Week 10

Backward approximate dynamic programming
Backward MDP and the curse of dimensionality
Curse of dimensionality

- The ultimate policy is to optimize from now on:

\[ X_t^*(S_t) = \arg \max_{x_t} \left( C(S_t, x_t) + \mathbb{E}\left\{ \max_{\pi \in \Pi} \left\{ \mathbb{E}\left[ \sum_{t'=t+1}^{T} C(S_{t'}, X_t^\pi(S_{t'})) \mid S_{t+1} \right] \right\} \mid S_t, x_t \right\} \right) \]

- Ideally, we would like to replace the future contributions with a single value function:

\[ X_t^*(S_t) = \arg \max_{x_t} \left( C(S_t, x_t) + \mathbb{E}\left\{ V_{t+1}(S_{t+1}) \mid S_t, x_t \right\} \right) \]

- Sometimes we can compute this function exactly!
Curse of dimensionality

Energy storage with stochastic prices, supplies and demands.

\[ E^\text{wind}_t \]

Wind speed

\[ P^\text{grid}_t \]

Electricity prices

\[ R^\text{battery}_t \]

\[ W_{t+1} = \text{Exogenous inputs} \]

\[ S_t = \text{State variable} \]

\[ x_t = \text{Controllable inputs} \]

\[ D_t \]

\[ \hat{E}^\text{wind}_t = E^\text{wind}_t + \hat{E}^\text{wind}_t \]

\[ \hat{P}^\text{grid}_t = P^\text{grid}_t + \hat{P}^\text{grid}_t \]

\[ \hat{D}^{\text{load}}_{t+1} = D^{\text{load}}_t + \hat{D}^{\text{load}}_{t+1} \]

\[ \hat{R}^{\text{battery}}_{t+1} = R^{\text{battery}}_t + Ax_t \]
Curse of dimensionality

Bellman’s optimality equation

$$V(S_t) = \min_{x_t \in \mathcal{X}} \left( C(S_t, x_t) + \gamma \mathbb{E} \{ V(S_{t+1} | S_t) \} \right)$$

These are the “three curses of dimensionality.”
Curse of dimensionality

Backward dynamic programming in one dimension

Step 0: Initialize $V_{T+1}(R_{T+1}) = 0$ for $R_{T+1} = 0, 1, ..., 100$

Step 1: Step backward $t = T, T - 1, T - 2, ...$

Step 2: Loop over $R_t = 0, 1, ..., 100$

Step 3: Loop over all decisions $-(R_{\text{max}} - R_t) \leq x_t \leq R_t$

Step 4: Take the expectation over exogenous information:

Compute $Q(R_t, x_t) = C(R_t, x_t) + \sum_{w=0}^{100} V_{t-1}(\min\{R_{\text{max}}, R_t - x + w\})P^W(w)$

End step 4;

End Step 3;

Find $V_t^*(R_t) = \max_{x_t} Q(R_t, x_t)$

Store $X_t^{\pi^*}(R_t) = \arg \max_{x_t} Q(R_t, x_t)$. (This is our policy)

End Step 2;

End Step 1;
Curse of dimensionality

**Dynamic programming in multiple dimensions**

Step 0: Initialize $V_{T+1}(S_{T+1}) = 0$ for all states.

Step 1: Step backward $t = T, T-1, T-2,...$

Step 2: Loop over $S_t = (R_t, D_t, p_t, E_t)$ (four loops)

Step 3: Loop over all decisions $x_t$ (all dimensions)

Step 4: Take the expectation over each random dimension $(\hat{D}_t, \hat{p}_t, \hat{E}_t)$

Compute $Q(S_t, x_t) = C(S_t, x_t) + \\sum_{w_1=0}^{100} \sum_{w_2=0}^{100} \sum_{w_3=0}^{100} V_{t+1} \left( S^M_{t+1} = (w_1, w_2, w_3) \right) P^W(w_1, w_2, w_3)

End step 4;
End Step 3;
Find $V^*_t(S_t) = \max_{x_t} Q(S_t, x_t)$
Store $X^*_t(S_t) = \arg \max_{x_t} Q(S_t, x_t)$. (This is our policy)
End Step 2;
End Step 1;
Curse of dimensionality

Notes:

» There are potentially three “curses of dimensionality” when using backward dynamic programming:
  • The state variable – We need to enumerate all states. If the state variable is a vector with more than two dimensions, the state space gets very big, very quickly.
  • The random information – We have to sum over all possible realizations of the random variable. If this has more than two dimensions, this gets very big, very quickly.
  • The decisions – Again, decisions may be vectors (our energy example has five dimensions). Same problem as the other two.

» Some problems fit this framework, but not very. However, when we can use this framework, we obtain something quite rare: an optimal policy.
Curse of dimensionality

Strategies for approximating value functions:

» Backward dynamic programming
  • Exact using lookup tables
  • Backward approximate dynamic programming:
    – Linear regression
    – Low rank approximations

» Forward approximate dynamic programming
  • Approximation architectures
    – Lookup tables
      » Correlated beliefs
      » Hierarchical
    – Linear models
    – Convex/concave
  • Updating schemes
    – Pure forward pass TD(0)
    – Double pass TD(1)
Backward ADP-Chapter 16
Backward ADP

- Classical backward dynamic programming
  - Uses lookup table representations of value functions
  - Assumes the one-step transition matrix can be computed (which is also lookup table).
  - “Dynamic programming” does *not* suffer from the curse of dimensionality (as we show below), but lookup tables do.
  - There are three curses of dimensionality, but often it is the state variable that causes the most problems.
  - Backward ADP uses a sample of states rather than all the states, and a statistical model for the value of being in a state. At a minimum this fixes two of the three curses of dimensionality.
Backward ADP

- Backward approximate dynamic programming
  » Basic idea is to step backward in time, just as we do with classical backward dynamic programming.
  » Instead of looping over all the states, loop over a random sample.
  » Now, use the sample of values and states to produce an approximate value function:
    • Any statistical model
    • Low-rank approximations (works well when value functions are smooth).
  » You still need to take full expectation (although this might be approximated) and search over all actions.
Backward ADP

Step 0: Initialize $\bar{V}_{T+1}(S_{T+1}) = 0$ for all states.

Step 1: Step backward $t = T, T - 1, T - 2, ...$

Step 2: Loop over a random sample of states $\hat{s}_t = (R_t, D_t, p_t, E_t)$ (one loop)

Step 3: Loop over all decisions $x_t$ (all dimensions)

Step 4: Take the expectation over each random dimension $(\hat{D}_t, \hat{p}_t, \hat{E}_t)$

Compute $Q(\hat{s}_t, x_t) = C(\hat{s}_t, x_t) +$

$$\sum_{w_1=0}^{100} \sum_{w_2=0}^{100} \sum_{w_3=0}^{100} \bar{V}_{t+1}^M(\hat{s}_t, x_t, W_{t+1} = (w_1, w_2, w_3)) P^W(w_1, w_2, w_3)$$

End step 4;

End Step 3;

Find $\hat{v}_t(\hat{s}_t) = \max_{x_t} Q(\hat{s}_t, x_t)$

End Step 2;

Use sampled $\hat{v}_t(\hat{s}_t)$'s to find an approximate $\bar{V}_t(s)$.

End Step 1;
Backward ADP

Backward ADP with the post-decision state

- Computing the imbedded expectation can be a pain.
- Instead of sampling over (pre-decision) states, sample post-decision states $s_{t-1}^x$.
- Then draw a sample $W_t$, and simulate our way to the next pre-decision state $s_t$.
- From $s_t$, compute the sampled value $v_t$ from
  \[ v_t = \max_x (C(s_t, x_t) + V_t^x(s_t^x)) \]
- Do this $N$ times and create a dataset $(s_{t-1}^{x,n}, v_t^n)_{n=1}^N$
- Now use this dataset to fit a statistical model for $V_{t-1}^x(s_{t-1}^x)$.
- Repeat.
Backward ADP

- Backward ADP for a clinical trial problem
  - Problem is to learn the value of a new drug within a budget of patients to be tested.
  - Backward MDP required 268-485 hours.
  - Forward ADP exploiting monotonicity (we will cover this later) required 18-30 hours.
  - Backward ADP required 20 minutes, with a solution that was 1.2 percent within optimal.

