single vector $V$ to store all values and update it directly).
An example: an episodic GridWorld

- Rewards of \(-1\) until terminal state (shown in grey) is reached
- Undiscounted episodic task:

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
</tr>
<tr>
<td>12</td>
<td>13</td>
<td>14</td>
<td></td>
</tr>
</tbody>
</table>

\( r = -1 \) on all transitions

Policy Evaluation for the GridWorld

- Iterative evaluation of \( V_k \) for equiprobable random policy \( \pi \):

\[
V_0 \rightarrow \begin{array}{c}
0.0 \ 0.0 \ 0.0 \ 0.0 \\
0.0 \ 0.0 \ 0.0 \ 0.0 \\
0.0 \ 0.0 \ 0.0 \ 0.0 \\
0.0 \ 0.0 \ 0.0 \ 0.0 \\
0.0 \ -2.4 \ -2.9 \ -3.0 \\
-2.4 \ -2.9 \ -3.0 \ -2.9 \\
-2.9 \ -3.0 \ -2.9 \ -2.4 \\
-3.0 \ -2.9 \ -2.4 \ 0.0 \\
\end{array}
\]

\[
V_1 \rightarrow \begin{array}{c}
0.0 \ -1.0 \ -1.0 \ -1.0 \\
-1.0 \ -1.0 \ -1.0 \ -1.0 \\
-1.0 \ -1.0 \ -1.0 \ -1.0 \\
-1.0 \ -1.0 \ -1.0 \ 0.0 \\
0.0 \ -6.1 \ -8.4 \ -9.0 \\
-6.1 \ -7.7 \ -8.4 \ -8.4 \\
-8.4 \ -8.4 \ -7.7 \ -6.1 \\
-9.0 \ -8.4 \ -6.1 \ 0.0 \\
\end{array}
\]

\[
V_2 \rightarrow \begin{array}{c}
0.0 \ -1.7 \ -2.0 \ -2.0 \\
-1.7 \ -2.0 \ -2.0 \ -2.0 \\
-2.0 \ -2.0 \ -2.0 \ -1.7 \\
-2.0 \ -2.0 \ -1.7 \ 0.0 \\
-14 \ -18 \ -20 \ -22 \\
-14 \ -18 \ -20 \ -20 \\
-20 \ -20 \ -18 \ -14 \\
-22 \ -20 \ -14 \ 0.0 \\
\end{array}
\]

\[
V_3 \rightarrow \begin{array}{c}
0.0 \ -2.4 \ -2.9 \ -3.0 \\
-2.4 \ -2.9 \ -3.0 \ -2.9 \\
-2.9 \ -3.0 \ -2.9 \ -2.4 \\
-3.0 \ -2.9 \ -2.4 \ 0.0 \\
-8.4 \ -8.4 \ -7.7 \ -6.1 \\
-8.4 \ -8.4 \ -7.7 \ -6.1 \\
-9.0 \ -8.4 \ -6.1 \ 0.0 \\
-9.0 \ -8.4 \ -6.1 \ 0.0 \\
\end{array}
\]

\[
V_{10} \rightarrow \begin{array}{c}
0.0 \ -1.7 \ -2.0 \ -2.0 \\
-1.7 \ -2.0 \ -2.0 \ -2.0 \\
-2.0 \ -2.0 \ -2.0 \ -1.7 \\
-2.0 \ -2.0 \ -1.7 \ 0.0 \\
-14 \ -18 \ -20 \ -22 \\
-14 \ -18 \ -20 \ -20 \\
-20 \ -20 \ -18 \ -14 \\
-22 \ -20 \ -14 \ 0.0 \\
\end{array}
\]

\[
V_{\infty} \rightarrow \begin{array}{c}
0.0 \ -14 \ -20 \ -22 \\
-14 \ -18 \ -20 \ -22 \\
-20 \ -20 \ -18 \ -14 \\
-22 \ -20 \ -14 \ 0.0 \\
-14 \ -20 \ -22 \\
-14 \ -18 \ -20 \ -22 \\
-20 \ -20 \ -18 \ -14 \\
-22 \ -20 \ -14 \ 0.0 \\
\end{array}
\]

