State Space Abstractions for Reinforcement Learning

Rowan McAllister and Thang Bui

MLG RCC

06 November 2014
1 Introduction
   - Markov Decision Process
   - Reinforcement Learning
   - State Abstraction

2 Abstraction Types and Properties

3 Aggregation
   - Hard Aggregation
   - Soft Aggregation

4 Basis Functions for Linear Function Approximation

5 Probabilistic State Representations
Introduction

MDPs and Abstract MDPs
A Markov Decision Process is a tuple \( \{S, \mathcal{A}, \mathcal{R}, \mathcal{T}, \gamma\} \)

- \( S \) set of states
- \( \mathcal{A} \) set of actions
- \( \mathcal{R}^a_s : S \times \mathcal{A} \rightarrow \mathbb{R} \) reward function
- \( \mathcal{T}^a_{ss'} = P(s_{t+1} = s' | s_t = s, a_t = a) \) transition function
- \( \gamma \in [0, 1] \) discount factor

**Goal:**
Find policy \( \pi : S \rightarrow \mathcal{A} \) that maximises expected

\[
\sum_{t=0}^{\infty} \gamma^t \mathcal{R}^\pi_{s_t}(s_t)
\]
Policy Evaluation

\[ Q^\pi(s_0, a) \triangleq \mathcal{R}_{s_0}^a + \mathbb{E}_T \left[ \sum_{t=1}^{\infty} \gamma^t \mathcal{R}_s^\pi(s_t) \right] \]

\[ = \mathcal{R}_{s_0}^a + \gamma \mathbb{E}_T \left[ \mathcal{R}_{s_1}^\pi(s_1) + \sum_{t=1}^{\infty} \gamma^t \mathcal{R}_{s_{t+1}}^\pi(s_{t+1}) \right] \]
Policy Evaluation

\[ Q^\pi(s_0, a) \triangleq R_{s_0}^a + \mathbb{E}_T \left[ \sum_{t=1}^{\infty} \gamma^t R_{s_t}^\pi(s_t) \right] \]

\[ = R_{s_0}^a + \gamma \mathbb{E}_T \left[ R_{s_1}^\pi(s_1) + \sum_{t=1}^{\infty} \gamma^t R_{s_{t+1}}^\pi(s_{t+1}) \right] \]

\[ \pi \text{-evaluation:} \]

\[ Q^\pi(s, a) \triangleq R_s^a + \gamma \sum_{s'} \left[ T_{ss'}^a Q^\pi(s', \pi(s')) \right] \]
Policy Evaluation

\[ Q^\pi(s_0, a) \triangleq R_{s_0} + E_T \left[ \sum_{t=1}^{\infty} \gamma^t R_{s_t}^{\pi(s_t)} \right] \]

\[ = R_{s_0} + \gamma E_T \left[ R_{s_1}^{\pi(s_1)} + \sum_{t=1}^{\infty} \gamma^t R_{s_{t+1}}^{\pi(s_{t+1})} \right] \]

\( \pi \)-evaluation:

\[ Q^\pi(s, a) \triangleq R_s + \gamma \sum_{s'} \left[ T_{ss'}^\pi Q^\pi(s', \pi(s')) \right] \]

Bellman equation for optimal value:

\[ Q^*(s, a) \triangleq R_s + \gamma \sum_{s'} \left[ T_{ss'}^a \max_{a'} Q^*(s', a') \right] \]
Policy Evaluation

\[ Q^{\pi}(s_0, a) \triangleq R_{s_0}^a + \mathbb{E}_T \left[ \sum_{t=1}^{\infty} \gamma^t R_{s_t}^{\pi} \right] \]

\[ = R_{s_0}^a + \gamma \mathbb{E}_T \left[ R_{s_1}^{\pi} + \sum_{t=1}^{\infty} \gamma^t R_{s_{t+1}}^{\pi} \right] \]

\pi\text{-evaluation:}

\[ Q^{\pi}(s, a) \triangleq R_s^a + \gamma \sum_{s'} \left[ T_{ss'} a Q^{\pi}(s', \pi(s')) \right] \]

Bellman equation for optimal value:

\[ Q^*(s, a) \triangleq R_s^a + \gamma \sum_{a'} \left[ T_{ss'} a' \max_{a'} Q^*(s', a') \right] \]

\[ V^*(s) \triangleq \max_a Q^*(s, a) = \max_a \left\{ R_s^a + \gamma \sum_{s'} \left[ T_{ss'} a V^*(s') \right] \right\} \]
MDP optimal value example

\[
\begin{array}{c|c|c}
Y=1 & 0 & 3 \\
\hline
0 & 3 & 0 \\
\end{array}
\]

\[
\begin{array}{c|c|c}
Y=0 & 1 & 0 \\
\hline
1 & 1 & 1 \\
\end{array}
\]

\([\text{Rewards}]\)
MDP optimal value example

![Diagram showing transitions and rewards](attachment:image.png)

[Rewards]
[Q: action-val.]
MDP optimal value example

Y=1

Y=0

X=0  X=1

[Rewards]

[Q: action-val.]