### Table 3: Computation Time Comparison between Backward MDP Algorithm and ADP Algorithm

<table>
<thead>
<tr>
<th>(n_1/n_3)</th>
<th>(n_2/n_3)</th>
<th>MDP CPU (hrs)</th>
<th>ADP</th>
<th>Gap (%)</th>
<th>Discretization</th>
<th>Interval Length</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Enrollment</td>
<td>Treatment</td>
</tr>
<tr>
<td>0.35</td>
<td>0.65</td>
<td>360</td>
<td>26</td>
<td>forward</td>
<td>3.2</td>
<td>20</td>
</tr>
<tr>
<td>0.40</td>
<td>0.65</td>
<td>290</td>
<td>18</td>
<td>forward</td>
<td>0.1</td>
<td>20</td>
</tr>
<tr>
<td>0.40</td>
<td>0.75</td>
<td>440</td>
<td>18</td>
<td>forward</td>
<td>-3.2</td>
<td>20</td>
</tr>
<tr>
<td>0.55</td>
<td>0.85</td>
<td>485</td>
<td>18</td>
<td>forward</td>
<td>3.7</td>
<td>20</td>
</tr>
<tr>
<td>0.55</td>
<td>0.85</td>
<td>268</td>
<td>0.3</td>
<td>backward</td>
<td>1.2</td>
<td>10</td>
</tr>
</tbody>
</table>

© 2019 Warren B. Powell
Backward ADP

- Energy storage problem
  - Lookup table – 99.3 percent of optimal, .67 hours.
  - Backward MDP 11.3 hours.

<table>
<thead>
<tr>
<th>Case</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>Avg</th>
<th>CPU (hrs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lookup-.01</td>
<td>99.1</td>
<td>96.7</td>
<td>98.0</td>
<td>88.8</td>
<td>98.0</td>
<td>100.0</td>
<td>93.2</td>
<td>98.4</td>
<td>99.6</td>
<td>99.4</td>
<td>98.8</td>
<td>98.9</td>
<td>97.4</td>
<td>0.41</td>
</tr>
<tr>
<td>Lin-.01</td>
<td>96.2</td>
<td>96.4</td>
<td>96.9</td>
<td>91.8</td>
<td>88.2</td>
<td>95.2</td>
<td>95.4</td>
<td>98.9</td>
<td>98.9</td>
<td>98.2</td>
<td>98.3</td>
<td>99.5</td>
<td>96.2</td>
<td>1.62</td>
</tr>
<tr>
<td>Lookup-.10</td>
<td>100.1</td>
<td>99.5</td>
<td>99.3</td>
<td>97.1</td>
<td>99.7</td>
<td>100.2</td>
<td>97.2</td>
<td>99.4</td>
<td>100.0</td>
<td>100.1</td>
<td>99.6</td>
<td>99.8</td>
<td>99.3</td>
<td>0.67</td>
</tr>
<tr>
<td>Lin-.10</td>
<td>96.3</td>
<td>96.5</td>
<td>98.1</td>
<td>91.1</td>
<td>88.3</td>
<td>95.2</td>
<td>94.9</td>
<td>98.9</td>
<td>98.9</td>
<td>98.2</td>
<td>99.0</td>
<td>99.2</td>
<td>96.2</td>
<td>2.72</td>
</tr>
<tr>
<td>API</td>
<td>82.7</td>
<td>77.9</td>
<td>79.5</td>
<td>57.3</td>
<td>76.2</td>
<td>90.8</td>
<td>50.5</td>
<td>80.4</td>
<td>94.7</td>
<td>90.0</td>
<td>86.4</td>
<td>86.9</td>
<td>79.5</td>
<td>12.0</td>
</tr>
<tr>
<td>PFA</td>
<td>93.6</td>
<td>92.3</td>
<td>93.4</td>
<td>71.4</td>
<td>80.6</td>
<td>91.3</td>
<td>72.3</td>
<td>93.4</td>
<td>97.3</td>
<td>95.2</td>
<td>96.0</td>
<td>94.6</td>
<td>89.3</td>
<td>5.14</td>
</tr>
<tr>
<td>DLA-24-.05</td>
<td>87.7</td>
<td>86.6</td>
<td>87.4</td>
<td>73.5</td>
<td>84.8</td>
<td>93.3</td>
<td>90.4</td>
<td>89.9</td>
<td>96.8</td>
<td>92.2</td>
<td>92.3</td>
<td>95.3</td>
<td>89.2</td>
<td>N/A</td>
</tr>
<tr>
<td>DLA-72-.01</td>
<td>91.1</td>
<td>91.5</td>
<td>90.8</td>
<td>86.1</td>
<td>99.4</td>
<td>95.5</td>
<td>101.7</td>
<td>93.0</td>
<td>97.9</td>
<td>94.4</td>
<td>94.2</td>
<td>94.3</td>
<td>94.2</td>
<td>N/A</td>
</tr>
</tbody>
</table>

© 2019 Warren B. Powell
Backward ADP

- Resource allocation in Africa
  - Extended widely cited myopic policy to a dynamic setting.

![Utility function in environment with shocks](image-url)
By now I have applied backward ADP to approximate four projects, three with rigorous benchmarks, and one (the resource allocation problem) with a high quality benchmark (the myopic policy).

Each time it seems to have worked very well.
Histories of approximate dynamic programming and reinforcement learning
1959 – Operations research

» Bellman recognizes the limitations of classical backward dynamic programming.

» Introduces the idea of statistically approximating value functions.

» This line of research quickly died out in the operations research community.
The fields of stochastic optimization

- **Approximate dynamic programming/reinforcement learning**
  - 1959 paper by Bellman – first attempt at ADP
  - ADP in control theory – 1974 dissertation of Paul Werbos
  - Reinforcement learning in computer science – 1980 research of Rich Sutton and Andy Barto
    - 1998 book *Reinforcement Learning* establishes the field
  - 1996 book *Neuro-Dynamic programming* – First to bridge the theory of stochastic approximation methods (Robbins and Monro) with reinforcement learning
  - Late 1990’s – ADP returns to operations research
    - 1994 dissertation of Ben van Roy
    - Late 1990’s onward – Value function approximations for MDPs (discrete actions)
    - 1998 onward – use of ADP for vector-valued actions (Powell and students)
    - Three curses of dimensionality; high dimensional decision vectors (action spaces)
Histories of ADP/reinforcement learning

1974 – Controls community

» Paul Werbos introduces “backpropagation” for approximating the “cost to go” function for continuous controls problems.

» Engineering controls community continues to develop these ideas, with special emphasis on the use of neural networks in two ways:
  • Actor nets – A neural network for the policy (which chooses the action given a state).
  • Critic nets – A neural network for approximating the value function (cost to go function in the language of control theory)

» Paul Werbos becomes an NSF program officer and continues to promote “approximate dynamic programming.” Funded workshops on ADP in 2002 and 2006.


» 1996 book “Neuro-Dynamic Programming” by Bertsekas and Tsitsiklis formally bridges Q-learning and stochastic approx. methods

© 2019 Warren B. Powell
Histories of ADP/reinforcement learning

Histories of ADP/reinforcement learning

History of Q-learning

» Began ~1980 with Andy Barto (supervisor) and Rich Sutton (student) studying behavior of mice in mazes.

» Heuristically developed basic feedback mechanism:

\[
\hat{q}^n(s^n, a^n) = C(s^n, a^n) + \gamma \max_{a'} Q^{n-1}(s', a')
\]

\[
Q^n(s^n, a^n) = (1 - \alpha_{n-1})Q^{n-1}(s^n, a^n) + \alpha \hat{q}^n(s^n, a^n)
\]

» Late 1980’s the link to Markov decision processes was made, and the community adopted the basic notation of Markov decision processes.

