Learning and Planning in POMDP with Weighted Automata

Guillaume Rabusseau
Assistant Professor at DIRO, UdeM
CIFAR Canada Chair in AI at Mila

May 4, 2021
Theoretically Inclined Machine Learning Seminar

Outline

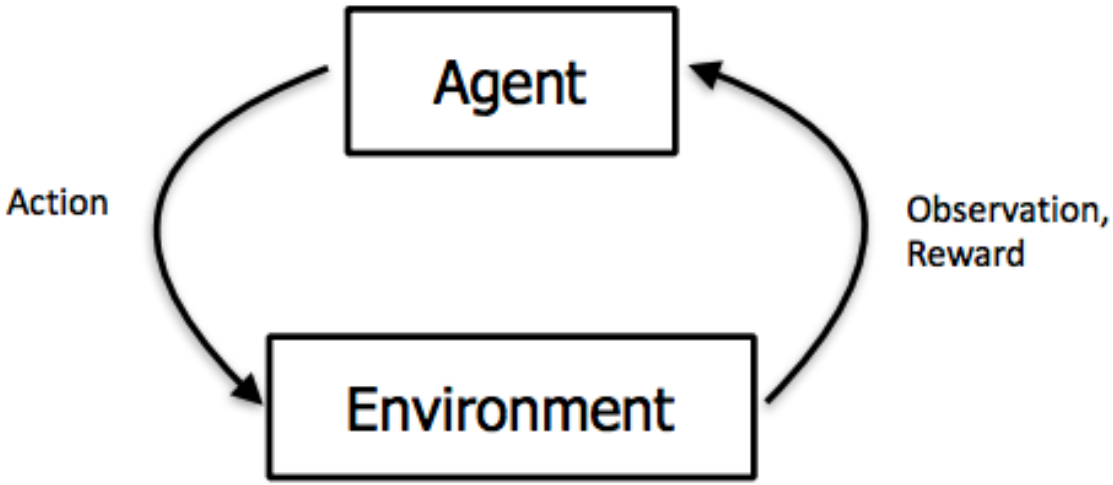
What I will talk about today is based on joint work with Tianyu Li, Bogdan Mazoure and Doina Precup:



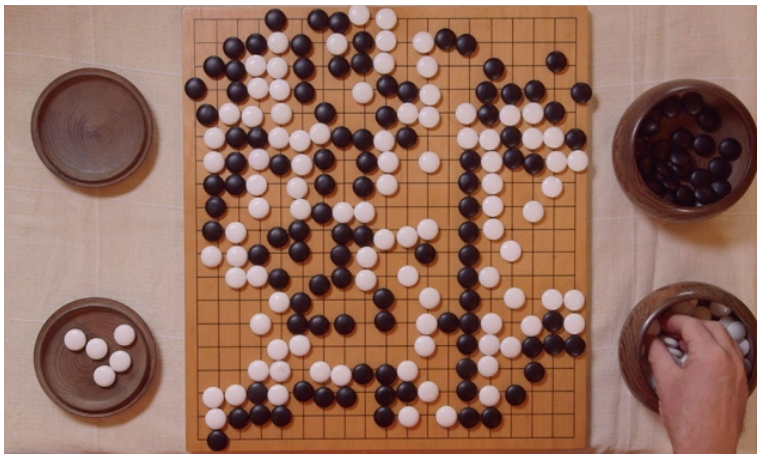
- Li, T., Mazoure, B., Precup, D., Rabusseau, G. (2020, June). *Efficient Planning under Partial Observability with Unnormalized Q Functions and Spectral Learning*. In AISTATS 2020.

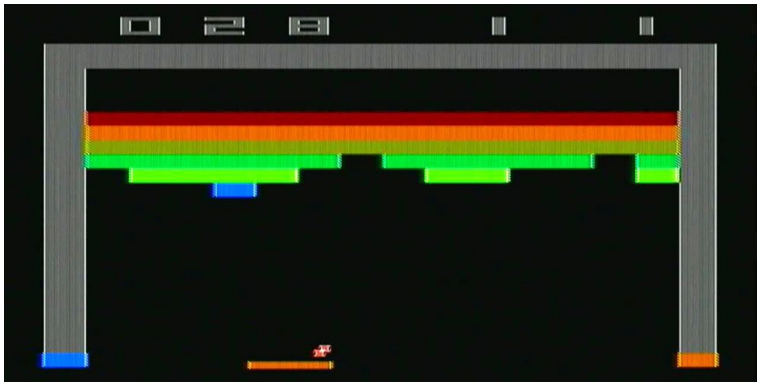
Introduction

Reinforcement Learning

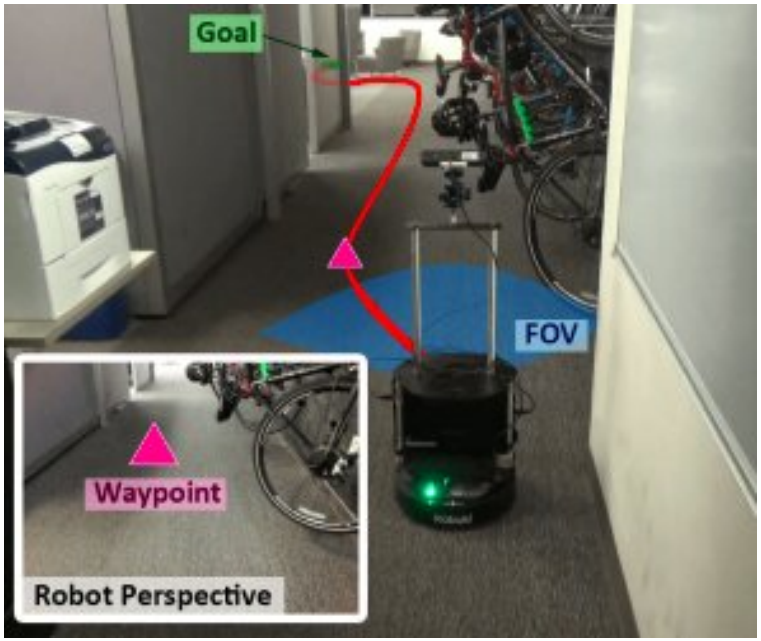


Fully Observable Environment

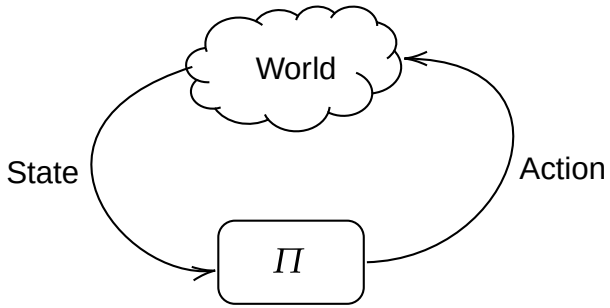




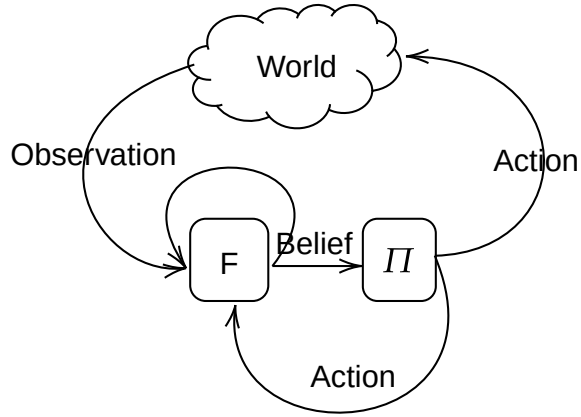
Partially Observable Environment



General Approaches



Fully observable environment



Partially observable environment

General Approach for Partially Observable Environment

1. Learning:
   - Learn the function F to estimate the belief states given a history sequence h.
   - Learn a reward model (mapping belief states to rewards / Q-value)
2. Planning: Policy acting optimally w.r.t. belief states estimator & reward model

General Approach for Partially Observable Environment

1. Learning:
   - Learn the function F to estimate the belief states given a history sequence h.
   - Learn a reward model (mapping belief states to rewards / Q-value)
2. Planning: Policy acting optimally w.r.t. belief states estimator & reward model

Examples:

