Metric-E^3

John Langford

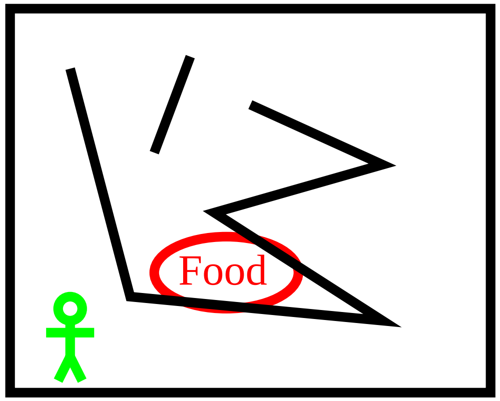
(With Sham Kakade and Michael Kearns)

The goal: solving reinforcement learning in a large state space

Solve: find a near-optimal policy

Reinforcement learning: On a Markov Decision Process

Large state space: continuous or merely large discrete



Markov Decision Process

Spaces:

1. states, S
2. actions, A

Dynamics:

1. Given $s \in S$, reward$=R(s)$
2. Given $s \in S$ and $a \in A$, next state $s' \in S$ drawn from $P(s'|a,s)$ for some P

Solving

Efficiently find a policy $\pi : S \to A$ to maximize expected reward in the next T steps.

NOT Solving

- “Policy Gradient” is not a solution (no guarantee of near optimality)

NOT Solving

- "Policy Gradient" is not a solution (no guarantee of near optimality)

- "Policy iteration" is not a solution (no guarantee of near optimality)

NOT Solving

- “Policy Gradient” is not a solution (no guarantee of near optimality)
- “Policy iteration” is not a solution (no guarantee of near optimality)
- Q-learning is not a solution (*far* to slow)

NOT Solving

- “Policy Gradient” is not a solution (no guarantee of near optimality)
- “Policy iteration” is not a solution (no guarantee of near optimality)
- Q-learning is not a solution (*far* to slow)
- Dynamic programming is not a solution (requires model)

Solution Methods

- E^3 (Explicit-Explore-or-Exploit) Running time $P(|S|)$. (KS 1998)

Solution Methods

- E^3 (Explicit-Explore-or-Exploit) Running time $P(|S|)$ish. (KS 1998)

- Factored-E^3, Running time $P(|L|)$, $L = \#$ of parameters (KK 1999)

Solving

- E^3 (Explicit-Explore-or-Exploit) Running time $P(|S|)$ish. (KS 1998)
- Factored-E^3, Running time $P(|L|)$, $L = \#$ of parameters (KK 1999)
- $RMAX$ (E^3 generalization & simplification) (BT 2001)

How can we make Reinforcement learning work in *large* state spaces?

You Can't

Theorem: There exists a set of problems requiring $O\left(\frac{|S||A|T \ln \frac{1}{\delta}}{\epsilon}\right)$ time to solve for any algorithm.

You Can

People do it all the time.

Reconciliation

Reconciliation

Lower bound $\Rightarrow$ additional assumption necessary

Reconciliation

Lower bound $\Rightarrow$ additional assumption necessary

Why can people succeed?

Reconciliation

Lower bound $\Rightarrow$ additional assumption necessary

Why can people succeed?

The world has structure.

The Metric-E^3 detective story

1. Approaches which fail.

2. Our structure.

3. The algorithm.

4. The analysis.

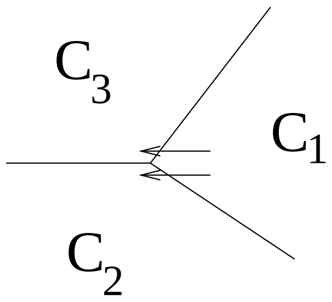
A dead end (commonly used)

1. (metric $\Rightarrow$) Aggregate states into superstates.

2. Derive (somehow) dynamics between the superstates.

3. Reinforcement Learn on the superstates.

Problem: dynamics CANNOT be markovian.



An elegant dead end (DG 1997)

1. dynamics $\Rightarrow$ aggregate states into superstates

2. Reinforcement Learn on superstates

Problem: in a real world, dynamics $\Rightarrow$ no simplification.

Our structure

1. Use metric

2. Assume local modeling

3. Assume approximate planning

assumption = difficulty abstraction

Metric

Not an assumption—a metric always exists.

