Value Function Approximation

Andrew W. Moore

CMU

(thanks: Justin Boyan, Scott Davies, Jeff Schneider)

Andrew W. Moore, Carnegie Mellon University
5000 Forbes Ave, Pittsburgh PA 15213
awm@cs.cmu.edu, phone 412-268-7599

Introduction

Grid-like (local) Value-function-approximation

Aggressive Value-function-approximation for huge tasks.

What are the current VFA approaches?

Alternatives to VFA

INTRODUCTION

When do we want to use reinforcement learning?

When do we want to use reinforcement learning?

(1) You have a system to control. You don't know its dynamics. You are told, or perhaps must learn, a reward function.

When do we want to use reinforcement learning?

(1) You have a system to control. You don't know its dynamics. You are told, or perhaps must learn, a reward function.

(2) (The Tesauro approach). You know a model of the system in advance.

When do we want to use reinforcement learning?

(1) You have a system to control. You don't know its dynamics. You are told, or perhaps must learn, a reward function.

- System can be described by $< 10^7$ discrete states.
- System has a huge or continuous state space.

(2) (The Tesauro approach). You know a model of the system in advance.

- System can be described by $< 10^7$ discrete states.
- System has a huge or continuous state space.

When do we want to use reinforcement learning?

(1) You have a system to control. You don't know its dynamics. You are told, or perhaps must learn, a reward function.

- System can be described by $< 10^7$ discrete states.
- System has a huge or continuous state space.

(2) (The Tesauro approach). You know a model of the system in advance.

- System can be described by $< 10^7$ discrete states.
  - **RL is not needed here. Use Val. Iter, Dijkstra, DAG-SP, multigrid, overrelax...**
- System has a huge or continuous state space.

When do we want to use reinforcement learning?

(1) You have a system to control. You don't know its dynamics. You are told, or perhaps must learn, a reward function.

- System can be described by $< 10^7$ discrete states.
  - **Model-free, online method like Q, TD.**
  - **OR learn model at same time as doing one of those. Dyna,Prisweep...**
- System has a huge or continuous state space.

(2) (The Tesauro approach). You know a model of the system in advance.

- System can be described by $< 10^7$ discrete states.
  - **RL is not needed here. Use Val. Iter, Dijkstra, DAG-SP, multigrid, overrelax...**
- System has a huge or continuous state space.

When do we want to use reinforcement learning?

(1) You have a system to control. You don't know its dynamics. You are told, or perhaps must learn, a reward function.

System can be described by < 10^7 discrete states.

Learn model at same time as doing one of those. Dyna,Prisweep...

System has a huge or continuous state space.

(2) (The Tesauro approach). You know a model of the system in advance.

System can be described by < 10^7 discrete states.

RL is not needed here. Use Val. Iter, Dijkstra, DAG-SP, multigrid, overrelax...

System has a huge or continuous state space.

WOULDN'T THIS BE NICE?

System can be described by < 10^7 discrete states.

Learn model at same time as doing one of those. Dyna,Prisweep...

System has a huge or continuous state space.

Learn model at same time as doing one of those. Dyna,Prisweep...

System can be described by < 10^7 discrete states.

RL is not needed here. Use Val. Iter, Dijkstra, DAG-SP, multigrid, overrelax...

System has a huge or continuous state space.

APPROXIMATE THE VALUE FUNCTION

RL is not needed here. Use Val. Iter, Dijkstra, DAG-SP, multigrid, overrelax.

THE MAIN POINT OF THIS TALK

Straightforward use of a function approximator for the value function in DP-like algorithms

UNRELIABLE

NOT THE MAIN POINT OF THIS TALK

Straightforward use of a function approximator for the value function in DP-like algorithms

CERTAIN TO FAIL

IN FACT...

(Subject to ignoring the fine details...)

For every continuous RL problem, there exists a function approximator for which value function approximation will work.

IN FACT...

(Subject to ignoring the fine details...)

For every continuous RL problem, there exists a function approximator for which value function approximation will work.