- The family of predictive state representations (PSRs) [(Littman and Sutton, 2002; Singh et al., 2003, 2004; Boots et al., 2011)]
  - Separate the reward information from the belief estimation
  - Inefficient state representation for planning
  - Potentially sample inefficient and time consuming

General Approach for Partially Observable Environment

1. Learning:
   - Learn the function F to estimate the belief states given a history sequence h.
   - Learn a reward model (mapping belief states to rewards / Q-value)
2. Planning: Policy acting optimally w.r.t. belief states estimator & reward model

Examples:

- The family of predictive state representations (PSRs) [(Littman and Sutton, 2002; Singh et al., 2003, 2004; Boots et al., 2011)]
  - Separate the reward information from the belief estimation
  - Inefficient state representation for planning
  - Potentially sample inefficient and time consuming
- Deep Q-Learning [(Mnih et al., 2013; Hausknecht and Stone, 2015)]
  - No theoretical guarantees
  - Partial observability is often ignored (e.g. state = last 10 observations)
  - Sample inefficient

Our Contribution

- Propose an algorithm that incorporate reward information into the belief estimation
- Sample efficient
- Consistent algorithm (spectral learning)

Background

Background

Partially Observable Markov Decision Processes

Partially Observable Markov Decision Processes (POMDPs)

A POMDP is a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{O}, \mathbf{P}, \mathbf{R}, \mathbf{O}, \mathbf{b}_0 \rangle$ where

- $\mathcal{S}$: set of states (finite);
- $\mathcal{A}$: set of actions (finite);
- $\mathcal{O}$: set of observations (finite);

Partially Observable Markov Decision Processes (POMDPs)

A POMDP is a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{O}, \mathbf{P}, \mathbf{R}, \mathbf{O}, \mathbf{b}_0 \rangle$ where

- $\mathcal{S}$: set of states (finite);
- $\mathcal{A}$: set of actions (finite);
- $\mathcal{O}$: set of observations (finite);
- $\mathbf{P} \in \mathbb{R}^{\mathcal{S} \times \mathcal{A} \times \mathcal{S}}$: Transition tensor $\mathbf{P}_{sas'} = \mathbb{P}(S_{t+1} = s' | S_t = s, A_t = a)$;

Partially Observable Markov Decision Processes (POMDPs)

A POMDP is a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{O}, \mathbf{P}, \mathbf{R}, \mathbf{O}, \mathbf{b}_0 \rangle$ where

- $\mathcal{S}$: set of states (finite);
- $\mathcal{A}$: set of actions (finite);
- $\mathcal{O}$: set of observations (finite);
- $\mathbf{P} \in \mathbb{R}^{\mathcal{S} \times \mathcal{A} \times \mathcal{S}}$: Transition tensor $\mathbf{P}_{sas'} = \mathbb{P}(S_{t+1} = s' | S_t = s, A_t = a)$;
- $\mathbf{R} \in \mathbb{R}^{\mathcal{S} \times \mathcal{A}}$: Reward matrix $\mathbf{R}_{sa} = R(s, a)$;

Partially Observable Markov Decision Processes (POMDPs)

A POMDP is a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{O}, \mathbf{P}, \mathbf{R}, \mathbf{O}, \mathbf{b}_0 \rangle$ where

- $\mathcal{S}$: set of states (finite);
- $\mathcal{A}$: set of actions (finite);
- $\mathcal{O}$: set of observations (finite);
- $\mathbf{P} \in \mathbb{R}^{\mathcal{S} \times \mathcal{A} \times \mathcal{S}}$: Transition tensor $\mathbf{P}_{sas'} = \mathbb{P}(S_{t+1} = s' | S_t = s, A_t = a)$;
- $\mathbf{R} \in \mathbb{R}^{\mathcal{S} \times \mathcal{A}}$: Reward matrix $\mathbf{R}_{sa} = R(s, a)$;
- $\mathbf{O} \in \mathbb{R}^{\mathcal{S} \times \mathcal{A} \times \mathcal{O}}$: Emission tensor $\mathbf{O}_{sao} = \mathbb{P}(O_t = o | S_t = s, A_t = a)$;

Partially Observable Markov Decision Processes (POMDPs)

A POMDP is a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{O}, \mathbf{P}, \mathbf{R}, \mathbf{O}, \mathbf{b}_0 \rangle$ where

- $\mathcal{S}$: set of states (finite);
- $\mathcal{A}$: set of actions (finite);
- $\mathcal{O}$: set of observations (finite);
- $\mathbf{P} \in \mathbb{R}^{\mathcal{S} \times \mathcal{A} \times \mathcal{S}}$: Transition tensor $\mathbf{P}_{sas'} = \mathbb{P}(S_{t+1} = s' | S_t = s, A_t = a)$;
- $\mathbf{R} \in \mathbb{R}^{\mathcal{S} \times \mathcal{A}}$: Reward matrix $\mathbf{R}_{sa} = R(s, a)$;
- $\mathbf{O} \in \mathbb{R}^{\mathcal{S} \times \mathcal{A} \times \mathcal{O}}$: Emission tensor $\mathbf{O}_{sao} = \mathbb{P}(O_t = o | S_t = s, A_t = a)$;
- $\mathbf{b}_0 \in \mathbb{R}^{\mathcal{S}}$: Initial state distribution.

Partially Observable Markov Decision Processes (POMDPs)

A POMDP is a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{O}, \mathbf{P}, \mathbf{R}, \mathbf{O}, \mathbf{b}_0 \rangle$ where

- $\mathcal{S}$: set of states (finite);
- $\mathcal{A}$: set of actions (finite);
- $\mathcal{O}$: set of observations (finite);
- $\mathbf{P} \in \mathbb{R}^{\mathcal{S} \times \mathcal{A} \times \mathcal{S}}$: Transition tensor $\mathbf{P}_{sas'} = \mathbb{P}(S_{t+1} = s' | S_t = s, A_t = a)$;
- $\mathbf{R} \in \mathbb{R}^{\mathcal{S} \times \mathcal{A}}$: Reward matrix $\mathbf{R}_{sa} = R(s, a)$;
- $\mathbf{O} \in \mathbb{R}^{\mathcal{S} \times \mathcal{A} \times \mathcal{O}}$: Emission tensor $\mathbf{O}_{sao} = \mathbb{P}(O_t = o | S_t = s, A_t = a)$;
- $\mathbf{b}_0 \in \mathbb{R}^{\mathcal{S}}$: Initial state distribution.

Goal

Find a policy $\pi : \mathcal{H} \to \mathcal{A}$ maximizing the expected discounted return

$$V(\pi) = \mathbb{E}\left[\sum_{t \geq 0} \gamma^t R(s_t, a_t) \middle| \pi, \mathbf{b}_0 \right]$$

where $\mathcal{H} = (\mathcal{A} \times \mathcal{O})^*$ is the set of all histories and γ is a discount factor.

Background

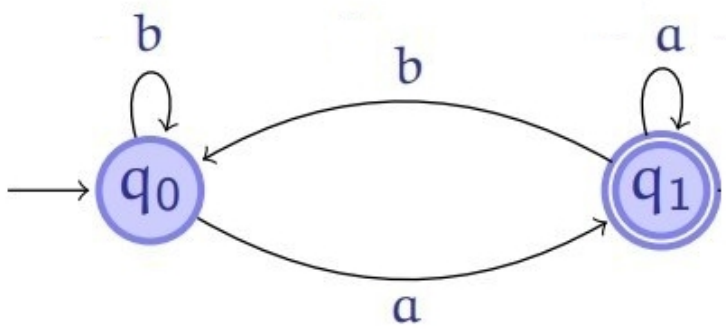
Weighted Finite Automata

String Weighted Automata (WA)

- Notations:
  - Σ: a finite alphabet (e.g. $\{a, b\}$)
  - Σ^*: strings on Σ (e.g. $abba$, ab, λ, ...)
  - λ: the empty string.