Policy Improvement

- Consider the following: how would the expected return change for a policy \( \pi \) if instead of following \( \pi(s) \) for a given state \( s \) we choose an action \( a \neq \pi(s) \)?
- For this setting, the value would be:

\[
Q^\pi(s,a) = E_\pi \{ r_{t+1} + \gamma V^\pi(s_{t+1}) | s_t = s, a_t = a \} = \sum_{s'} P_{ss'}(\pi_a) [R_{ss'} + \gamma V^\pi(s_{t+1})]
\]

- So, \( a \) should be preferred iff \( Q^\pi(s,a) > V^\pi(s) \)

Policy improvement theorem

If choosing \( a \neq \pi(s) \) implies \( Q^\pi(s,a) \geq V^\pi(s) \) for a state \( s \), then the policy \( \pi' \) obtained by choosing \( a \) every time \( s \) is encountered (and following \( \pi \) otherwise) is at least as good as \( \pi \) (i.e. \( V^\pi(s) \geq V^{\pi'}(s) \)). If \( Q^\pi(s,a) > V^\pi(s) \) then \( V^\pi(s) > V^{\pi'}(s) \)
If we apply this strategy to all states to get a new greedy policy \( \pi'(s) = \arg \max_a Q_\pi(s,a) \), then \( V^\pi' \geq V^\pi \)

\[ V^\pi'(s) = \max_a \sum_{s'} P_{as'}^a [R_{as'}^a + \gamma V^\pi(s')] \]

which is... a form of the Bellman optimality equation.

Therefore \( V^\pi = V^\pi' = V^* \)

### Improving the GridWorld policy

| Initialisation: | for all \( s \in S \) | \( V(s) \leftarrow \text{an arbitrary } v \in \mathbb{R} \) |
| Policy Improvement (\( \pi \)): | do | stable (\( \pi \)) \leftarrow true |
| | V \leftarrow \text{IPE(\( \pi \))} | for each \( s \in S \) |
| | b \leftarrow \pi(s) | for each \( s \in S \) |
| | \pi(s) \leftarrow \arg \max_a \sum_{s'} P_{as'}^a [R_{as'}^a + \gamma V^\pi(s')] | if ( b \neq \pi(s)) |
| | stable (\( \pi \)) \leftarrow false | while (not stable (\( \pi \))) |
| | return \( \pi \) |

### Other DP methods

- **Value Iteration**: evaluation is stopped after a single sweep (one backup of each state). The backup rule is then:

\[ V_{k+1}(s) \leftarrow \max_a \sum_{s'} P_{as'}^a [R_{as'}^a + \gamma V_k(s')] \]
• Asynchronous DP: back up the values of states in any order, using whatever values of other states happen to be available.
  
  – On problems with large state spaces, asynchronous DP methods are often preferred

**Generalised policy iteration**

![Diagram of Generalised policy iteration]

**Value Iteration**

• Potential computational savings over Policy Iteration in terms of policy evaluation

---

**Initialisation:**

for all $s \in S$

$V(s) \leftarrow$ an arbitrary $v \in \mathbb{R}$

**Value Iteration ($\pi$):**

repeat

for each $s \in S$

$\pi \leftarrow V(s)$

$V(s) \leftarrow \max_a \sum_{s'} P_{a,s,s'} [R_{a,s,s'} + \gamma V(s')]$

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

until $\Delta < \theta$

return deterministic $\pi$ s.t.

$\pi(s) = \arg \max_a \sum_{s'} P_{a,s,s'} [R_{a,s,s'} + \gamma V(s')]$

---

**Summary of methods**
Monte Carlo Methods

- Complete knowledge of environment is not necessary
- Only experience is required
- Learning can be on-line (no model needed) or through simulated experience (model only needs to generate sample transitions.
  - In both cases, learning is based on averaged sample returns.
- As in DP, one can use an evaluation-improvement strategy.
- Evaluation can be done by keeping averages of:
  - Every Visit to a state in an episode, or
  - of the First Visit to a state in an episode.

Estimating value-state functions in MC

The first visit policy evaluation method: \[2em\]

```
FirstVisitMC(\pi)

Initialisation:
V \leftarrow \text{arbitrary state values}
Returns(s) \leftarrow \text{empty list of size } |S|

Repeat
    Generate an episode \( E \) using \( \pi \)
    For each \( s \) in \( E \)
        R \leftarrow \text{return following the first occurrence of } s
        Append R to Returns(s)
    V(s) \leftarrow \text{mean(Returns(s))}
```
**Example**

Evaluate the policy described below for blackjack (?, section 5.1)

- Actions: stick (stop receiving cards), hit (receive another card)
- Play against dealer, who has a fixed strategy (`hit` if $sum < 17$; `stick` otherwise).
- You win if your card sum is greater than the dealer’s without exceeding 21.
- States:
  - current sum (12-21)
  - dealers showing card (ace-10)
  - do I have a useable ace (can be 11 without making sum exceed 21)?
- Reward: +1 for winning, 0 for a draw, -1 for losing
- Policy: Stick if my sum is 20 or 21, else hit

Note: We are assuming sampling with replacement (something like an infinite card deck).