Just discretize state-space with a fine enough grid. That grid is the said function approximator.

Introduction

Grid-like (local) Value-function-approximation

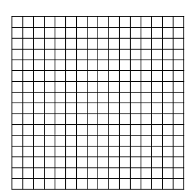
Aggressive Value-function-approximation for huge tasks.

What are the current VFA approaches?

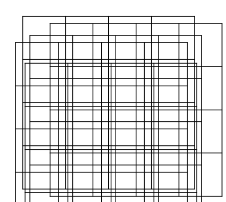
Alternatives to VFA

GRID-LIKE FUNCTION APPROXIMATORS

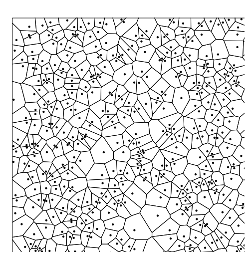
Grid-like approximators, when taken to high enough resolution, will work.



A Grid



Overlapping grids (CMAC)



Local Memory-based

Multilinear interpolation

GRID-LIKE METHODS

...are one reasonable approach to some value function approximation problems.

How coarse can the grid get?

Pretty coarse, actually, if you use the right algorithms.

USING ROBOT MOTION PLANNING METHODS FOR VF A

Example (in 2d space): the Car On The Hill.

USING ROBOT MOTION PLANNING METHODS FOR VF A

Example (in 2d space): the Car On The Hill.

POSSIBLE APPROACHES TO COMPUTING SOLUTION GIVEN A MODEL

- **Use a grid and use value iteration.**
- **Use a grid and use an online method.**
- **Use a coarser grid with some interpolation and an online method.**
- **Or....**

AN EASY ALGORITHM: BREADTH FIRST SEARCH

(Like value iteration, but backups occur only where they're needed)

- **Discretize action space.**
- **Compute all states one step from start.**
- **Compute all state two steps from start…**
- **..etc until find a state at the goal.**

INSIGHT FROM ROBOT MOTION PLANNING

Reinforcement learning based on an adjusted version of Latombe's non-holonomic motion planner [Latombe91]

Breadth first search creates too many states.

So place a coarse grid over state space.

Only expand a state in a grid cell if no other states have been previously expanded there.

RESUL TS FOR CAR-ON-HILL

- **16x16 grid, 486 Backups, trajectory length 80 steps**
- **7x7 grid, 115 Backups, trajectory length 91 steps**
- **LEARNING (between each attempt, fitting the dynamics with a locally weighted model selected by AutonMBL):**
  - **Learning trials to converge: 4**
  - **Total backups during learning: 1248**
  - **Total steps during learning: 479**
  - **Fourth traj length: 79 steps**

RESUL TS FOR ACROBOT

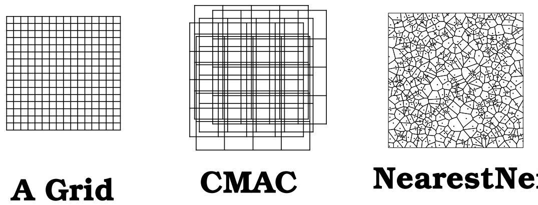
Model based:

- **9x9x9x9 Grid (306 grid cells actually visited).**
- **Backups to converge: 906**
- **Number of steps= 36 at 0.2secs/ step.**

Learning:

- **Lack of any exploration strategy meant Failure. More algorithmic complexity needed!**
- **But see later for a different learning algorithm for acrobot.**

Grid-like approximators, when taken to high enough resolution, will work.



Grid like methods...

USE MODEL		MODEL FREE
		(Sutton)
Much Faster		More biologically plausible
Can be more coarse: more generalizing; fewer parameters?		