String Weighted Automata (WA)

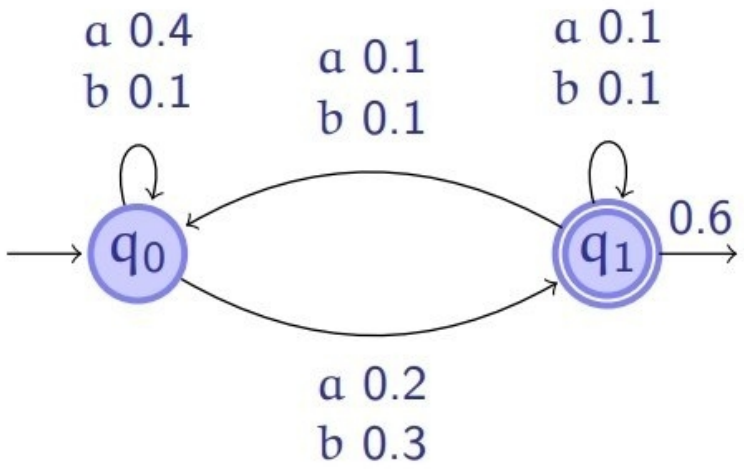
- Notations:
  - Σ: a finite alphabet (e.g. $\{a, b\}$)
  - Σ^*: strings on Σ (e.g. $abba$, ab, λ, ...)
  - λ: the empty string.
- Recall: a Deterministic Finite Automaton (DFA) recognizes a *language* (subset of Σ^*).



$\hookrightarrow$ a DFA computes a function $f : \Sigma^* \to \{\top, \bot\}$.

Weighted Automata: States and Weighted Transitions

Example with 2 states and alphabet $\Sigma = \{a, b\}$

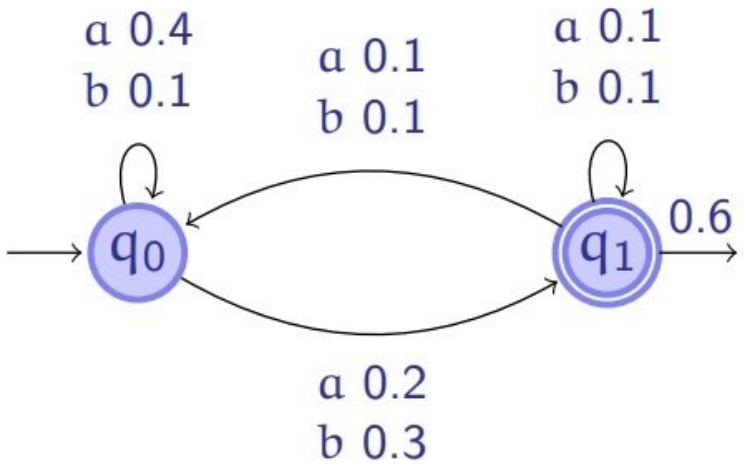


$$f(ab) = 0.4 \times 0.3 \times 0.6 + 0.2 \times 0.1 \times 0.6 = 0.084$$

image credits: B. Balle, X. Carreras, A. Quattoni - ENMLP'14 tutorial

Weighted Automata: States and Weighted Transitions

Example with 2 states and alphabet $\Sigma = \{a, b\}$



Operator Representation

$$\boldsymbol{\alpha} = \begin{bmatrix} 1.0 \\ 0.0 \end{bmatrix} \quad \mathbf{A}^{a} = \begin{bmatrix} 0.4 & 0.2 \\ 0.1 & 0.1 \end{bmatrix}$$

$$\boldsymbol{\omega} = \begin{bmatrix} 0.0 \\ 0.6 \end{bmatrix} \quad \mathbf{A}^{b} = \begin{bmatrix} 0.1 & 0.3 \\ 0.1 & 0.1 \end{bmatrix}$$

$$f(ab) = 0.4 \times 0.3 \times 0.6 + 0.2 \times 0.1 \times 0.6 = 0.084$$

$$= \boldsymbol{\alpha}^{\top} \mathbf{A}^{a} \mathbf{A}^{b} \boldsymbol{\omega}$$

slide credits: B. Balle, X. Carreras, A. Quattoni - ENMLP'14 tutorial

String Weighted Automata (WA)

- Σ a finite alphabet (e.g. $\{a, b\}$), Σ^* strings on Σ (e.g. $abba$)
- A WA computes a function $f : \Sigma^* \to \mathbb{R}$

String Weighted Automata (WA)

- Σ a finite alphabet (e.g. $\{a, b\}$), Σ^* strings on Σ (e.g. $abba$)
- A WA computes a function $f : \Sigma^* \to \mathbb{R}$
- Weighted Automaton: $A = (\boldsymbol{\alpha}, \{\mathbf{A}^\sigma\}_{\sigma \in \Sigma}, \boldsymbol{\omega})$ where

$\boldsymbol{\alpha} \in \mathbb{R}^n$ initial weights vector
$\boldsymbol{\omega} \in \mathbb{R}^n$ final weights vector
$\mathbf{A}^\sigma \in \mathbb{R}^{n \times n}$ transition weights matrix for each $\sigma \in \Sigma$

String Weighted Automata (WA)

- Σ a finite alphabet (e.g. $\{a, b\}$), Σ^* strings on Σ (e.g. $abba$)
- A WA computes a function $f : \Sigma^* \to \mathbb{R}$
- Weighted Automaton: $A = (\boldsymbol{\alpha}, \{\mathbf{A}^\sigma\}_{\sigma \in \Sigma}, \boldsymbol{\omega})$ where

$$\begin{aligned} \boldsymbol{\alpha} &\in \mathbb{R}^n && \text{initial weights vector} \\ \boldsymbol{\omega} &\in \mathbb{R}^n && \text{final weights vector} \\ \mathbf{A}^\sigma &\in \mathbb{R}^{n \times n} && \text{transition weights matrix for each } \sigma \in \Sigma \end{aligned}$$

- A computes a function $f_A : \Sigma^* \to \mathbb{R}$ defined by

$$f_A(\sigma_1 \sigma_2 \cdots \sigma_k) = \boldsymbol{\alpha}^\top \mathbf{A}^{\sigma_1} \mathbf{A}^{\sigma_2} \cdots \mathbf{A}^{\sigma_k} \boldsymbol{\omega} = \boldsymbol{\alpha}^\top \mathbf{A}^{\sigma_1 \sigma_2 \cdots \sigma_k} \boldsymbol{\omega}$$

String Weighted Automata (WA)

- Σ a finite alphabet (e.g. $\{a, b\}$), Σ^* strings on Σ (e.g. $abba$)
- A WA computes a function $f : \Sigma^* \to \mathbb{R}$
- Weighted Automaton: $A = (\boldsymbol{\alpha}, \{\mathbf{A}^\sigma\}_{\sigma \in \Sigma}, \boldsymbol{\omega})$ where

$$\begin{aligned} \boldsymbol{\alpha} &\in \mathbb{R}^n && \text{initial weights vector} \\ \boldsymbol{\omega} &\in \mathbb{R}^n && \text{final weights vector} \\ \mathbf{A}^\sigma &\in \mathbb{R}^{n \times n} && \text{transition weights matrix for each } \sigma \in \Sigma \end{aligned}$$

- A computes a function $f_A : \Sigma^* \to \mathbb{R}$ defined by

$$f_A(\sigma_1 \sigma_2 \cdots \sigma_k) = \boldsymbol{\alpha}^\top \mathbf{A}^{\sigma_1} \mathbf{A}^{\sigma_2} \cdots \mathbf{A}^{\sigma_k} \boldsymbol{\omega} = \boldsymbol{\alpha}^\top \mathbf{A}^{\sigma_1 \sigma_2 \cdots \sigma_k} \boldsymbol{\omega}$$

Theorem ((Singh et al., 2004))

For any POMDP, there exists a WFA A computing the trajectory probabilities. I.e., such that $f_A(h) = \mathbb{P}(h)$ for all $h \in \mathcal{H}$, where $\mathcal{H} = (\mathcal{A} \times \mathcal{O})^*$ is the set of all histories.