[π: Policy]
MDP optimal value example

![Diagram of MDP optimal value example]

- **Rewards**
- **Q**: action-val.
- **π**: Policy
- **V**: state-val.
MDP optimal value example

- Rewards
- $Q$: action-val.
- $\pi$: Policy
- $V$: state-val.

X=0 | X=1
---|---
Y=1 | 3 3 0
Y=0 | 0 1 0

6 / 24
MDP optimal value example

- **Rewards**
- **Q**: action-val.
- **π**: Policy
- **V**: state-val.

The diagram illustrates a Markov Decision Process (MDP) with states and actions. The values indicate the rewards associated with each state-action pair.
MDP optimal value example

![Diagram](image)

- **Rewards**
- **Q**: action-val.
- **π**: Policy
- **V**: state-val.
MDP optimal value example

\[\begin{array}{c|c|c}
Y=1 & 3 & 3 \\
2 & 0 & 3 \\
\hline
Y=0 & 1 & 0 \\
& 1 & 1 \\
\end{array}\]

[Rewards]
[\(Q: \) action-val.]
[\(\pi: \) Policy]
[V: state-val.]
MDP optimal value example

[Y=0]

[X=0]

[X=1]

[Y=1]

[Rewards]

[Q: action-val.] [π: Policy] [V: state-val.]
MDP optimal value example

\begin{align*}
Y=1 & \quad \begin{array}{c|c|c}
X=0 & 3 & 3 \\
Y=0 & 3 & 3 \\
\end{array} \\
& \quad \text{[Rewards]} \\
\end{align*}

\begin{align*}
X=0 & \quad \begin{array}{c|c|c}
Y=1 & 0 & 1 \\
Y=0 & 0 & 1 \\
\end{array} \\
& \quad \text{[Q: action-val.]} \\
& \quad \text{[π: Policy]} \\
& \quad \text{[V: state-val.]} \\
\end{align*}
MDP optimal value example

[Rewards]
[Q: action-val.]
[\(\pi\): Policy]
[V: state-val.]
MDP optimal value example

\[
\begin{array}{c|c|c|c|c}
& X=0 & & X=1 & \\
\hline
Y=1 & 3 & 3 & 3 & 3 \\
\hline
Y=0 & 3 & 2 & 1 & 0 \\
\hline
& X=0 & & X=1 & \\
\end{array}
\]

[Rewards]
[Q: action-val.]
[\(\pi\): Policy]
[V: state-val.]
MDP optimal value example

\[
\begin{array}{cc|cc|cc}
X=0 & X=1 & Y=1 & Y=0 \\
0 & 3 & 3 & 3 \\
1 & 2 & 3 & 3 \\
3 & 2 & 3 & 3 \\
\end{array}
\]

[Rewards] [Q: action-val.] [\(\pi\): Policy] [V: state-val.]
Reinforcement Learning

$\mathcal{R}, \mathcal{T}$ unknown!

Can implement backups with e.g. $Q$-learning:
On experience $\{s_t, a_t, r_{t+1}, s_{t+1}\}$,

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha_t (r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t))$$

new estimate \quad old estimate \quad learning rate \quad Bellman error
Reinforcement learning drawbacks

Reinforcement learning with lookup-tables for values, indexed by states, is not compact enough to scale to real world problems.

No. parameters to learn:

\[ EXP(dim[S]) \]

Solutions?
Reinforcement learning drawbacks

Reinforcement learning with lookup-tables for values, indexed by states, is not compact enough to scale to real world problems.

No. parameters to learn:

\[ EXP(dim[S]) \]

Solutions?

- Function approximation
  (impose some functional form on \( V \), to restrict expressivity.)
Reinforcement learning drawbacks

Reinforcement learning with lookup-tables for values, indexed by states, is not compact enough to scale to real world problems.