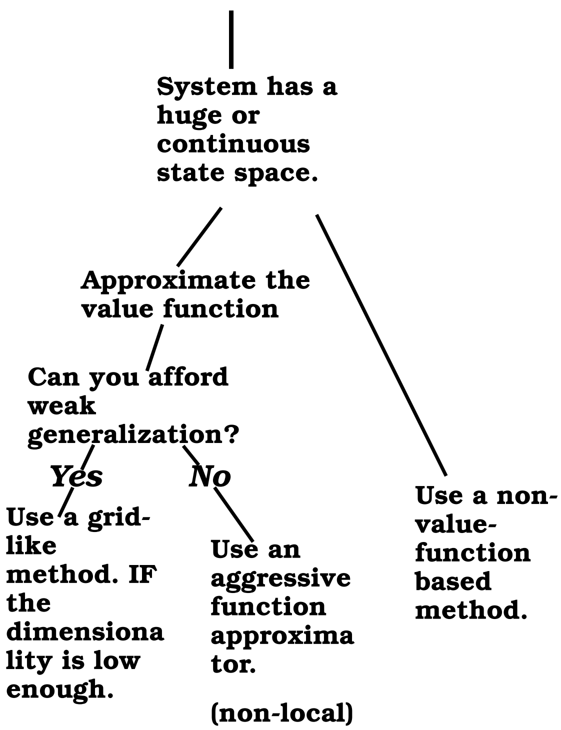
Introduction

Grid-like (local) Value-function-approximation

Aggressive Value-function-approximation for huge tasks.

What are the current VFA approaches?

Alternatives to VFA



USING AN AGGRESSIVE FUNCTION APPROXIMA TOR

- **DP-like methods maintain an estimate of the value function.**
- **J(s)---the current estimate of the long-term cost to go starting at state s.**
- **Sometimes we suspect that J*(s), the optimal value function, has a simple form. Perhaps one that could be described by a simple neural net, or low order polynomial terms.**

I REALLY REALLY WANT TO BE ABLE TO TAKE ADVANTAGE OF THAT SIMPLICITY!

A PROBLEM FORMULATION

I provide a simple function approximator. I want to ask:

"Get me the value function representable by this approximator which most closely fits J*"

How to do this?

- **Generate-and-test?**
- **Compute J* by some other means then approximate it?**
- **Run some form of DP-like algorithm (e.g. Value iteration, TD, Dijkstra's, DAG-SP...) replacing discrete state lookups by approximator lookups.**
- **Other**

"Get me the value function representable by this approximator which most closely fits J*"

Notice: Even this question is just a heuristic. What we really want is:

"Get me the value function representable by this approximator which causes the best performance at my task"

A COUNTERINTUITIVE RESULT

You decide to approximate the optimal value function for tic-tac-toe, based on 10 reasonable features of board positions.

You train using ALL STATES and using the CORRECT J* values.

- **Fun.Approximator A (cubic regression) gets a poor RMS error.**
- **Fun.Approximator B (neural net) gets almost the same RMS error.**

You test them in empirical play.

- **Fun.Approximator B plays much much better than A. (OUCH!)**

(results obtained by Justin Boyan)

HEURISTIC 1: Asking the question:

> **"Get me the value function representable by this approximator which most closely fits J*"**

HEURISTIC 2: Answering that question by the VFA technique:

> **Run some form of DP-like algorithm (e.g. Value iteration, TD, Dijkstra's, DAG-SP...) replacing discrete state lookups by approximator lookups.**

THE "FUNCTION APPROX PLUS DP-LIKE METHOD" HEURISTICS

- **Good news. The heuristics have resulted in good performance at a number of very complex tasks.**
- **Bad news. CounterExamples. The heuristic is not reliable. Can produce very bad performance on really simple tasks.**
  - **With value iteration [See shortly]**
  - **With TD [Bertsekas...]**
  - **With Q-learning [Thrun & Schwartz]**

FUNCTION APPROXIMATION WITH VALUE ITERATION

You have a deterministic random-walk task....

Regular value iteration would find the correct answer:

FUNCTION APPROXIMATION WITH VALUE ITERATION

The J* is a linear function. You decide to do value iteration with J() represented by a linear approximator at each time step, using a sample of input points...

That actually works.