Background

Spectral Learning of WFAs

Hankel matrix

- Σ a finite alphabet of size d (e.g. $\{a, b\}$)
- Σ^* strings on Σ (e.g. *abba*)
- A WFA computes a function $f : \Sigma^* \to \mathbb{R}$:

$$f(\sigma_1 \cdots \sigma_k) = \boldsymbol{\alpha}^\top \mathbf{A}^{\sigma_1} \mathbf{A}^{\sigma_2} \cdots \mathbf{A}^{\sigma_k} \boldsymbol{\omega}$$

Hankel matrix

- Σ a finite alphabet of size d (e.g. $\{a, b\}$)
- Σ^* strings on Σ (e.g. $abba$)
- A WFA computes a function $f : \Sigma^* \to \mathbb{R}$:

$$f(\sigma_1 \cdots \sigma_k) = \boldsymbol{\alpha}^\top \mathbf{A}^{\sigma_1} \mathbf{A}^{\sigma_2} \cdots \mathbf{A}^{\sigma_k} \boldsymbol{\omega}$$

- $\mathbf{H}_f \in \mathbb{R}^{\Sigma^* \times \Sigma^*}$: Hankel matrix of $f : \Sigma^* \to \mathbb{R}$
  - *Definition*: prefix p, suffix $s \Rightarrow (\mathbf{H}_f)_{p,s} = f(ps)$

$$\begin{array}{c} \\ \lambda \\ a \\ b \\ aa \\ ab \\ \vdots \end{array} \begin{array}{c} \begin{array}{ccccccc} \lambda & a & b & aa & ab & bb & \dots \end{array} \\ \left[\begin{array}{ccccccc} f(\lambda) & f(a) & f(b) & f(aa) & f(ab) & \dots & \dots \\ f(a) & f(aa) & f(ab) & f(aaa) & f(aab) & \dots & \dots \\ f(b) & f(ba) & f(bb) & f(baa) & f(bab) & \dots & \dots \\ f(aa) & f(aaa) & f(aab) & f(aaaa) & f(aaab) & \dots & \dots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots \end{array}\right] \end{array}$$

Hankel matrix

- We consider the case where inputs are sequences of discrete symbols:
  - Σ a finite alphabet of size d (e.g. $\{a, b\}$)
  - Σ^* strings on Σ (e.g. $abba$)
  - A WFA computes a function $f : \Sigma^* \to \mathbb{R}$:

$$f(\sigma_1 \cdots \sigma_k) = \boldsymbol{\alpha}^\top \mathbf{A}^{\sigma_1} \mathbf{A}^{\sigma_2} \cdots \mathbf{A}^{\sigma_k} \boldsymbol{\omega}$$

- $\mathbf{H}_f \in \mathbb{R}^{\Sigma^* \times \Sigma^*}$: Hankel matrix of $f : \Sigma^* \to \mathbb{R}$
  - *Definition*: prefix p, suffix $s \Rightarrow (\mathbf{H}_f)_{p,s} = f(ps)$

$$\begin{array}{c} \\ \lambda \\ a \\ b \\ aa \\ ab \\ \vdots \end{array} \begin{array}{c} \begin{array}{ccccccc} \lambda & a & b & aa & ab & bb & \dots \end{array} \\ \begin{bmatrix} f(\lambda) & f(a) & f(b) & f(aa) & f(ab) & \dots & \dots \\ f(a) & f(aa) & f(ab) & f(aaa) & f(aab) & \dots & \dots \\ f(b) & f(ba) & f(bb) & f(baa) & f(bab) & \dots & \dots \\ f(aa) & f(aaa) & f(aab) & f(aaaa) & f(aaab) & \dots & \dots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots \end{bmatrix} \end{array}$$

Spectral Learning of WFA

- $\mathbf{H}_f \in \mathbb{R}^{\Sigma^* \times \Sigma^*}$: Hankel matrix of $f : \Sigma^* \to \mathbb{R}$

  Definition: prefix p, suffix $s \Rightarrow (\mathbf{H}_f)_{p,s} = f(ps)$

- Fundamental theorem [Carlyle and Paz, 1971; Fliess 1974]:

  $\text{rank}(\mathbf{H}_f) < \infty \iff f$ can be computed by a WFA

Spectral Learning of WFA

- $\mathbf{H}_f \in \mathbb{R}^{\Sigma^* \times \Sigma^*}$: Hankel matrix of $f : \Sigma^* \to \mathbb{R}$

  Definition: prefix p, suffix $s \Rightarrow (\mathbf{H}_f)_{p,s} = f(ps)$

- Fundamental theorem [Carlyle and Paz, 1971; Fliess 1974]:

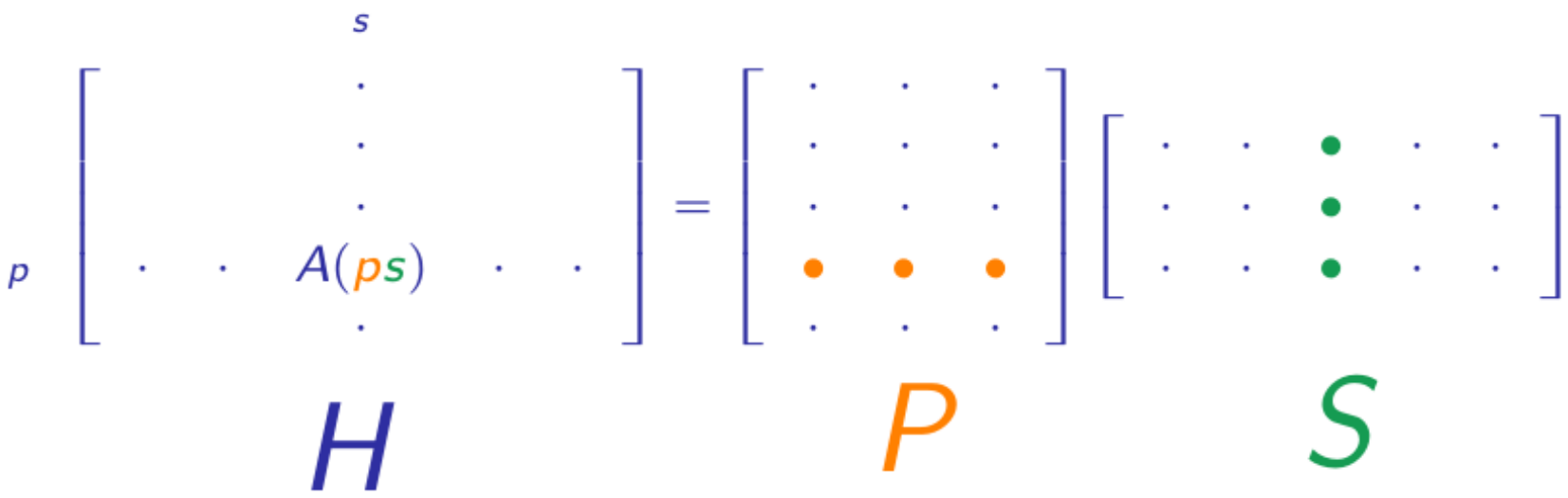
  $\text{rank}(\mathbf{H}_f) < \infty \iff f$ can be computed by a WFA

$\hookrightarrow$ Proof is constructive! From a low rank factorization of $\mathbf{H}_f$ we can recover a WFA computing f...

Hankel matrix and WA

Theorem (Fliess '74)
The rank of a *real* Hankel matrix H equals the minimal number of states of a WFA recognizing the weighted language of H

$$A(p_1 \cdots p_t s_1 \cdots s_{t'}) = \alpha^\top A_{p_1} \cdots A_{p_t} A_{s_1} \cdots A_{s_{t'}} \beta$$



slide credits: B. Balle

Hankel matrices

In addition to the Hankel matrix

$$\mathbf{H} = \begin{array}{c} \\ a \\ b \\ aa \\ ab \\ \vdots \end{array} \begin{array}{c} \begin{array}{ccccc} a & b & aa & ab & \dots \end{array} \\ \begin{bmatrix} f(aa) & f(ab) & \dots & \dots & \dots \\ f(ba) & f(bb) & \dots & \dots & \dots \\ f(aaa) & f(aab) & \dots & \dots & \dots \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{bmatrix} \end{array}$$