» Bizarrely, the RL community adopted popular test problems from the controls community, which are primarily deterministic:
  • Inverted pendulum problem.
  • Hill-climbing
  • Truck backer-upper
  • Robotics applications.
Histories of ADP/reinforcement learning

Richard S. Sutton
Computer scientist

Richard S. Sutton is a Canadian computer scientist. Currently he is professor of Computer Science and iCORE chair at the University of Alberta. Wikipedia

Born: Ohio
Doctoral advisor: Andrew Barto
Residence: Canada
Alma maters: University of Massachusetts Amherst, Stanford University
Fields: Artificial intelligence, Reinforcement learning

Andrew Barto
Professor

Andrew G. Barto is a professor of computer science at University of Massachusetts Amherst, and chair of the department since January 2007. His main research area is reinforcement learning. Wikipedia

Born: 1948
Education: University of Michigan
Field: Computer Science
Notable student: Richard S. Sutton

Reinforcement Learning

Cited by 29594
Histories of ADP/reinforcement learning

- Second edition of Reinforcement Learning

© 2019 Warren B. Powell
Histories of ADP/reinforcement learning

From Q-learning to other policies

» From the second edition:

1.4 Limitations and Scope

Most of the reinforcement learning methods we consider in this book are structured around estimating value functions, but it is not strictly necessary to do this to solve reinforcement learning problems. For example, methods such as genetic algorithms, genetic programming, simulated annealing, and other optimization methods have been used to approach reinforcement learning problems without ever appealing to value functions. These methods evaluate the “lifetime” behavior of many non-learning agents, each using a different policy for interacting with its environment, and select those that are able to obtain the most reward. We call these evolutionary methods because their operation is analogous to the way biological evolution produces organisms with skilled behavior even when they do not learn during their individual lifetimes. If the space of policies is sufficiently small, or can be structured so that good policies are common or easy to find—or if a lot of time is available for the search—then evolutionary methods can be effective. In addition, evolutionary methods have advantages on problems in which the learning agent cannot accurately sense the state of its environment.

» This hints at policy search, but still ignores lookahead policies (Monte Carlo tree search)
Histories of ADP/reinforcement learning

1990’s – Operations research

» Ben van Roy (student of John Tsitsklis) developed the ideas of using regression models for solving the “curse of dimensionality problem” of dynamic programming.

» 1991 – Pereira and Pinto introduce the idea of Benders cuts for “solving the curse of dimensionality” for stochastic linear programs. Method called “stochastic dual decomposition procedure” (SDDP)

» ~2000 – Work of WBP on “adaptive dynamic programming” for high-dimensional problems in logistics.

» With Ben van Roy (who first introduced the term), WBP developed the idea of the post-decision state which opened the door for solving high-dimensional convex DPs.

» WBP switches to “approximate dynamic programming” after attending Werbos’ 2002 conference on “Approximate dynamic programming and reinforcement learning”
Histories of ADP/reinforcement learning

Today:

- Controls community now uses the term “adaptive dynamic programming”
  - Balanced using of “control laws” (PFAs), Hamilton/Jacobi/Bellman equations (VFAs) and “model predictive control” (DLAs)
- “Reinforcement learning” has spread from Q-learning to include “policy search” and Monte Carlo tree search (a form of direct lookahead – we will get to this later).
- 2014-2016 Two tutorials by WBP establish the “four classes of policies”
- “Optimization under Uncertainty” book (being written for this class) is the first to truly unify all of the different subcommunities of stochastic optimization.
Next steps

What we are going to cover:

» Forward approximate dynamic programming

• Estimating the value of a fixed policy

• Optimizing while learning
Learning the value of a policy

TD-learning
TD-learning

Temporal difference learning

» The “temporal difference” is given by

$$\delta_t(S^n_t, x^n_t) = \bar{V}^{n-1}_t(S^n_t) - (C(S^n_t, x^n_t) + \bar{V}^{n-1}_t(S^M(S^n_t, x^n_t, W_{t+1}))$$

$$= \bar{V}^{n-1}_t(S^n_t) - v^n_t$$

» In other words, this is “old estimate minus new estimate”. We can update our value function approximation using

$$\bar{V}^n_t(S^n_t) = \bar{V}^{n-1}_t(S^n_t) - \alpha_{n-1}\delta(S^n_t, x^n_t)$$

$$= (1 - \alpha_{n-1})\bar{V}^{n-1}_t(S^n_t) + \alpha_{n-1}v^n_t$$

» This is a basic form of “temporal difference learning” known as TD(0). The “temporal difference” reflects learning from one decision to the next (which occurs over time).
TD-learning

TD(\(\lambda\))

» A more general form of TD-learning uses discounted costs over the entire trajectory.

\[
V_t^n(S_t) = V_t^{n-1}(S_t) + \alpha_{n-1} \sum_{\tau=t}^{T} \lambda^{\tau-t} \delta_\tau. \tag{17.7}
\]

We derived this formula without a time discount factor. We leave as an exercise to the reader to show that if we have a time discount factor \(\gamma\), then the temporal-difference update becomes

\[
V_t^n(S_t) = V_t^{n-1}(S_t) + \alpha_{n-1} \sum_{\tau=t}^{T} (\gamma \lambda)^{\tau-t} \delta_\tau. \tag{17.8}
\]

» Think of \(\lambda\) as an “algorithmic discount factor” that helps to give credit for downstream rewards to earlier decisions. This has to be carefully tuned.
Approximating the value function

Temporal difference updates can be used in any recursive estimation algorithm:

- Lookup tables
  - Independent beliefs
  - Correlated beliefs
- Parametric models
  - Linear
  - Nonlinear
  - Shallow neural networks
- Nonparametric
  - Kernel regression
  - Locally linear
  - Deep neural networks
Q-learning

“Reinforcement learning”
Q-learning

- Mouse in a maze problem

Receive reward = 1
Q-learning

- **AlphaGo**
  - Much more complex state space.
  - Uses hybrid of policies:
    - PFA
    - VFA
    - Lookahead (DLA)
Q-learning

**Basic Q-learning algorithm**

» Basic update:

\[ \hat{q}^n(s^n, a^n) = C(s^n, a^n) + \gamma \max_{a'} Q_{n-1}^{n-1}(s', a') \]

\[ \bar{Q}_n^n(s^n, a^n) = (1 - \alpha_{n-1})\bar{Q}_n^{n-1}(s^n, a^n) + \alpha_{n-1}\hat{q}^n(s^n, a^n) \]

where

\[ s' = S^M(s^n, a^n, W^{n+1}) \]

» Given a state \( s^n \) and action \( a^n \), we simulate our way to state \( s' \).

» Need to determine:
  - State sampling process/policy
  - Action sampling policy
Q-learning

Some terms from reinforcement learning:

» “Behavior policy” is the policy used to choose actions
  • E.g. these are actions observed by a real system

» “Target policy” is the policy that we are trying to learn, which is to say the policy we want to implement.

» When the target policy is different from the behavior policy, then this is termed “off policy learning”

In this course

» The “learning policy” is the policy (often called an algorithm) that learns the value functions (or Q-factors)

» The “implementation policy” is the policy determined by the value functions (or Q-factors).
Q-learning

Learning policy

» This is the policy that determines what action to choose as a part of learning the Q-factors.

» “Exploitation”:

\[ a^n = \text{argmax}_{a',Q^n(s^n,a')} \]

» Other policies that involve exploration:
  • Epsilon-greedy – Choose greedy policy with probability \( \epsilon \), and explore with probability \( 1 - \epsilon \).
  • Policies based on upper confidence bounding, Thompson sampling, knowledge gradient, …
Q-learning

State sampling policies

» Trajectory following

\[ s^{n+1} = S^M(s^n, a^n, W^{n+1}) \]

• Helps to avoid sampling states that never happen
• Problem is that a suboptimal policy may mean that you are not sampling important states.