FUNCTION APPROXIMATION WITH VALUE ITERATION

The J* is a linear function. You decide to do value iteration with J() represented by a quadratic approximator. Part way through, the value function looks like this:

The system diverges.

MORE COUNTEREXAMPLES

TASKS:

- **Linear walks**
- **"Puddle world"**
- **Car-on-hill**

APPROXIMATORS

- **Polynomial regression**
- **locally weighted & kernel regression (converges, but the policy doesn't work)**
- **neural nets**

In some cases, with enough tweaking, parameters can be found which *do* work. And of course, grid-like methods are fine.

Introduction

Grid-like (local) Value-function-approximation

Aggressive Value-function-approximation for huge tasks.

What are the current VFA approaches?

Alternatives to VFA

- **Good news. The heuristic has resulted in good performance at a number of very complex tasks.**
- **Bad news. CounterExamples. The heuristic is not reliable. Can produce very bad performance on really simple tasks.**

WHAT SHALL WE DO?

WHAT SHALL WE DO?

Idea One:

Don't get upset!

Enjoy the success stories!

Try to create more successful case studies and learn from them. [Tesauro, Crites, Dietterich+Zhang, Utgoff].

(but... let' s try to find out if and why the "DP backups" component is really helping us)

WHAT SHALL WE DO?

Idea Two:

Instead of

- **Run some form of DP-like algorithm replacing discrete state lookups by approximator lookups.**

could try

- **Run a new kind of DP-like algorithm designed to reduce function-approximation problems.**

[Boyan]

Under-relaxation may help [Sutton].

(but... no evidence yet on big real-world problems)

WHAT SHALL WE DO?

Idea Three:

Instead of approximating J* by:

- **Run some form of DP-like algorithm replacing discrete state lookups by approximator lookups.**

could try

- **Approximate J* by a new means.**

[Baird] Residual algorithms

[Dreyfus] Gradient approaches.

(but... no evidence yet on big real-world problems)

WHAT SHALL WE DO?

Idea Four:

Instead of approximating J* by:

- **Run some form of DP-like algorithm replacing discrete state lookups by approximator lookups.**

could try

- **Use a well-chosen function approximator.**

[This talk, Sutton] Grid-like local methods.

[Gordon] A class of divergence-proof function approximators

(but... grid-like won't always cut it. And no evidence yet on big real-world problems)

WHAT SHALL WE DO?

Idea Five:

Use techniques adapted from other fields.

From Optimal control (for dynamic systems; not, say, backgammon):

- **[Atkeson] Linear Quadratic Regulators**
- **[Singh] Using the Hamilton-Jacobi-Bellman equations.**

From Computer Game playing

- **[Utgoff]**
- **What do they do in Computer Chess? [McConnell]**

POMDPS [Littman]

ALTERNA TIVES TO VF A

Idea Six:

Forget about using Value Function Approximation!

- **Valueless Reinforcement Learning [MacDonald+Moore in progress]**
- **Tree-based generalizaion [Moore91 VRDP, Chapman+Kaelbling G*, Dayan Feudal Learning, Moore94 Partigame, McCallum95]**
- **Optimization of a parameterized policy.**
- **others....?**

ACROBOT AS AN EXPERIMENT DESIGN EXAMPLE

- **Invent a parameterized form of a policy:**

  action = f ($\mathbf{p}$, *state*)

  where p is a parameter vector.

- **Optimize p by running trials.**
- **We used AutoRSM, a memory-based experiment design system [Moore+Schneider95]**

  ADVANTAGE: Doesn't use Bellman

  DISADVANTAGE: doesnt use Bellman

  ACTUALITY:Finds a good policy in 5 trials

CONCLUSION

- **(1) You have a system to control. You don't know its dynamics. You are told, or perhaps must learn, a reward function.**
  - System can be described by < 10^7 discrete states.
  - System has a huge or continuous state space.
- **(2) (The Tesauro approach). You know a model of the system in advance.**
  - System can be described by < 10^7 discrete states.
  - System has a huge or continuous state space.

It would be really really good to be able to do this reliably.

It is possible that VFA is the key.