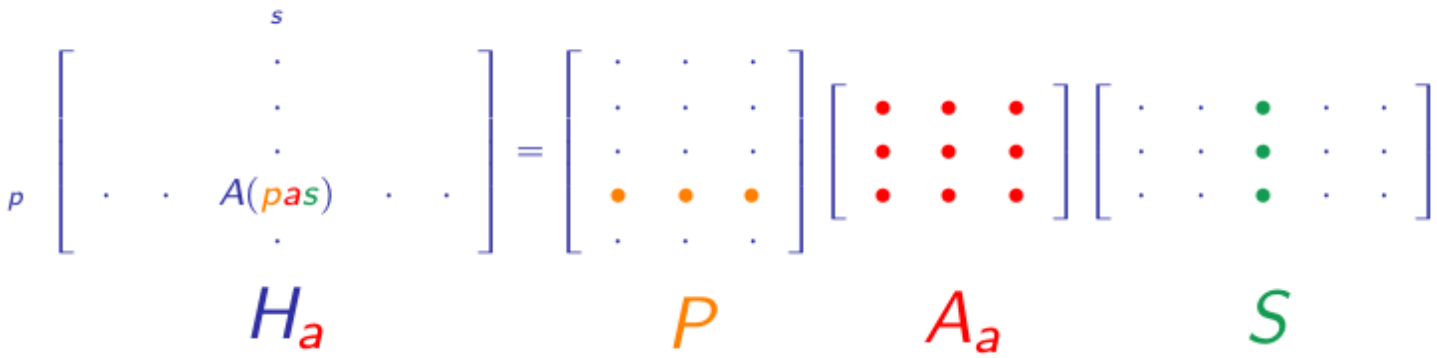
For each $\sigma \in \Sigma$ we also define

$$\mathbf{H}_\sigma = \begin{array}{c} \\ a \\ b \\ aa \\ ab \\ \vdots \end{array} \begin{array}{c} \begin{array}{ccccc} a & b & aa & ab & \dots \end{array} \\ \begin{bmatrix} f(a\sigma a) & f(a\sigma b) & \dots & \dots & \dots \\ f(b\sigma a) & f(b\sigma b) & \dots & \dots & \dots \\ f(aa\sigma a) & f(aa\sigma b) & \dots & \dots & \dots \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{bmatrix} \end{array}$$

Hankel matrix: spectral learning

$$H_a(p, s) = A(pas)$$

$$A(p_1 \cdots p_t a s_1 \cdots s_{t'}) = \alpha^\top A_{p_1} \cdots A_{p_t} A_a A_{s_1} \cdots A_{s_{t'}} \beta$$



$$H = P\ S \qquad H_a = P\ A_a\ S \qquad A_a = P^+\ H_a\ S^+$$

slide credits: B. Balle

Spectral Learning of Weighted Automata (in a nutshell)

1. Choose a set of prefixes and suffixes, $\mathcal{P}, \mathcal{S} \subset \Sigma^*$.
2. Estimate the Hankel sub-blocks $\mathbf{H}$ and $\mathbf{H}_\sigma \in \mathbb{R}^{\mathcal{P} \times \mathcal{S}}$ for each $\sigma \in \Sigma$

$$\mathbf{H} = \begin{array}{c} \\ a \\ b \\ aa \end{array} \begin{array}{c} \begin{array}{cc} a & b \end{array} \\ \begin{bmatrix} f(aa) & f(ab) \\ f(ba) & f(bb) \\ f(aaa) & f(aab) \end{bmatrix} \end{array} \qquad \mathbf{H}_\sigma = \begin{array}{c} \\ a \\ b \\ aa \end{array} \begin{array}{c} \begin{array}{cc} a & b \end{array} \\ \begin{bmatrix} f(a\sigma a) & f(a\sigma b) \\ f(b\sigma a) & f(b\sigma b) \\ f(aa\sigma a) & f(aa\sigma b) \end{bmatrix} \end{array}$$

Spectral Learning of Weighted Automata (in a nutshell)

1. Choose a set of prefixes and suffixes, $\mathcal{P}, \mathcal{S} \subset \Sigma^*$.
2. Estimate the Hankel sub-blocks $\mathbf{H}$ and $\mathbf{H}_\sigma \in \mathbb{R}^{\mathcal{P} \times \mathcal{S}}$ for each $\sigma \in \Sigma$

$$\mathbf{H} = \begin{array}{c} \\ a \\ b \\ aa \end{array} \begin{array}{c} \begin{array}{cc} a & b \end{array} \\ \begin{bmatrix} f(aa) & f(ab) \\ f(ba) & f(bb) \\ f(aaa) & f(aab) \end{bmatrix} \end{array} \quad \mathbf{H}_\sigma = \begin{array}{c} \\ a \\ b \\ aa \end{array} \begin{array}{c} \begin{array}{cc} a & b \end{array} \\ \begin{bmatrix} f(a\sigma a) & f(a\sigma b) \\ f(b\sigma a) & f(b\sigma b) \\ f(aa\sigma a) & f(aa\sigma b) \end{bmatrix} \end{array}$$

3. Perform rank n decomposition $\mathbf{H} = \mathbf{PS}$

Spectral Learning of Weighted Automata (in a nutshell)

1. Choose a set of prefixes and suffixes, $\mathcal{P}, \mathcal{S} \subset \Sigma^*$.
2. Estimate the Hankel sub-blocks $\mathbf{H}$ and $\mathbf{H}_\sigma \in \mathbb{R}^{\mathcal{P} \times \mathcal{S}}$ for each $\sigma \in \Sigma$

$$\mathbf{H} = \begin{array}{c} \\ a \\ b \\ aa \end{array} \begin{array}{c} \begin{array}{cc} a & b \end{array} \\ \begin{bmatrix} f(aa) & f(ab) \\ f(ba) & f(bb) \\ f(aaa) & f(aab) \end{bmatrix} \end{array} \quad \mathbf{H}_\sigma = \begin{array}{c} \\ a \\ b \\ aa \end{array} \begin{array}{c} \begin{array}{cc} a & b \end{array} \\ \begin{bmatrix} f(a\sigma a) & f(a\sigma b) \\ f(b\sigma a) & f(b\sigma b) \\ f(aa\sigma a) & f(aa\sigma b) \end{bmatrix} \end{array}$$

3. Perform rank n decomposition $\mathbf{H} = \mathbf{P}\mathbf{S}$
4. WA with initial/final weights $\boldsymbol{\alpha} = \mathbf{P}_{\lambda,:}$, $\boldsymbol{\omega} = \mathbf{S}_{:,\lambda}$ and transition matrices $\mathbf{A}^\sigma = \mathbf{P}^\dagger \mathbf{H}_\sigma \mathbf{S}^\dagger$ is a minimal WFA for f.

Spectral Learning of Weighted Automata (in a nutshell)

1. Choose a set of prefixes and suffixes, $\mathcal{P}, \mathcal{S} \subset \Sigma^*$.
2. Estimate the Hankel sub-blocks $\mathbf{H}$ and $\mathbf{H}_\sigma \in \mathbb{R}^{\mathcal{P} \times \mathcal{S}}$ for each $\sigma \in \Sigma$

$$\mathbf{H} = \begin{array}{c} \\ a \\ b \\ aa \end{array} \begin{array}{c} \begin{array}{cc} a & b \end{array} \\ \begin{bmatrix} f(aa) & f(ab) \\ f(ba) & f(bb) \\ f(aaa) & f(aab) \end{bmatrix} \end{array} \quad \mathbf{H}_\sigma = \begin{array}{c} \\ a \\ b \\ aa \end{array} \begin{array}{c} \begin{array}{cc} a & b \end{array} \\ \begin{bmatrix} f(a\sigma a) & f(a\sigma b) \\ f(b\sigma a) & f(b\sigma b) \\ f(aa\sigma a) & f(aa\sigma b) \end{bmatrix} \end{array}$$

3. Perform rank n decomposition $\mathbf{H} = \mathbf{P}\mathbf{S}$
4. WA with initial/final weights $\boldsymbol{\alpha} = \mathbf{P}_{\lambda,:}$, $\boldsymbol{\omega} = \mathbf{S}_{:,\lambda}$ and transition matrices $\mathbf{A}^\sigma = \mathbf{P}^\dagger \mathbf{H}_\sigma \mathbf{S}^\dagger$ is a minimal WFA for f.

$\rightarrow$ Efficient and consistent learning algorithms for weighted automata [Hsu et al., 2009; Bailly et al. 2009; Balle et al., 2014, ...].

Spectral Learning: when does it work?