» Exploration
• Pick a state at random

» Hybrid
• Use trajectory following with randomization, e.g.

\[ s^{n+1} = S^M(s^n, a^n, W^{n+1}) + \epsilon^{n+1} \]
Q-learning

- Implementation policy
  » This is the policy we are going to follow based on the Q-factors:

\[
A^\pi(s) = \arg\max_{a, \tilde{Q}^n(s, a')}
\]

- The value of the implementation policy:

\[
F^{\pi,n} = \sum_{t=0}^{T} C(S_t, X^\pi(S_t), W_{t+1}(\omega)) \quad \text{where} \quad S_{t+1} = S^M(S_t, X^\pi(S_t), W_{t+1}(\omega))
\]

  » or

\[
F^{\pi,n} = \frac{1}{N} \sum_{n=1}^{N-1} \sum_{t=0}^{T} C(S_t, X^\pi(S_t), W_{t+1}(\omega^n)) \quad \text{where} \quad S_{t+1} = S^M(S_t, X^\pi(S_t), W_{t+1}(\omega^n))
\]

- The goal is to find an effective learning policy so that we obtain the best implementation policy.
Q-learning

Convergence rates:

![Policy Improvement vs. Updating Iterations](chart.png)
Q-learning

- **On vs off-policy learning:**
  - **On-policy learning** – Learning the value of a fixed policy:
    
    From a state $s^n$, choose action $a^n = \arg \max_{a'} Q^n(s^n, a')$
    
    Now go to state $s^{n+1}$:
    
    $$s^{n+1} = S^M(s^n, a^n, W^{n+1})$$
    
    Where $W^{n+1}$ is observed or sampled from some distribution.
  
  - **Off-policy learning:**
    - Sample actions according to a *learning policy* (called “behavior policy” in the RL literature). This is the policy used for learning the *implementation policy* (called the “target policy” in the RL literature).
    - Needs to be combined with a state sampling policy.
Q-learning

Model-free vs. model-based

» Model-based means we have a mathematical statement of how the problem evolves that can be simulated in the computer.

» Model-free refers to a physical process that can be observed, but where we do not have equations describing the evolution over time.
  • The behavior of a human or animal
  • The behavior of the climate
  • The behavior of a complex system such as a chemical plant

» Q-learning is often described as “model free” because it can be learned while observing a system.

» The resulting policy does not require a model:

  • \( A^\pi(s) = \arg\max_a \overline{Q}^n(s, a) \)
Q-learning

Notes

» Lookup table belief models are most popular, but do not scale (limit of 3 dimensions).

» Various smoothing strategies have been suggested (basically nonparametric statistics), but still limited to 3 dimensions.

» Need to be very careful with stepsizes. Q-learning is a form of approximate value iteration where the backward learning is slowed by the use of stepsizes.
Q-learning

Max operator bias:

» Second issue arises when there is randomness in the reward.
» Imagine that we are purchasing energy at a price $p_t$ which evolves randomly from one time period to the next.
» Imagine buying and selling energy using real time prices:
Max operator bias (cont’d)

» This introduces noise in $\hat{Q}^n(s^n, a^n)$:

$$
\hat{Q}^n(s^n, a^n) = C(s^n, a^n) + \gamma \max_{a'} \bar{Q}^{n-1}(s', a')
$$

$$
\bar{Q}^n(s^n, a^n) = (1 - \alpha_{n-1})\bar{Q}^{n-1}(s^n, a^n) + \alpha_{n-1}\hat{Q}^n(s^n, a^n)
$$

» Finding the max over a set of noisy estimates $\hat{Q}^n(s^n, a^n)$ introduces bias in the estimates $\bar{Q}^{n-1}(s', a')$. This bias can be quite large.

» Testing on roulette
Q-learning

- Roulette
  - Optimal solution is not to play – optimal value of game is zero
  - Q-learning over 10,000 iterations

Hint: Optimal $Q = 0$!
Q-learning

- Roulette
  - Optimal solution is not to play – optimal value of game is zero
  - Q-learning over 10,000 iterations
Q-learning

Roulette

- Optimal solution is not to play – optimal value of game is zero
Controlling Sample Bias in Q-learning through Bias-Corrected Q-Learning with Multistate Extension

Donghun Lee, Warren B. Powell, Member, IEEE,

Abstract—Q-learning is a sample-based model-free algorithm that solves Markov decision problems asymptotically, but in finite time it can perform poorly when random rewards have large variance. We pinpoint its cause to be the estimation bias due to the maximum operator in Q-learning algorithm, and present the evidence of max-operator bias in its Q value estimates. We then present an asymptotically optimal bias-correction strategy and construct bias-corrected Q-learning algorithm with asymptotic convergence properties as strong as those from Q-learning. We report the empirical performance of the bias-corrected Q-learning algorithm in select real-world problems: a multi-armed bandit problem and an electricity storage control simulation. The bias-corrected Q-learning algorithm with multistate extension is shown to be resistant to max-operator bias.

Index Terms—Q-learning, Bias Correction, Electricity Storage, Smart Grid

I. INTRODUCTION

Large inherent randomness in reward function poses major challenge in learning the optimal control policy in many classes of problems such as stochastic shortest path problem, multi-armed bandit problem, and more generally, Markov be seen as a blend of exact value iteration (VI) and stochastic approximation (SA) as follows:

\[
\tilde{Q}^n \leftarrow \tilde{C}(s^n, a^n) + \gamma \max_{a' \in A(s^{n+1})} \left( \tilde{Q}^{n-1}(s^{n+1}, a') \right)
\]

(1)

\[
\tilde{Q}^n(s^n, a^n) \leftarrow (1 - \alpha_{n-1}(s^n, a^n)) \tilde{Q}^{n-1}(s^n, a^n) + \alpha_{n-1}(s^n, a^n)
\]

(2)

where \((s^n, a^n)\) is a determined state-action pair, and \(a^{n+1}\) is a realization of random state transition due to taking action \(a^n\) in state \(s^n\) (we defer the detailed definition of other terms). Also, when the \(\tilde{Q}^n\) estimate is represented in tabular format, Q-learning enjoys asymptotic convergence properties with a mild set of technical assumptions as demonstrated in [2], [3], and [4]. The assumptions in [2] allow many stochastic models for \(C\) and \(s^{n+1}\) given \(s^n, a^n\) that can be applied to Q-learning with its convergence guarantee. Moreover, the asymptotic rate of convergence of Q-learning has been studied theoretically by a number of authors including [5], [6], and [7]. Thanks to its generally applicable set of assumptions and robust theoretical properties, Q-learning has been applied to a wide range of real-world problems including...

... ongoing research.
Approximate dynamic programming

Algorithms
Algorithms

- Approximate value iteration
  - Single-pass
    - Need to define policies for choosing states and actions
  - Double-pass with “discount” \( \lambda \)

- Approximate policy iteration

- Relationship to TD-learning
  - Approximate value iteration uses TD(0) updates.
  - Approximate policy iteration uses TD(1) updates.
Approximate value iteration

Step 1: Start with a pre-decision state $S_t^n$

Step 2: Solve the deterministic optimization using an approximate value function:

$$\hat{v}_t^n = \min_x \left( C_t(S_t^n, x_t) + \bar{V}_{t-1}^{n-1}(S_t^{M,x}(S_t^n, x_t)) \right)$$

to obtain $x^n$.

Step 3: Update the value function approximation

$$\bar{V}_{t-1}^n(S_t^{x,n}) = (1 - \alpha_{n-1})\bar{V}_{t-1}^{n-1}(S_t^{x,n}) + \alpha_{n-1}\hat{v}_t^n$$

Step 4: Obtain Monte Carlo sample of $W_t(\omega^m)$ and compute the next pre-decision state:

$$S_{t+1}^n = S_t^M(S_t^n, x_t^n, W_{t+1}(\omega^n))$$

Step 5: Return to step 1.
Approximate value iteration

Step 1: Start with a pre-decision state $S_t^n$

Step 2: Solve the deterministic optimization using an approximate value function:

$$\hat{v}^m = \min_x \left\{ C(S^m, x) + \sum_f \theta_f^{n-1} \phi_f (S^M(S^m, x)) \right\}$$

to obtain $x^n$.