No. parameters to learn:

\[ EXP(dim[S]) \]

Solutions?

- Function approximation
  (impose some functional form on \( V \), to restrict expressivity.)
- State abstraction
  (reduce your state space \( S \) before planning in it.)
**State Abstraction**

**Problem:** my states-inputs encode superfluous information!

**Qu:** How do I discern relevant vs. irrelevant info automatically?
State Abstraction: Idea

Idea:

1. Start with ground representation $S$ (relevant + irrelevant info.)
2. Map $S \rightarrow$ abstract-feature space $\mathcal{X}$ (relevant info.)
3. Solve MDP in $\mathcal{X}$-space (easier)
4. Map solution $\mathcal{X}$-space $\rightarrow$ $S$-space
State Abstraction: Idea

Idea:
1. Start with ground representation $S$ (relevant + irrelevant info.)
2. Map $S \rightarrow$ abstract-feature space $X$ (relevant info.)
3. Solve MDP in $X$-space (easier)
4. Map solution $X$-space $\rightarrow S$-space

Example:
- A ‘changing lane’ driving task should not depend on street name.
- Without abstraction: agent might re-learn changing lanes each street.
State Abstraction: Idea

Idea:
1. Start with ground representation $S$ (relevant + irrelevant info.)
2. Map $S \rightarrow$ abstract-feature space $\mathcal{X}$ (relevant info.)
3. Solve MDP in $\mathcal{X}$-space (easier)
4. Map solution $\mathcal{X}$-space $\rightarrow$ $S$-space

Example:
- A ‘changing lane’ driving task should not depend on street name.
- Without abstraction: agent might re-learn changing lanes each street.

Good abstractions:
- Retains task-relevant information, ignores the rest.
- Represent similar situations similarly (discover generalisations)
Abstract MDP

Ground representation: \( S \)
Abstract representation: \( \mathcal{X} \), where \(|\mathcal{X}| << |S|\)
Abstraction function:
\[
\phi : S \rightarrow \mathcal{X} \\
\phi^{-1} : \mathcal{X} \rightarrow S^m
\]

[Li et al., 2006]
Abstract MDP

Ground representation: $S$ [Li et al., 2006]
Abstract representation: $\mathcal{X}$, where $|\mathcal{X}| << |S|$
Abstraction function:
\[
\phi : S \rightarrow \mathcal{X} \\
\phi^{-1} : \mathcal{X} \rightarrow S^m
\]
Cluster probability
\[
P(x|s) = \mathbb{1}[s \in \phi^{-1}(x)]
\]
Abstract MDP

Ground representation: \( S \)  
Abstract representation: \( \mathcal{X} \), where \(|\mathcal{X}| < < |S|\)  
Abstraction function: \( \phi: S \rightarrow \mathcal{X} \)  
\( \phi^{-1}: \mathcal{X} \rightarrow S^m \)  
Cluster probability  
\( P(x|s) = \mathbb{1}[s \in \phi^{-1}(x)] \)  
Abstract solution:  
\( Q(x, a) = R_x^a + \gamma \sum_{x'} \left[ T_{xx'}^a \max_{a'} Q(x', a') \right] \)
Abstract MDP

Ground representation: $S$  
Abstract representation: $\mathcal{X}$, where $|\mathcal{X}| << |S|$  
Abstraction function: $\phi : S \rightarrow \mathcal{X}$, $\phi^{-1} : \mathcal{X} \rightarrow S^m$  
Cluster probability $P(x|s) = 1[s \in \phi^{-1}(x)]$  
Abstract solution: $Q(x, a) = R_x^a + \gamma \sum_{x'} [T_{xx'}^a \max_{a'} Q(x', a')]$  
Ground solution: $Q(s, a) = \mathbb{E}_{x|s}[Q(x, a)]$  

[Li et al., 2006]
Abstract MDP (soft)

Ground representation: $S$  
Abstract representation: $\mathcal{X}$, where $|\mathcal{X}| << |S|$  
Abstraction function: $\phi : S \rightarrow \mathcal{X}^n$  
$\phi^{-1} : \mathcal{X} \rightarrow S^m$  
Cluster probability $P(x|s) \in [0, 1]$  
Abstract solution: $Q(x, a) = R_x^a + \gamma \sum_{x'} \left[ T_{xx'}^a \max_{a'} Q(x', a') \right]$  
Ground solution: $Q(s, a) = \mathbb{E}_{x|s}[Q(x, a)]$  