Theorem (Exact case)

If the set of prefixes and suffixes $\mathcal{P}, \mathcal{S} \subset \Sigma^*$ are such that

$$\text{rank}(\mathbf{H}_{\mathcal{P},\mathcal{S}}) = \text{rank}(\mathbf{H}_f) < \infty$$

then the spectral learning algorithm returns a WFA computing f.

Spectral Learning: when does it work?

Theorem (Exact case)

If the set of prefixes and suffixes $\mathcal{P}, \mathcal{S} \subset \Sigma^*$ are such that

$$\text{rank}(\mathbf{H}_{\mathcal{P},\mathcal{S}}) = \text{rank}(\mathbf{H}_f) < \infty$$

then the spectral learning algorithm returns a WFA computing f.

- By a continuity argument, one can show that the result approximately holds when we are given noisy estimates of the Hankel matrices.

$\hookrightarrow$ When f is a probability distribution, we get an unbiased and consistent estimator! [c.f. work of B. Balle]

Learning and Planning in POMDP with WFA

Learning and Planning with WFA: Previous Approaches

Previous approaches based on spectral learning relies on 2 stages:

1. Learn a model of the POMDP dynamics
   - Learn a WFA computing $\mathbb{P}(h)$ from sampled trajectories
   - Spectral learning $\rightarrow$ Consistent estimator of the POMDP's dynamics

Learning and Planning with WFA: Previous Approaches

Previous approaches based on spectral learning relies on 2 stages:

1. Learn a model of the POMDP dynamics
   - Learn a WFA computing $\mathbb{P}(h)$ from sampled trajectories
   - Spectral learning $\rightarrow$ Consistent estimator of the POMDP's dynamics
2. Plan in the learned POMDP
   - Learn a reward model (e.g. Q function) and plan accordingly
   - Point-based Value Iteration (Pineau et al., 2003) or Fitted-Q (Ernst et al., 2005)

Learning and Planning with WFA: Previous Approaches

Previous approaches based on spectral learning relies on 2 stages:

1. Learn a model of the POMDP dynamics
   - Learn a WFA computing $\mathbb{P}(h)$ from sampled trajectories
   - Spectral learning $\rightarrow$ Consistent estimator of the POMDP's dynamics
2. Plan in the learned POMDP
   - Learn a reward model (e.g. Q function) and plan accordingly
   - Point-based Value Iteration (Pineau et al., 2003) or Fitted-Q (Ernst et al., 2005)

Drawbacks:

- Reward and dynamics are decoupled in the learning process
- Sample inefficient (parts of the learned dynamics may be useless for planning)
- Planning is computationally costly

Learning and Planning with WFA: Previous Approaches

Previous approaches based on spectral learning relies on 2 stages:

1. Learn a model of the POMDP dynamics
   - Learn a WFA computing $\mathbb{P}(h)$ from sampled trajectories
   - Spectral learning $\rightarrow$ Consistent estimator of the POMDP's dynamics
2. Plan in the learned POMDP
   - Learn a reward model (e.g. Q function) and plan accordingly
   - Point-based Value Iteration (Pineau et al., 2003) or Fitted-Q (Ernst et al., 2005)

Drawbacks:

- Reward and dynamics are decoupled in the learning process
- Sample inefficient (parts of the learned dynamics may be useless for planning)
- Planning is computationally costly

Our approach: *directly* learn a reward model from trajectories

Learning and planning with the Q function

Q function: Given policy π, the value of taking action a_t after history h_t is

$$Q^{\pi}(h_t, a_t) = \mathbb{E}[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots | h_t, a_\tau = \pi(h_\tau), \tau > t]$$

Learning and planning with the Q function

Q function: Given policy π, the value of taking action a_t after history h_t is

$$Q^{\pi}(h_t, a_t) = \mathbb{E}[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots | h_t, a_\tau = \pi(h_\tau), \tau > t]$$

Policy improvement: Acting greedily w.r.t. Q^{π} results in a better policy $\tilde{\pi}$, i.e., $V(\tilde{\pi}) \geq V(\pi)$.

Learning and planning with the Q function

Q function: Given policy π, the value of taking action a_t after history h_t is

$$Q^{\pi}(h_t, a_t) = \mathbb{E}[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots | h_t, a_\tau = \pi(h_\tau), \tau > t]$$

Policy improvement: Acting greedily w.r.t. Q^{π} results in a better policy $\tilde{\pi}$, i.e., $V(\tilde{\pi}) \geq V(\pi)$.

Policy iteration: Alternate between policy evaluation and improvement

$$\pi_0 \xrightarrow{evaluate} Q^{\pi_0} \xrightarrow{improve} \pi_1 \xrightarrow{evaluate} Q^{\pi_1} \xrightarrow{improve} \cdots \rightarrow \pi^*$$

Learning and planning with the Q function

Q function: Given policy π, the value of taking action a_t after history h_t is

$$Q^{\pi}(h_t, a_t) = \mathbb{E}[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots | h_t, a_\tau = \pi(h_\tau), \tau > t]$$

Policy improvement: Acting greedily w.r.t. Q^{π} results in a better policy $\tilde{\pi}$, i.e., $V(\tilde{\pi}) \geq V(\pi)$.

Policy iteration: Alternate between policy evaluation and improvement

$$\pi_0 \xrightarrow{evaluate} Q^{\pi_0} \xrightarrow{improve} \pi_1 \xrightarrow{evaluate} Q^{\pi_1} \xrightarrow{improve} \cdots \rightarrow \pi^*$$

Main Contribution

We design a consistent learning algorithm to perform policy evaluation and improvement from sampled trajectories.

Goal

Given a policy π, we want to act greedily w.r.t. the Q-function. I.e., for any history h we want find the action a s.t.

$$a = \arg\max_{a \in \mathcal{A}} Q^{\pi}(h, a)$$

Goal

Given a policy π, we want to act greedily w.r.t. the Q-function. I.e., for any history h we want find the action a s.t.

$$a = \arg\max_{a \in \mathcal{A}} Q^{\pi}(h, a)$$

Let $\tilde{R}(h) = \sum_{s \in \mathcal{S}} \mathbb{P}(s|h) R(s)$ be the expected immediate reward after history h.

Goal

Given a policy π, we want to act greedily w.r.t. the Q-function. I.e., for any history h we want find the action a s.t.

$$a = \arg\max_{a \in \mathcal{A}} Q^{\pi}(h, a)$$

Let $\tilde{R}(h) = \sum_{s \in \mathcal{S}} \mathbb{P}(s|h) R(s)$ be the expected immediate reward after history h. We have

$$\begin{aligned} Q^{\pi}(h,a) &= \mathbb{E}(r_t + \gamma r_{t+1} + \cdots + \gamma^i r_{t+i} + \cdots | ha) \\ &= \sum_{z \in \mathcal{H}} \sum_{o \in \mathcal{O}} \gamma^{|z|} \tilde{R}(haoz) \mathbb{P}(haoz|ha) \\ &= \frac{\sum_{o \in \mathcal{O}} \sum_{z \in \mathcal{H}} \gamma^{|z|} \tilde{R}(haoz) \mathbb{P}(haoz)}{\mathbb{P}(ha)} \\ &= \frac{\sum_{o \in \mathcal{O}} \sum_{z \in \mathcal{H}} \gamma^{|z|} \tilde{R}(haoz) \mathbb{P}(haoz)/\pi(a|h)}{\mathbb{P}(h)} := \frac{\tilde{Q}^{\pi}(h,a)}{\mathbb{P}(h)} \end{aligned}$$

Lemma

Given a policy π, let $\tilde{Q}^\pi$ be the un-normalized Q-function defined by

$$\tilde{Q}^\pi(h,a) = \sum_{o\in\mathcal{O}} \sum_{z\in\mathcal{H}} \gamma^{|z|} \tilde{R}(haoz)\mathbb{P}(haoz)/\pi(a|h)$$

where $\tilde{R}(h) = \sum_{s\in\mathcal{S}} \mathbb{P}(s|h)R(s)$ is the expected immediate reward.
Then, $Q^\pi(h,a) = \frac{\tilde{Q}^\pi(h,a)}{\mathbb{P}(h)}$ for all $h \in \mathcal{H}, a \in \mathcal{A}$.