Motivation:

1. Fundamental structure.
2. Intuitive.
3. Appears related to human RL ability.

$d((s,a),(s',a')) \geq 0$ and $d((s',a'),(s,a)) = d((s,a),(s',a'))$

Note: triangle inequality unnecessary

Local Model

GIVEN: sufficient "nearby" experience

ASSUME: ability to model local dynamics

$\forall$nearby $s, a\ \exists \hat{P} :\ \sum_{s'} |\hat{P}(s'|s, a) - P(s'|s, a)| \leq \alpha$

Motivation: lower bound $\Rightarrow$ model generalization necessary

Approximate Plan

GIVEN: model $+$ Reward structure

ASSUME: ability to find near-optimal policy, π.

$V_M(\pi) \geq V_M(\pi^*) - \beta$

$V_M(\pi)$ = value of policy π in model M.

Motivation:

1. Approximate planning algorithms exist (KL 2002, KMN 1999)

2. planning time lower bound $\Rightarrow$ generalization necessary

Algorithm

1. Move randomly until "known state" (local model)
2. Approximate Plan to:
   (a) escape ($M =$ known world, $R = 1$ in unknown states)
   (b) exploit ($M =$ known world, $R = 0$ in unknown states)
3. If $V_{\text{escape}}(\pi_{\text{escape}}) > \epsilon$, then use π_{escape}
4. otherwise, return π_{exploit}.

Analysis I

Theorem: (correctness) $Pr(\text{bad output policy}) \leq \delta$

bad output policy $= V_M(\pi, s) < V_M(\pi^*, s) - \epsilon - 2\beta - 2\alpha T$

T = horizon time

β = approximate plan error

α = local model error

ϵ = nearness-to-optimal

Analysis II

Theorem: (speed) $Pr(\text{bad sample complexity}) \leq \delta$

$N(\alpha)$ = covering number.

m = enough experience for local model.

bad sample complexity $> \frac{TmN(\alpha)}{\epsilon - \alpha T} \ln \frac{1}{\delta} + mN(\alpha)$

Analysis III

Let k = sample complexity.

Theorem: (running time)

$$\text{run time} \leq \frac{k(k-1)}{2} t_{\text{metric}} + 2k(t_{\text{plan}} + c_{\text{gen}} t_{\text{model}}) + O(k)$$

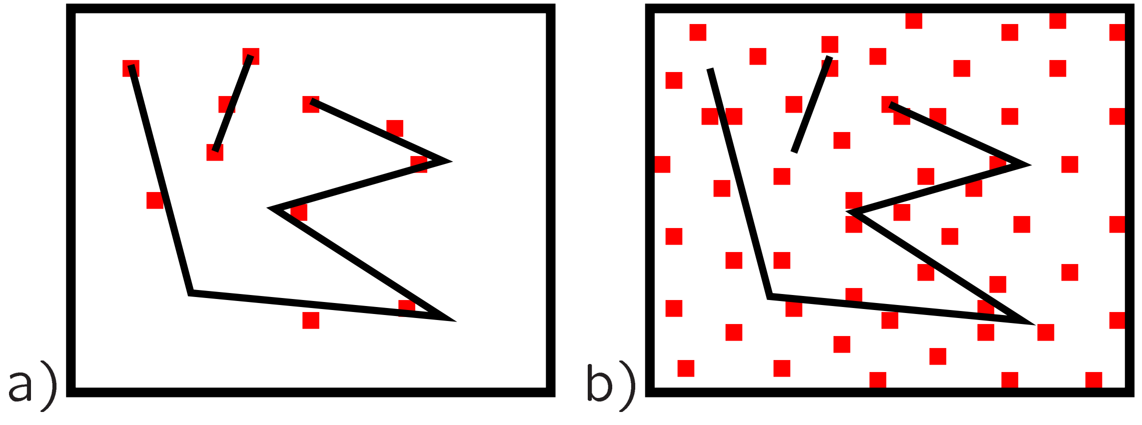
t_{metric} = execution time of metric

t_{plan} = execution time of approximate plan

t_{model} = execution time of local model

c_{gen} = number of calls to local model per approximate plan

Example



a) cover with Line-of-sight metric.

b) cover with Line-of-sight + max distance metric.

Conclusion

1. We (finally) have math to validate intuitive answer.

2. Don't aggregate. Instead, locally model.

3. RL is closer to viable.

Important undone work

1. write a program, see it work
   (a) caveat: parti-game
   (b) caveat: robot path planning
2. Very few algorithms for local modeling exist.
3. Measurement of modeling failure very important.