[Li et al., 2006]
Abstract MDP (soft)

Ground representation: $S$

Abstract representation: $\mathcal{X}$, where $|\mathcal{X}| << |S|$

Abstraction function: $\phi: S \rightarrow \mathcal{X}^n$

$\phi^{-1}: \mathcal{X} \rightarrow S^m$

Cluster probability $P(x|s) \in [0, 1]$

Abstract solution: $Q(x, a) = R^a_x + \gamma \sum_{x'} \left[ T^a_{xx'} \max_{a'} Q(x', a') \right]$

Ground solution: $Q(s, a) = \mathbb{E}_{x|s}[Q(x, a)]$

Relation [Singh et al., 1995]:

$$P(s|x) = \frac{P(x|s)P^\pi(s)}{P^\pi(x)}$$

$R^a_x = \sum_s \left[ P(s|x)R^a_s \right] = \mathbb{E}_{s|x}[R^a_s]$}

$T^a_{xx'} = \sum_{s,s'} \left[ P(s|x)P(x'|s') T^a_{ss'} \right] $
Abstractions:
Types and Properties
Abstraction Objective

Bellman error:

\[ J(s) = V(s) - \max_a \left\{ R_s^a + \gamma \sum_{s'} [T_{ss'}^a, V(s')] \right\} \]

Minimise: \( J^2(s) \)
Abstraction Objective

Bellman error:

$$J(s) = V(s) - \max_a \left\{ R_s^a + \gamma \sum_{s'} T_{ss'}^a V(s') \right\}$$

Minimise: $J^2(s)$

Why?

- True value $V^\pi$ not available for comparison.
- Approximation error, $V^\pi - V$, is bounded by a constant multiple of the Bellman error norm [Bertsekas et al., 1995].
Aggregation
Adaptive Partitioning


A Voronoi (or nearest-neighbour) quantiser maps states $s \in \mathbb{R}^n$ to a finite set of centroids $\{x_1, ..., x_m\}$ each also in $\mathbb{R}^n$.

$$\phi(s) = \{x_i : \|s - x_i\| \leq \|s - x_j\|, \forall i \neq j\}$$

Desire:

- aggregate states of similar $Q^*$-values (Class $\phi_{Q^*}$-ish abstraction)
- number of cells grows (or shrink) as needed
Adaptive Partitioning: Experiments

1000 trials.

1200 trials.

- **Aggregation**
- **Hard Aggregation**

- **START**
- **GOAL**

- **Codebook vector position and its number**
- **Sample trajectory**
- **Voronoi cell boundary**
- **Barrier or obstacle**

- **Highest-valued action within each Voronoi cell**
- **Failed region**
Adaptive Partitioning: Algorithm (approx.)

\[
\begin{align*}
\text{init } & s_t, a_t \\
\text{init } & \mathcal{X} \leftarrow \{s_t\} \\
\text{REPEAT:} & \\
& \bullet \text{ execute } a_t \\
& \bullet a_{t+1} \leftarrow \epsilon\text{-greedy policy } Q(x_{t+1}) \\
& \bullet \text{ IF } x_{t+1} == x_t \\
& & \quad \bullet \text{ accReward} + = r_{t+1} \\
& & \quad \bullet \text{ IF } \text{ accReward} > \text{ threshold} \\
& & \quad \quad \star \mathcal{X} \leftarrow \{\mathcal{X}, s_{t+1}\} \\
& \bullet \text{ ELSE} \\
& & \bullet \text{ accReward} \leftarrow 0 \\
& \bullet \text{ IF any neighbouring cells } x_i, x_j \text{ have similar } Qs \\
& & \quad \bullet \text{ remove } x_i \text{ and } x_j \text{ from } \mathcal{X} \text{ and insert } \frac{1}{2}(x_i + x_j) \\
& \bullet \text{ backup } Q \\
& \bullet t \leftarrow t + 1 
\end{align*}
\]
Adaptive soft state aggregation

Adaptive State Aggregation [Singh et al., 1995]:

1. Compute $V(x|\theta)$ for all $x \in X$,
2. Compute cluster probabilities: $P(x|s; \theta) = \frac{\exp[\theta(x,s)]}{\sum_{x'} \exp[\theta(x',s)]}$
3. Compute Bellman error + derivatives:

$$J(s|\theta) = V(s|\theta) - [R_s + \gamma \sum_{s'} T_{ss'} V(s'|\theta)]$$

$$= \mathbb{E}_{x|s;\theta}[V(x|\theta)] - [R_s + \gamma \sum_{s'} T_{ss'} \mathbb{E}_{x|s;\theta}[V(x'|\theta)]]$$

$$\frac{\partial J^2(\theta)}{\partial \theta(x, s)} = 2J(s|\theta)P(x|s; \theta)(1 - \gamma T_{ss})(V(x|\theta) - V(s, \theta))$$