Lemma

Given a policy π, let $\tilde{Q}^\pi$ be the un-normalized Q-function defined by

$$\tilde{Q}^\pi(h,a) = \sum_{o\in\mathcal{O}} \sum_{z\in\mathcal{H}} \gamma^{|z|} \tilde{R}(haoz)\mathbb{P}(haoz)/\pi(a|h)$$

where $\tilde{R}(h) = \sum_{s\in\mathcal{S}} \mathbb{P}(s|h)R(s)$ is the expected immediate reward. Then, $Q^\pi(h,a) = \frac{\tilde{Q}^\pi(h,a)}{\mathbb{P}(h)}$ for all $h \in \mathcal{H}, a \in \mathcal{A}$.

Goal

Given a policy π, we want to act greedily w.r.t. the Q-function. I.e., for any history h we want find the action a s.t.

$$a = \arg\max_{a\in\mathcal{A}} Q^\pi(h,a) = \arg\max_a \tilde{Q}^\pi(h,a)$$

Goal

Given a policy π, we want to compute the un-normalized Q-function

$$\tilde{Q}^{\pi}(h,a) = \sum_{o \in \mathcal{O}} \sum_{z \in \mathcal{H}} \gamma^{|z|} \tilde{R}(haoz)\mathbb{P}(haoz)/\pi(a|h)$$

Goal

Given a policy π, we want to compute the un-normalized Q-function

$$\tilde{Q}^{\pi}(h,a) = \sum_{o \in \mathcal{O}} \sum_{z \in \mathcal{H}} \gamma^{|z|} \tilde{R}(haoz)\mathbb{P}(haoz)/\pi(a|h)$$

Main idea:

1. Recover a WFA computing the function $h \mapsto \tilde{R}(h)\mathbb{P}(h)$ from data

Goal

Given a policy π, we want to compute the un-normalized Q-function

$$\tilde{Q}^{\pi}(h,a) = \sum_{o \in \mathcal{O}} \sum_{z \in \mathcal{H}} \gamma^{|z|} \tilde{R}(haoz)\mathbb{P}(haoz)/\pi(a|h)$$

Main idea:

1. Recover a WFA computing the function $h \mapsto \tilde{R}(h)\mathbb{P}(h)$ from data
2. Use this WFA to compute $\tilde{Q}^{\pi}$ using the following result:

> Let $A = (\boldsymbol{\alpha}, \{\mathbf{A}^{\sigma}\}_{\sigma \in \Sigma}, \boldsymbol{\omega})$ be a WFA and $0 < \gamma < 1$ such that $\rho(\gamma \sum_{\sigma \in \Sigma}) < 1$. Then,
>
> $$\sum_{x \in \Sigma^*} \gamma^{|x|} f_A(x) =$$

Goal

Given a policy π, we want to compute the un-normalized Q-function

$$\tilde{Q}^{\pi}(h,a) = \sum_{o \in \mathcal{O}} \sum_{z \in \mathcal{H}} \gamma^{|z|} \tilde{R}(haoz)\mathbb{P}(haoz)/\pi(a|h)$$

Main idea:

1. Recover a WFA computing the function $h \mapsto \tilde{R}(h)\mathbb{P}(h)$ from data
2. Use this WFA to compute $\tilde{Q}^{\pi}$ using the following result:

> Let $A = (\boldsymbol{\alpha}, \{\mathbf{A}^{\sigma}\}_{\sigma \in \Sigma}, \boldsymbol{\omega})$ be a WFA and $0 < \gamma < 1$ such that $\rho(\gamma \sum_{\sigma \in \Sigma}) < 1$. Then,
>
> $$\sum_{x \in \Sigma^*} \gamma^{|x|} f_A(x) = \sum_{x \in \Sigma^*} \gamma^{|x|} \boldsymbol{\alpha}^{\top} \mathbf{A}^{x} \boldsymbol{\omega}$$

Goal

Given a policy π, we want to compute the un-normalized Q-function

$$\tilde{Q}^{\pi}(h,a) = \sum_{o \in \mathcal{O}} \sum_{z \in \mathcal{H}} \gamma^{|z|} \tilde{R}(haoz)\mathbb{P}(haoz)/\pi(a|h)$$

Main idea:

1. Recover a WFA computing the function $h \mapsto \tilde{R}(h)\mathbb{P}(h)$ from data
2. Use this WFA to compute $\tilde{Q}^{\pi}$ using the following result:

Let $A = (\boldsymbol{\alpha}, \{\mathbf{A}^{\sigma}\}_{\sigma \in \Sigma}, \boldsymbol{\omega})$ be a WFA and $0 < \gamma < 1$ such that $\rho(\gamma \sum_{\sigma \in \Sigma}) < 1$. Then,

$$\sum_{x \in \Sigma^*} \gamma^{|x|} f_A(x) = \sum_{x \in \Sigma^*} \gamma^{|x|} \boldsymbol{\alpha}^{\top} \mathbf{A}^{x} \boldsymbol{\omega} = \boldsymbol{\alpha}^{\top} (\mathbf{I} - \gamma \sum_{\sigma \in \Sigma} \mathbf{A}^{\sigma})^{-1} \boldsymbol{\omega}$$

Main Results

Theorem

For any POMDP, there exists a WFA A computing the function $h \mapsto \tilde{R}(h)\mathbb{P}(h)$.

Main Results

Theorem

For any POMDP, there exists a WFA A computing the function $h \mapsto \tilde{R}(h)\mathbb{P}(h)$.

Theorem

Let the WFA $B = \langle \beta, \{\mathbf{B}^{ao} \mid (a,o) \in \mathcal{A} \times \mathcal{O}\}, \boldsymbol{\tau} \rangle$ be a WFA computing the function $h \mapsto \tilde{R}(h)\mathbb{P}(h)$.
Then, for any policy π, the UQF for a history $h \in \mathcal{H}$ and an action $a \in \mathcal{A}$ can be computed by:

$$\tilde{Q}^{\pi}(h,a) = \frac{1}{\pi(a|h)} \beta^{\top} \boldsymbol{B}^{h} \left(\sum_{o \in \mathcal{O}} \boldsymbol{B}^{ao} \right) \left(\mathbf{I} - \gamma \sum_{(a',o') \in \mathcal{A} \times \mathcal{O}} \mathbf{B}^{a'o'} \right)^{-1} \boldsymbol{\tau}$$

Spectral Learning Algorithm for the UQF

1. Collect trajectories from the policy π
2. Estimate WFA $B = \langle \beta, \{\mathbf{B}^{ao} \mid (a,o) \in \mathcal{A} \times \mathcal{O}\}, \boldsymbol{\tau} \rangle$ computing the function $\tilde{R}(h)\mathbb{P}(h)$ from data (spectral learning algorithm).
3. Construct the WFA A with initial weight vector $\boldsymbol{\alpha} = \beta$, transition matrices $\mathbf{A}^{ao} = \mathbf{B}^{ao}$ and final weight vector

$$\boldsymbol{\omega} = \left(\mathbf{I} - \gamma \sum_{\sigma \in \Sigma} \mathbf{B}_\sigma\right)^{-1} \boldsymbol{\tau}$$

4. For any $h \in \Sigma^*$, $a \in \mathcal{A}$, the UQF is computed by:

$$\tilde{Q}^{\Pi}(h,a) = \frac{1}{\pi(a|h)} \sum_{o \in \mathcal{O}} \boldsymbol{\alpha}^\top \mathbf{A}^h \mathbf{A}^{ao} \boldsymbol{\omega}$$