Step 3: Update the value function approximation

$$\bar{V}_{t-1}^n(S_{t-1}^{x,n}) = (1 - \alpha_{n-1})\bar{V}_{t-1}^{n-1}(S_{t-1}^{x,n}) + \alpha_{n-1}\hat{v}_t^n$$

Step 4: Obtain Monte Carlo sample of $W_t(\omega^m)$ and compute the next pre-decision state:

$$S_{t+1}^n = S^M(S_t^n, x_t^n, W_{t+1}(\omega^n))$$

“Trajectory following”

Step 5: Return to step 1.
Approximate policy iteration

Step 1: Start with a pre-decision state $S^n_t$

Step 2: Inner loop: Do for $m=1,\ldots,M$:

Step 2a: Solve the deterministic optimization using an approximate value function:

$$
\hat{v}^m = \min_x \left( C(S^m, x) + V^{n-1}(S^M, x(S^m, x)) \right)
$$

to obtain $x^m$.

Step 2b: Update the value function approximation

$$
V^{n-1,m}(S^{x,m}) = (1 - \alpha_{m-1})V^{n-1,m-1}(S^{x,m}) + \alpha_{m-1}\hat{v}^m
$$

Step 2c: Obtain Monte Carlo sample of $W(\omega^m)$ and compute the next pre-decision state:

$$
S^{m+1} = S^M(S^m, x^m, W(\omega^m))
$$

Step 3: Update $V^n(S)$ using $V^{n-1,M}(S)$ and return to step 1.
Approximate policy iteration

Step 1: Start with a pre-decision state $S^n_t$

Step 2: Inner loop: Do for $m=1,\ldots,M$:

   Step 2a: Solve the deterministic optimization using an approximate value function:
   $$\hat{v}^m = \min_x \left( C(S^m, x) + \sum_f \theta_f^{n-1} \phi_f (S^M (S^m, x)) \right)$$
   to obtain $x^m$.

   Step 2b: Update the value function approximation using recursive least squares.

   Step 2c: Obtain Monte Carlo sample of $W(\omega^m)$ and compute the next pre-decision state:
   $$S^{m+1} = S^M (S^m, x^m, W(\omega^m))$$

Step 3: Update $\bar{\nu}^n (S)$ using $\bar{\nu}^{n-1, M} (S)$ and return to step 1.
Approximate dynamic programming

Nomadic trucker problem
Approximate value iteration
Fleet management problem

- Optimize the assignment of drivers to loads over time.
- Tremendous uncertainty in loads being called in
Approximate dynamic programming

Pre-decision state: we see the demands

\[ S_t = \left( \frac{TX}{t}, \hat{D}_t \right) \]
Approximate dynamic programming

We use initial value function approximations...

\[ S_t = \left( \frac{TX}{t}, \hat{D}_t \right) \]

© 2018 W.B. Powell
Approximate dynamic programming

... and make our first choice: $x^1$

$$S_t^x = \begin{pmatrix} NY \\ t+1 \end{pmatrix}$$

© 2018 W.B. Powell
Approximate dynamic programming

Update the value of being in Texas.

$V^0(MN) = 0$
$V^0(CO) = 0$
$V^0(CA) = 0$
$V^1(TX) = 450$
$V^0(NY) = 0$

$S_t^x = \left( \begin{array}{c} NY \\ t+1 \end{array} \right)$
Approximate dynamic programming

Now move to the next state, sample new demands and make a new decision

\[ S_{t+1} = \left( \frac{NY}{t+1}, \hat{D}_{t+1} \right) \]
Approximate dynamic programming

- Update value of being in NY

\[ \bar{V}^0(MN) = 0 \]
\[ \bar{V}^0(CO) = 0 \]
\[ \bar{V}^0(CA) = 0 \]
\[ \bar{V}^1(TX) = 450 \]

\[ S_{t+1}^x = \begin{pmatrix} \frac{CA}{t+2} \end{pmatrix} \]

© 2018 W.B. Powell
Move to California.

\[ S_{t+2} = \left( \begin{array}{c} CA \\ t+2 \end{array} \right), \hat{D}_{t+2} \]
Approximate dynamic programming

Make decision to return to TX and update value of being in CA

\[ S_{t+2} = \left( \begin{array}{c} CA \\ t + 2 \end{array} \right), \hat{D}_{t+2} \]
Approximate dynamic programming

- An updated value of being in TX

\[ S_{t+3} = \left( \frac{TX}{t+3}, \hat{D}_{t+3} \right) \]
Updating the value function:

Old value:
\[ \bar{V}^1 (TX) = $450 \]

New estimate:
\[ \hat{v}^2 (TX) = $800 \]

How do we merge old with new?
\[
\bar{V}^2 (TX) = (1 - \alpha)\bar{V}^1 (TX) + (\alpha)\hat{v}^2 (TX) \\
= (0.90)\$450 + (0.10)\$800 \\
= $485
\]
Approximate dynamic programming

An updated value of being in TX

\[ S_{t+3} = \left( \frac{TX}{t+3}, \hat{D}_{t+3} \right) \]
Approximate dynamic programming

Hierarchical learning
Hierarchical learning

Resource attribute: 
\[ a = "State" \text{ that the trucker is currently in} \]
Hierarchical learning

© 2019 Warren B. Powell
Hierarchical learning

© 2019 Warren B. Powell
Hierarchical learning
Hierarchical learning

Our optimization problem at time $t$ looks like:

$$V_t(S_t) = \max_x \left( C_t(S_t, x_t) + \sum_{a \in A} \bar{v}_{ta} \cdot R^x_{la} \right)$$

There are a lot of these attributes!

We had to develop novel machine learning strategies to estimate this function, since the attribute space was very large.
Hierarchical learning

- Different levels of aggregation:

\[
a = \begin{bmatrix}
\text{Time} & \text{Region Location} & \text{Region Domicile} & \text{Type} \\
\text{Time} & \text{Region Location} & \text{Region Domicile} & \text{Type} \\
\text{Time} & \text{Region Location} & \text{Region Domicile} & \text{Type} \\
\text{Time} & \text{Area Location} & \text{Type} & \text{Type}
\end{bmatrix}
\]

\[
|\mathcal{A}| \approx 3,293,136 \quad 33,264 \quad 5,544 \quad 672
\]
Hierarchical learning

Estimating value functions

» Most aggregate level

\[ \bar{v}^n([\text{Location}]) = (1 - \alpha)\bar{v}^{n-1}([\text{Location}]) + \alpha\hat{v}([\begin{bmatrix} \text{Location} \\ \text{Fleet} \\ \text{Domicile} \\ \text{DOThrs} \\ \text{DaysFromHome} \end{bmatrix}]) \]
Hierarchical learning

Estimating value functions

» Middle level of aggregation

\[ \bar{v}^n \left( \begin{bmatrix} Location \\ Fleet \end{bmatrix} \right) = (1 - \alpha) \bar{v}^{n-1} \left( \begin{bmatrix} Location \\ Fleet \end{bmatrix} \right) + \alpha \hat{v} \left( \begin{bmatrix} Location \\ Fleet \\ Domicile \\ DOThrs \\ DaysFromHome \end{bmatrix} \right) \]
Hierarchical learning

Estimating value functions

» Most disaggregate level

\[
\bar{v}^n \begin{bmatrix} Location \\ Fleet \\ Domicile \end{bmatrix} = (1 - \alpha) \bar{v}^{n-1} \begin{bmatrix} Location \\ Fleet \\ Domicile \end{bmatrix} + \alpha \hat{v} \begin{bmatrix} Location \\ Fleet \\ Domicile \\ DOThrs \\ DaysFromHome \end{bmatrix}
\]
Hierarchical learning

- Adaptive hierarchical estimation procedure developed as part of this project (George, Powell and Kulkarni, 2008)
  - Use weighted sum across different levels of aggregation.

\[
\bar{v}_a = \sum_g w_a^{(g)} \bar{v}_a^{(g)} \quad \sum_g w_a^{(g)} = 1
\]

where

\[
w_a^{(g)} \propto \left( \text{Var} \left( \bar{v}_a^{(g)} \right) + \left( \beta_a^{(g)} \right)^2 \right)^{-1}
\]

- Estimate of variance - \((\sigma_a^2)^{(g)}\)
- Estimate of bias

Both can be computed using simple recursive formulas.