4. $\theta \leftarrow \theta - \alpha \frac{\partial J^2(\theta)}{\partial \theta}$
Adaptive soft state aggregation: Results

The graph shows the summed squared Bellman error over iterations of ASA for different cluster sizes. The clusters are:

- 2 Clusters
- 4 Clusters
- 10 Clusters
- 20 Clusters

The error decreases with each iteration, and the rate of decrease is faster for smaller cluster sizes.
Basis Functions for Linear Function Approximation
Idea

Idea: abstract $S$ and approximate $V(\cdot)$

Want: easy-to-train $V(\cdot)$ form, a linear function

Method:
1. compute approx $V(x) = \beta^T x$ (use LSTD)
2. compute Bellman errors $J(s)$
3. learn abstract mapping $x = \phi_\theta(s)$ parameterised by $\theta$
   (use Bellman error to guide)
4. (optional) aggregate $x$

Choices for $\phi$ functional form?

- Neighbourhood Component Analysis (NCA) [Keller et al., 2006]
- Radial Basis Functions (RBF) [Menache et al., 2005]
Neighbourhood Component Analysis

**Idea:** learn a distance metric that optimises nearest-neighbour leave-one-out performance.

\[
D(s_i, s_j) = \begin{pmatrix} s_i - s_j \end{pmatrix}^\top \Sigma^{-1} \begin{pmatrix} s_i - s_j \end{pmatrix},
\]

\[
= \begin{pmatrix} s_i - s_j \end{pmatrix}^\top A^\top A(s_i - s_j),
\]

\[
= \begin{pmatrix} As_i - As_j \end{pmatrix}^\top (As_i - As_j),
\]

\[
= \begin{pmatrix} x_i - x_j \end{pmatrix}^\top (x_i - x_j),
\]

where \( s \in \mathbb{R}^n, A \in \mathbb{R}^{d \times n}, d << n. \)

\[
\text{class}(s_i) = \text{class}(s_j) \text{ with probability } p_{ij} = \frac{\exp(-||As_i - As_j||)}{\sum_{k \neq i} \exp(-||As_i - As_k||)}
\]

Objective of [Keller et al., 2006] for gradient decent:

\[
f(A) = \sum_{ij} p_{ij} [J(s_i) - J(s_j)]
\]
Least-Squares Temporal Difference learning (LSTD)

**Goal:** Find $\beta$ that mean squared (projected) Bellman error$^1$ of linear approximation $V_\pi(x) \approx \beta^T x$. [Boyan, 1999]

We observe $\{x_0, r_0, ..., x_t, r_t\}$.

For each $\tau \in [0, t]$, can generate Bellman errors $j_\tau = r_{\tau+1} + (\gamma x_{\tau+1} - x_\tau)^T \beta$

TD update: $\beta \leftarrow \beta + \alpha z_\tau j_\tau$

$$
\begin{align*}
b_t &= \sum_{\tau=0}^{t} z_\tau r_\tau \\
A_t &= \sum_{\tau=0}^{t} z_\tau (x_\tau - \gamma x_{\tau+1})^T \\
z_t &= \sum_{\tau=0}^{t} \gamma^{t-\tau} x_\tau \\
\beta_t &= A_t^{-1} b_t
\end{align*}
$$

$^1$Good survey of alternatives to MSPBE is [Dann et al., 2014]
Probabilistic State Representations
*Dynamic programming and optimal control*, volume 1.
Athena Scientific Belmont, MA.

Least-squares temporal difference learning.


Automatic basis function construction for approximate dynamic programming and reinforcement learning.

Adaptive state space partitioning for reinforcement learning.  
*Engineering applications of artificial intelligence, 17*(6):577–588.

Towards a unified theory of state abstraction for mdps.  
In *ISAIM*.

Basis function adaptation in temporal difference reinforcement learning.  

Reinforcement learning with soft state aggregation.  
Radial Basis Functions

\[ \phi_{\theta_i}(s) = \exp \left( -\frac{1}{2} (s - c_i)^\top W_i^{-1} (s - c_i) \right) \]