5. Return the policy:

$$\pi : h \mapsto \arg\max_{a \in \mathcal{A}} \tilde{Q}^{\Pi}(h,a)$$

Experimental Results

Grid World Navigation

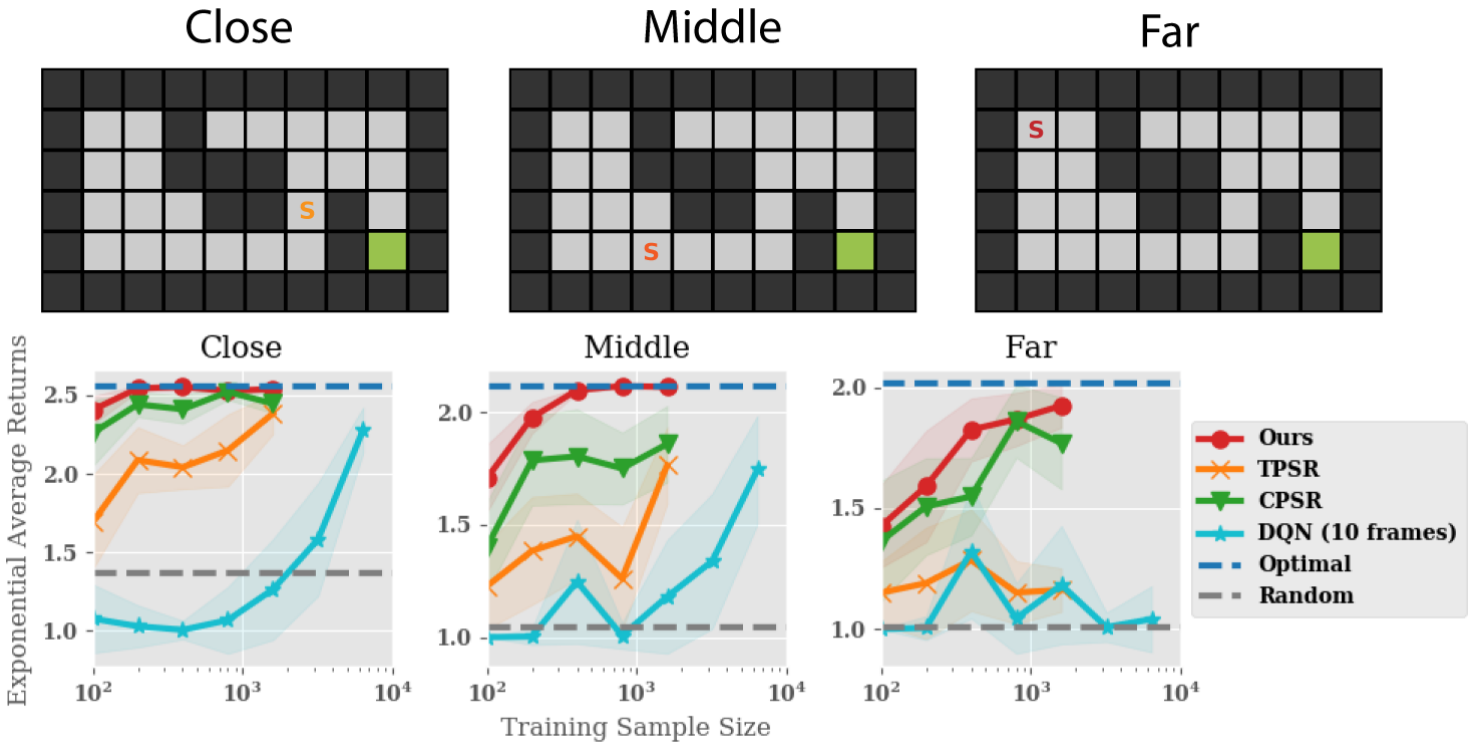


Figure: Experiments on three grid world tasks. The agent only receives knowledge about the number of the surrounding walls in four direction .

S-PocMan Domain (Hamilton et al., 2014)

- Pacman is unaware of the maze structure.
- Receives a 4-bit observation describing the wall configuration at its current location.
- Receives a 4-bit observation indicating whether a ghost is visible via direct line of sight

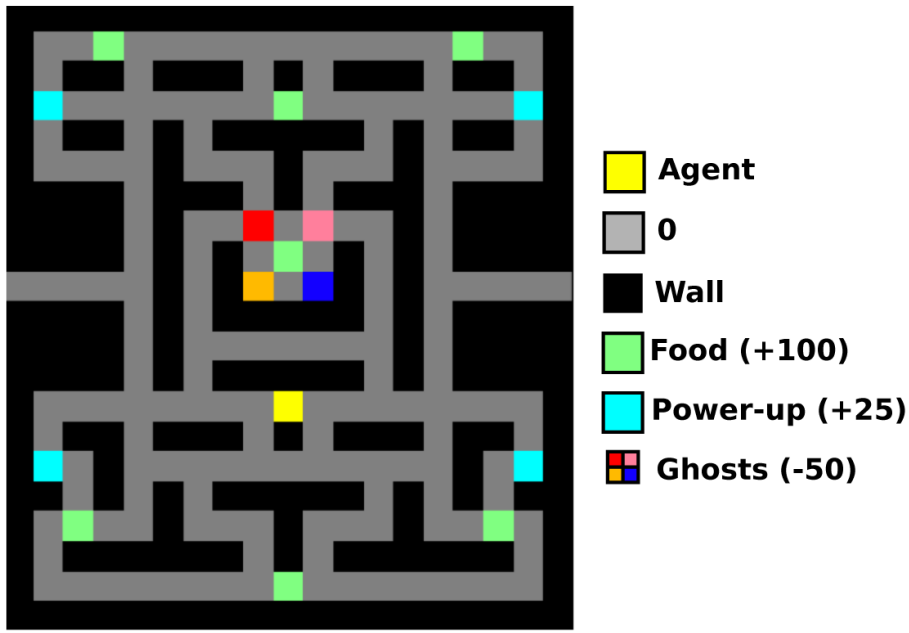


Figure: S-PocMan domain

S-PocMan Domain

Method	Fitted-Q Iterations	Time (s)	Returns
UQF	-	**2**	**-92**
CPSR	400	489	-101
	100	116	-109
	50	60	-150
	10	15	-200

Table: Training time for one policy iteration and averaged accumulated discounted rewards on S-PocMan trained on 500 trajectories.

Conclusion and Future Work

Conclusion and Future Work

- We proposed a sample efficient and consistent learning and planning algorithm for POMDP
- Our approach integrates planning and learning in one step
- The resulting learning algorithm comes with theoretical guarantees (e.g., it is consistent)

Conclusion and Future Work

- We proposed a sample efficient and consistent learning and planning algorithm for POMDP
- Our approach integrates planning and learning in one step
- The resulting learning algorithm comes with theoretical guarantees (e.g., it is consistent)

Future Work:

- Extending the approach to continuous actions/observations (Rabusseau et al., 2018)
- Scale up the approach to larger problems leveraging connections with tensor networks
  - Li, T., Precup, D., Rabusseau, G. *Connecting Weighted Automata, Tensor Networks and Recurrent Neural Networks through Spectral Learning.* arXiv preprint arXiv:2010.10029 (2020).

Thanks for listening!
Questions?

Boots, B., Siddiqi, S. M., and Gordon, G. J. (2011). Closing the learning-planning loop with predictive state representations. *The International Journal of Robotics Research*, 30(7):954–966.

Ernst, D., Geurts, P., and Wehenkel, L. (2005). Tree-based batch mode reinforcement learning. *Journal of Machine Learning Research*, 6:503–556.

Hamilton, W., Fard, M. M., and Pineau, J. (2014). Efficient learning and planning with compressed predictive states. *The Journal of Machine Learning Research*, 15(1):3395–3439.

Hausknecht, M. and Stone, P. (2015). Deep recurrent q-learning for partially observable mdps. *arXiv preprint arXiv:1507.06527*.

Littman, M. L. and Sutton, R. S. (2002). Predictive representations of state. In *Advances in neural information processing systems*, pages 1555–1561.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. (2013). Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*.

Pineau, J., Gordon, G., Thrun, S., et al. (2003). Point-based value iteration: An anytime algorithm for pomdps. In *IJCAI*, volume 3, pages 1025–1032. Citeseer.

Rabusseau, G., Balle, B., and Pineau, J. (2017). Multitask spectral learning of weighted automata. In *Advances in Neural Information Processing Systems*, pages 2588–2597.

Rabusseau, G., Li, T., and Precup, D. (2018). Connecting weighted automata and recurrent neural networks through spectral learning. *arXiv preprint arXiv:1807.01406*.

Singh, S., James, M. R., and Rudary, M. R. (2004). Predictive state representations: A new theory for modeling dynamical systems. In *Proceedings of the 20th conference on Uncertainty in artificial intelligence*, pages 512–519. AUAI Press.

Singh, S. P., Littman, M. L., Jong, N. K., Pardoe, D., and Stone, P. (2003). Learning predictive state representations. In *Proceedings of the 20th International Conference on Machine Learning (ICML-03)*, pages 712–719.