Hierarchical learning

Hierarchical aggregation

Approximating a nonlinear function using two-levels of aggregation.
Hierarchical learning

© 2019 Warren B. Powell
Hierarchical learning
Hierarchical learning

- Hierarchical aggregation

Average weight on most disaggregate level

Average weight on most aggregate levels

Hierarchical aggregation

Aggregation level
Hierarchical learning

Notes:

» In the early iterations, we do not have enough data to provide estimates at the detail level.
» So, we put more weight on the most aggregate estimates.
» As the algorithm progresses and we gain more information, we can put more weight on the more disaggregate estimates.
» But the weights depend on how much data we have in different regions.
» This type of adaptive learning, from coarse-grained to fine-grained, is common across all learning problems in stochastic optimization.
Hierarchical learning

Hierarchical aggregation

Aggregate approximation shows faster initial convergence; disaggregate shows better asymptotic performance.
Hierarchical learning

Hierarchical aggregation

But adaptive weighting outperforms both. This hints at a strategy for adaptive learning.
The exploration-exploitation problem
Exploration vs. exploitation

© 2019 Warren B. Powell
What decision do we make?

» The one we think is best?

  • Exploitation

» Or do we make a decision just to try something and learn more about the result?

  • Exploration

» This is the reason that the “exploration vs. exploitation” problem is so well known in ADP/RL.
Exploration vs. exploitation

Pure exploitation
Exploration vs. exploitation

\[ w_a^0 \tilde{V}^0 (a_{11}) + w_a^1 \tilde{V}^1 \begin{pmatrix} a_{11} \\ a_{12} \end{pmatrix} + w_a^2 \tilde{V}^2 \begin{pmatrix} a_{11} \\ a_{12} \\ a_{13} \end{pmatrix} \]
Exploration vs. exploitation

Pure exploitation with generalized learning.
Exploration vs. exploitation

Notes

» This is a learning problem in the presence of a physical state (the location).

» Above, we are using a pure exploitation strategy, but with generalized learning (visiting one location teaches us about another location).

» An active area of research, with painfully little progress, is how to do active learning for state-dependent problems (in general) and more specifically, problems with a physical state.

» Note that we have avoided modeling the uncertainty in the value functions as part of the state variable. This is a historical oversight.
Hierarchical Knowledge Gradient for Sequential Sampling

Martijn R.K. Mes
Department of Operational Methods for Production and Logistics
University of Twente
Enschede, The Netherlands

Warren B. Powell
Department of Operations Research and Financial Engineering
Princeton University
Princeton, NJ 08544, USA

Peter I. Frazier
Department of Operations Research and Information Engineering
Cornell University
Ithaca, NY 14855, USA

Abstract

We propose a sequential sampling policy for noisy discrete global optimization and ranking and selection, in which we aim to efficiently explore a finite set of alternatives before selecting an alternative as best when exploration stops. Each alternative may be characterized by a multidimensional vector of categorical and numerical attributes and has independent normal rewards. We use a Bayesian probability model for the unknown reward of each alternative and follow a fully sequential sampling policy called the knowledge-gradient policy. This policy myopically optimizes the expected increment in the value of sampling information in each time period. We propose a hierarchical aggregation technique that uses the common features shared by alternatives to learn about

Optimal learning with a single physical state and hierarchical learning.

© 2019 Warren B. Powell
Exploration vs. exploitation

- Comparison of policies for pure learning problems
Exploration vs. exploitation

Notes:

» We have extensive research on pure learning problems.
» Very little has been done with problems that combine a belief state (which is the basis of any active learning problem) and a physical state.
» Central to the value of making a decision that is balancing the value of information is how this information is used in future decisions.
  • E.g. if we learn more about the cost $c_{ij}$ by going from $i$ to $j$, then this is only useful if we return to $i$ so that we can use this information.
  • For this reason, the presence of generalized learning architectures is key.
Exploration vs. exploitation

What about learning?

» “Active learning” with general dynamic programs is a very young field.

» Typically “exploration vs. exploitation” issues are solved with CFAs using approximate value functions as part of the estimate of the value of a decision.

Bayesian Exploration for Approximate Dynamic Programming
Ilya O. Ryzhov, Martijn R. K. Mes, Warren B. Powell, Gerald van den Berg

Abstract. Approximate dynamic programming (ADP) is a general methodological framework for multistage stochastic optimization problems in transportation, finance, energy, and other domains. We propose a new approach to the exploration/exploitation dilemma in ADP that leverages two important concepts from the optimal learning literature: first, we show how a Bayesian belief structure can be used to express uncertainty about the value function in ADP; second, we develop a new exploration strategy based on the concept of value of information and prove that it systematically explores the state space. An important advantage of our framework is that it can be integrated into both parametric and nonparametric value function approximations, which are widely used in practical implementations of ADP. We evaluate this strategy on a variety of distinct resource allocation problems and demonstrate that, although more computationally intensive, it is highly competitive against other exploration strategies.

1. Introduction

1. Commodity storage. A firm stores a commodity such as electricity or natural gas (Lai et al., 2010, Löhndorf and

© 2019 W.B. Powell
Exploration vs. exploitation

- Some numerical experiments from Ryzhov et al. paper.

Figure 2. Experimental Comparisons for Commodity Storage with Stochastic Price

Notes. (a) Online performance. (b) Offline performance.
Notes:

» The exploration vs. exploitation problem is well known in approximate dynamic programming, but lacks the elegant solution of pure learning problems.

» Most algorithms use fairly simple heuristics to balance exploration and exploitation.

» We have been pursuing research in optimal learning:
  • First paper: optimally sampling a function represented by a hierarchical belief model.
  • Second paper (in preparation): optimal learning with a physical state (the truck)

» Our research in optimal learning with a physical state is modest. The challenge is our ability to learn from one physical state, and generalize to others.

» Open question right now: how much does active learning contribute when there is a physical state? This is likely to be very problem-dependent.
From one truck to many
Schneider National
Optimizing fleets

- From one truck to many trucks
  - If there is one truck and N “states” (locations), then our dynamic program (post-decision state) has N states.
  - But what if there is more than one truck? Then we have to capture the state of the fleet.
The state of the fleet
The state of the fleet
The state of the fleet
The state of the fleet
The state of the fleet
The state of the fleet
The state of the fleet
The state of the fleet
The state of the fleet
Optimizing fleets
What if we have N > 1 trucks?
 No. trucks  | Locations | 1
| States | 

|
Locations
|

1


Number
Attribute
of
space
resources
1
1
1
100
1
1000
5
10
5
100
5
1000
50
10
50
100
50
1000

State space
1
100
1,000
2,002
91,962,520
8,416,958,750,200
12,565,671,261
13,419,107,273,154,600,000,000,000,000,000,000,000,000
109,740,941,767,311,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000

Real problems: 500 to 5,000 trucks, attribute space 50,000 up to 10
© 2019 Warren B. Powell

attributes.


Optimizing fleets

The pre-decision state: drivers and loads

$$S_t = (R_t, D_t)$$
Optimizing fleets

The post-decision state - drivers and loads after a decision is made:

\[ S^x_{t} = S^{M,x}_{t} (S_t, x_t) \]
Optimizing fleets

The transition: Adding new information

\[ S_t^x, S_{t+1} = S_{M,W}^x(S_t^x, W_{t+1}) \]

\[ W_{t+1} = (\hat{R}_{t+1}, \hat{D}_{t+1}) \]
Optimizing fleets

The next pre-decision state

\[ S_{t+1} \]
Optimizing fleets

Assignment network

» Capture the value of downstream driver.

\[ a^M(a_3, d_1) = \text{Attribute vector of driver in the future given a decision } d. \]

» Add this value to the assignment arc.
Optimizing fleets

- The assignment problem

  - We now have a basic assignment problem, but where we have to capture the downstream value of a truck:

\[
X_t^n(S_t) = \text{arg max}_{x_t \in \mathcal{X}_t} \left( \sum_{a \in \mathcal{A}} \sum_{d \in \mathcal{D}} c_{lad} x_{lad} + \gamma \sum_{a' \in \mathcal{A}} \tilde{v}_{lad} \cdot \sum_{a \in \mathcal{A}} \sum_{d \in \mathcal{D}} \delta_{a'}(a, d) x_{lad} \right)
\]

\[
= \text{arg max}_{x_t \in \mathcal{X}_t(\omega)} \sum_{a \in \mathcal{A}} \sum_{d \in \mathcal{D}} \left( c_{lad} + \gamma \sum_{a' \in \mathcal{A}} \tilde{v}_{lad} \delta_{a'}(a, d) \right) x_{lad}.
\]