**Question**: What is the total number of states?

**MC value function for blackjack example**

![Diagram showing value function](attachment:diagram.png)

- Question: compare MC to DP in estimating the value function. Do we know the environment? The transition probabilities? The expected returns given each state and action? What does the MC backup diagram look like?

Unlike DP, MC explores entire episodes, exploring a single choice of successor state at a time. Therefore the algorithm doesn’t need to know the transition probabilities. The rewards are sampled by averaging over a large number of episodes. MC does not bootstrap; that is, unlike DP it doesn’t update estimates of the value of a state based on the estimated value of its successor.
Monte Carlo control

- Monte Carlo version of DP policy iteration:

\[
Q^{\pi_k}(s, \pi_{k+1}(s)) = \max_a Q^{\pi_k}(s, a) \\
\geq Q^{\pi_k}(s, a) = V^{\pi_k}(s)
\]

- *Policy improvement* theorem applies:

\[
Q^{\pi_k}(s, \pi_{k+1}(s)) = Q^{\pi_k}(s, \arg \max_a Q^{\pi_k}(s, a))
\]

MC policy iteration (exploring starts)

- As with DP, we have *evaluation-improvement* cycles.
- Learn \( Q^* \) (if no model is available)
- One must make sure that each state-action pair can be a starting pair (with probability > 0).

---

1. MonteCarloES( )
2. Initialization, \( \forall s \in S, a \in A: \)
3. \( Q(s, a) \leftarrow \text{arbitrary}; \pi(s) \leftarrow \text{arbitrary} \)
4. \( \text{Returns}(s, a) \leftarrow \text{empty list of size } |S| \)
5. Repeat until stop-criterion met
6. Generate an episode \( E \) using \( \pi \) and exploring starts
7. For each \( (s, a) \) in \( E \)
8. \( R \leftarrow \text{return following the first occurrence of } s, a \)
9. Append \( R \) to \( \text{Returns}(s, a) \)
10. \( Q(s, a) \leftarrow \text{mean}(\text{Returns}(s, a)) \)
11. For each \( s \) in \( E \)
12. \( \pi(s) \leftarrow \arg \max_a Q(s, a) \)

---

Optimal policy for blackjack example

- Optimal policy found by MonteCarloES for the blackjack example, and its *state-value* function:
On- and off-policy MC control

- **MonteCarloES** assumes that all states are observed an infinite number of times and episodes are generated with exploring starts
  - For an analysis of convergence properties, see (?)
- On-policy and off-policy methods relax these assumptions to produce practical algorithms
- On-policy methods use a given policy and $\epsilon$-greedy strategy (see lecture on Evaluative Feedback) to generate episodes.
- Off-policy methods evaluate a policy while generating an episode through a different policy

On-policy control

Initialize, for all $s \in S$, $a \in A(s)$:
- $Q(s, a) \leftarrow$ arbitrary
- $\text{Returns}(s, a) \leftarrow$ empty list
- $\pi \leftarrow$ an arbitrary $\epsilon$-soft policy

Repeat forever:
- (a) Generate an episode using $\pi$
- (b) For each pair $s, a$ appearing in the episode:
  - $R \leftarrow$ return following the first occurrence of $s, a$
  - Append $R$ to $\text{Returns}(s, a)$
  - $Q(s, a) \leftarrow$ average$(\text{Returns}(s, a))$
- (c) For each $s$ in the episode:
  - $a^* \leftarrow \arg \max_a Q(s, a)$
  - For all $a \in A(s)$:
    - $\pi(s, a) \leftarrow \begin{cases} 1 - \frac{\epsilon}{|A(s)|} & \text{if } a = a^* \\ \frac{\epsilon}{|A(s)|} & \text{if } a \neq a^* \end{cases}$
Summary of methods

Notes based on (?, ch 4-6).
Convergence results for several MC algorithms are given by (?).

References