Recognizing that \(\sum_{a' \in \mathcal{A}} \delta_{a'}(a, d) = \delta_{a^M(a_t, d_t)}(a, d) = 1\), we can write Equation (14) as

\[
X_t^n(S_t) = \text{arg max}_{x_t \in \mathcal{X}_t(\omega)} \sum_{a \in \mathcal{A}} \sum_{d \in \mathcal{D}} \left( c_{lad} + \gamma \tilde{v}^{n-1}_{l, a^M(a, d)} \right) x_{lad}.
\]

\[
\delta_{a'}(a, d) = \begin{cases} 
1, & \text{if } a^M(a, d) = a', \\
0, & \text{otherwise}.
\end{cases}
\]
Optimizing fleets

Finding the marginal value of a driver:
  » Dual variables
    • Can provide unreliable estimates.
    • Need to get the marginal value of drivers who are not actually there.
  » Numerical derivatives:
Optimizing fleets

Step 1: Start with a pre-decision state $S^n_t$

Step 2: Solve the deterministic optimization using an approximate value function:

$$\hat{v}^n = \min_x \left(C_x(S^n_t, x) + \bar{V}^{n-1}_t(S^{M,x}(S^n_t, x))\right)$$

to obtain $x^n$.

Step 3: Update the value function approximation

$$\bar{V}^n_{t-1}(S^{x,n}_{t-1}) = (1 - \alpha_{n-1})\bar{V}^{n-1}_{t-1}(S^{x,n}_{t-1}) + \alpha_{n-1}\hat{v}^n$$

Step 4: Obtain Monte Carlo sample of $W_t(\omega^n)$ and compute the next pre-decision state:

$$S^n_{t+1} = S^{M}(S^n_t, x^n_t, W_{t+1}(\omega^n))$$

Step 5: Return to step 1.

“on policy learning”
Optimizing fleets
Optimizing fleets
Optimizing fleets
Approximate dynamic programming

... a typical performance graph.
Approximate dynamic programming

Stepsizes for forward APD
Stepsizes

- Approximate value iteration requires updates of the form:

\[
V_{t-1}^n(S_{t-1}^x) = (1 - \alpha_{n-1})V_{t-1}^{n-1}(S_{t-1}^x) + \alpha_{n-1}\hat{y}_t^n
\]

- The stepsizes are
  - "Learning rate"
  - "Smoothing factor"
Stepsizes

- Single state, single action Markov chain
  - Updating – receives reward=1 at last node. All other rewards = 0.
  - Same as adding up random rewards with mean of 1. Noise may be zero, or quite high.

![Diagram of a Markov chain with states 0 to 5 and rewards at each state.]

<table>
<thead>
<tr>
<th>Iteration</th>
<th>$\hat{V}_0$</th>
<th>$\hat{V}_1$</th>
<th>$\hat{V}_2$</th>
<th>$\hat{V}_3$</th>
<th>$\hat{V}_4$</th>
<th>$\hat{V}_5$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>1</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>2</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>3</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.167</td>
<td>0.500</td>
<td>0.667</td>
</tr>
<tr>
<td>4</td>
<td>0.000</td>
<td>0.000</td>
<td>0.042</td>
<td>0.292</td>
<td>0.667</td>
<td>0.750</td>
</tr>
<tr>
<td>5</td>
<td>0.008</td>
<td>0.042</td>
<td>0.092</td>
<td>0.383</td>
<td>0.800</td>
<td>0.833</td>
</tr>
<tr>
<td>6</td>
<td>0.022</td>
<td>0.092</td>
<td>0.140</td>
<td>0.453</td>
<td>0.800</td>
<td>0.833</td>
</tr>
<tr>
<td>7</td>
<td>0.039</td>
<td>0.140</td>
<td>0.185</td>
<td>0.507</td>
<td>0.833</td>
<td>0.857</td>
</tr>
<tr>
<td>8</td>
<td>0.057</td>
<td>0.185</td>
<td>0.225</td>
<td>0.551</td>
<td>0.857</td>
<td>0.875</td>
</tr>
<tr>
<td>9</td>
<td>0.076</td>
<td>0.225</td>
<td>0.261</td>
<td>0.587</td>
<td>0.875</td>
<td>0.889</td>
</tr>
<tr>
<td>10</td>
<td>0.095</td>
<td>0.261</td>
<td>0.294</td>
<td>0.587</td>
<td>0.889</td>
<td>0.900</td>
</tr>
</tbody>
</table>

Table 17.1 Effect of stepsize on backward learning
Stepsizes

- Bound on performance using $1/n$:

Single state, single action

$$V = \sum_{n=0}^{\infty} \gamma^n c = \frac{c}{1-\gamma}$$

$$\hat{v}^{n+1} = c + \gamma \hat{v}^n$$

$$\bar{v}^{n+1} = (1 - \alpha_n) \bar{v}^n + \alpha_n \hat{v}^{n+1}$$

$$\nu^L(n) = \frac{c}{1-\gamma} \left(1 - \left(\frac{1}{1+n}\right)^{1-\gamma}\right)$$

© 2019 Warren B. Powell
Stepsizes

**Bias-adjusted Kalman filter (BAKF)**

Where:

\[ \alpha_n = 1 - \frac{\sigma^2}{(1 + \lambda^{n-1})\sigma^2 + (\beta^n)^2} \]

\[ \lambda^n = (1 - \alpha_n)^2 \lambda^{n-1} + (\alpha_n)^2 \]

- As \( \sigma^2 \) increases, stepsize decreases toward \( 1/n \)
- As \( \beta^n \) increases, stepsize increases toward 1.
- At all times, \( \alpha_n \geq \frac{1}{n} \)

© 2019 Warren B. Powell
Stepsizes

The bias-adjusted Kalman filter

![Graph showing observed values and stepsize rules.]

© 2019 Warren B. Powell
Stepsizes

The bias-adjusted Kalman filter
Stepsizes

Notes:

» Because $\beta$ and $\sigma^2$ have to be estimated from data, it is a stochastic stepsize policy. The challenge is estimating $\beta$.

» This stepsize rule is designed for a nonstationary time series. It does not recognize the feedback that arises in approximate value iteration.
Optimizing fleets

- The effect of stepsizes

Figure 5: Average Value Function When We Use Forward and Backward Passes, Numerical Derivatives and Dual Variables, and the OSA Stepsize or the McClain Stepsize

© 2019 Warren B. Powell
Schneider National case study
We were able to calibrate our ADP model very closely to the behavior of the company, which helped to build confidence.
The adaptive learning of value functions produced results that more accurately matched historical performance.

© 2019 Warren B. Powell
ADP for trucking

Notes:

» We can use the value functions to estimate the marginal value of drivers whose home is in a particular region.

» In the next slides, we are going to show two ways to estimate these values:

- In the first, we are going to run a series of (very expensive) simulations where we hire an increasing number of drivers in a region. Each time, we have to reoptimize the entire fleet over many iterations. We would have to repeat this for each possible home location.
- In the second, we are going to estimate the marginal value by using the value function approximations to estimate the value of drivers in each region, from one run of the model.
Case study: truckload trucking

Simulation objective function

# of drivers with attribute a

© 2019 Warren B. Powell
Case study: truckload trucking

simulation objective function

# of drivers with attribute a

© 2019 Warren B. Powell
This shows that the value functions provide reasonable approximations of the
Case study: truckload trucking

Where to add drivers:
Case study: truckload trucking

Where to reduce drivers:
Approximate dynamic programming

Exploiting monotonicity
Exploiting monotonicity

Monotonicity in dynamic programming

» There are many problems where the value function increases (or decreases) monotonically with all dimensions of a state variable:

» Operations research
  • Optimal equipment replacement – Replace when a measure of deterioration exceeds some point
  • Dispatching customers in a queue – Costs increase monotonically with the number of customers in the queue. Dispatch when the queue is over some number.
Exploiting monotonicity

Monotonicity in dynamic programming (cont’d)

» Energy

• Buy when electricity price is below one number, sell when above another number:

- Value of energy in storage increases with amount being held (and perhaps price, speed of wind, demand, …)
Exploiting monotonicity

Monotonicity in dynamic programming (cont’d)

» Health
  • Dosage of a diabetes drug increases with blood sugar.
  • Dosage of statins (for reducing cholesterol) increase as the cholesterol level increases (and also increases with age, weight).

» Finance
  • The value of holding cash in a mutual fund increases with the amount of redemptions, stock indices, and interest rates.
  • The value of holding an option increases with price and volatility.
Exploiting monotonicity

Bid is placed at 1pm, consisting of charge and discharge prices between 2pm and 3pm.
Exploiting monotonicity

The exact value function

© 2019 Warren B. Powell
Exploiting monotonicity

Approximate value function without monotonicity
Exploiting monotonicity

- Maintaining monotonicity

= observations

0

5

10

© 2019 Warren B. Powell
Exploiting monotonicity

- Maintaining monotonicity

= observations

0

5

10

© 2019 Warren B. Powell
Exploiting monotonicity

- Maintaining monotonicity

\[ \bigcirc \bigcirc = \text{observations} \]

\[ \begin{array}{c}
0 \\
5 \\
10 \\
\end{array} \]

\[ \Pi_M \]

© 2019 Warren B. Powell
Exploiting monotonicity

Maintaining monotonicity

= observations

\[ \Pi_M \]

© 2019 Warren B. Powell
Exploiting monotonicity
Exploiting monotonicity

Benchmarking

» M-ADP and AVI (no. of iterations)
Exploiting monotonicity

Observations

» Initial experiments using various machine learning approximations produced poor results (60-80 percent of optimal)

» Vanilla lookup table works poorly (within computing budgets that spanned days)

» Lookup table with monotonicity worked quite well

» We have run this algorithm on problems with up to 7 dimensions (but that is our limit)
Backward ADP

Backward ADP for a clinical trial problem

» Problem is to learn the value of a new drug within a budget of patients to be tested.

» Backward MDP required 268-485 hours.

» Forward ADP exploiting monotonicity (we will cover this later) required 18-30 hours.

» Backward ADP required 20 minutes, with a solution that was 1.2 percent within optimal.
Approximate dynamic programming

Driverless EV problem
Driverless EV optimization

Current Uber logic:

» Show nearest 8 drivers.
» Contact closest driver to confirm assignment.
» If driver does not confirm, contact second closest driver.

Limitations:

» Ignores potential future opportunities for each driver.
Driverless EV optimization

Assigning car in less dense area allows closer car to handle potential demands in more dense areas.

Closest car...but it moves car away from busy downtown area and strands other car in low density area.
Driverless EV optimization
Driverless EV optimization

The central operator should think about:

- Should it accept this trip?
- What is the best car to assign to the trip considering
  - The type of car
  - The charge level of the battery

If it doesn’t assign a car to a trip, should it:

- Sit where it is?
- Reposition to a better location?
- Recharge the battery?
- Move to a parking facility?
Driverless EV optimization
Driverless EV optimization
Driverless EV optimization
The assignment of cars to riders evolves over time, with new riders arriving, along with updates of cars available.
Driverless EV optimization

1) Policy function approximations (PFAs)
   » Lookup tables, rules, parametric/nonparametric functions

2) Cost function approximation (CFAs)
   » \( X_{CFA}^{CFA}(S_t | \theta) = \arg \max_{x_t \in \tilde{x}_t^\pi(\theta)} \overline{C}^\pi(S_t, x_t | \theta) \)

3) Policies based on value function approximations (VFAs)
   » \( X_{VFA}^{VFA}(S_t) = \arg \max_{x_t} \left( C(S_t, x_t) + \overline{V}_t^x\left(S_t^x(S_t, x_t)\right) \right) \)

4) Direct lookahead policies (DLAs)
   » Deterministic lookahead/rolling horizon proc./model predictive control
     \( X_{t}^{LA-D}(S_t) = \arg \max_{\tilde{x}_t, \ldots, \tilde{x}_{t,T}} C(\tilde{S}_t, \tilde{x}_t) + \sum_{t'=t+1} C(\tilde{S}_{t'}, \tilde{x}_{t'}) \)
   » Chance constrained programming
     \( P[A_t x_t \leq f(W)] \leq 1 - \delta \)
   » Stochastic lookahead /stochastic prog/Monte Carlo tree search
     \( X_{t}^{LA-S}(S_t) = \arg \max_{\tilde{x}_t, \ldots, \tilde{x}_{t,T}} C(\tilde{S}_t, \tilde{x}_t) + \sum_{\tilde{\omega} \in \Omega_t} p(\tilde{\omega}) \sum_{t'=t+1} C(\tilde{S}_{t'}, \tilde{x}_{t'}(\tilde{\omega})) \)
   » “Robust optimization”
     \( X_{t}^{LA-RO}(S_t) = \arg \max_{\tilde{x}_t, \ldots, \tilde{x}_{t,T}} \min_{w \in W(\theta)} C(\tilde{S}_t, \tilde{x}_t) + \sum_{t'=t+1} C(\tilde{S}_{t'}, w, \tilde{x}_{t'}(w)) \)

Lookahead approximations

© 2019 Warren B. Powell
Policies based on value function approximations

\[ X_t^{VFA} (S^n_t) = \arg \max_{x_t} \left( C(S^n_t, x_t) + V_t^{n-1} \left( S_t^{M,x} (S^n_t, x_t) \right) \right) \]

- **Contribution for taking decision** \(x_t\) **from state** \(S^n_t\)
- **Expected Value of the Post-decision State**

- Exact value functions are rare:
  - Discrete states and actions, with a computable one-step transition matrix.

- Approximate value functions are defined by:
  - Approximation architecture
    - Linear, nonlinear separable, nonlinear
  - Learning strategy
    - Pure forward pass, two-pass
Driverless EV optimization

\[ v(a'_1) \]

\[ v(a''_1) \]
Driverless EV optimization
Driverless EV optimization
Driverless EV optimization
Driverless EV optimization
Driverless EV optimization

New car assignments:

New future values
Driverless EV optimization
Driverless EV optimization

New solution

\[ a_1, a_2, a_3, a_4 \]

\[ v(a_1'), v(a_1''), v(a_2'), v(a_2''), v(a_3'), v(a_3''), v(a_4'), v(a_4'') \]

© 2019 Warren B. Powell
Driverless EV optimization

\[ \text{Difference} = \hat{v}(a_1) \]

\[ v(a_1') - v(a_2') \]

\[ v(a_2') - v(a_3') \]

\[ v(a_3') - v(a_4') \]
Difference in assignment costs

\[ \hat{v}(a_1) - v(a_2) \]

Changes in future values

\[ v(a_1) - v(a_2) \]

\[ v(a_2) - v(a_3) \]

\[ v(a_3) - v(a_4) \]
Driverless EV optimization

Assignment network

» Capture the value of downstream driver.

» $a^M(a_3,d_1) =$ Attribute vector of driver in the future given a decision $d$.

» Add this value to the assignment arc.
Driverless EV optimization

Finding the marginal value of a driver:

» Dual variables
  • Can provide unreliable estimates.
  • Need to get the marginal value of drivers who are not actually there.

» Numerical derivatives:
Driverless EV optimization

Estimating average values:

\[ \bar{v}^n(a) = (1 - \alpha)\bar{v}^{n-1}(a) + (\alpha)\hat{v}^n(a) \]

- Old estimate of the value of a car with attribute \(a\)
- New estimate of the value of a car with attribute \(a\)
- Attribute of car
Driverless EV optimization

We had to use two statistical techniques to accelerate learning:

» Monotonicity with respect to time and charge level.

» Hierarchical aggregation (just as we did with the trucking example).
Driverless fleets of EVs using ADP

- The value of a vehicle in the future
  - Value function approximation captures charge level, as well as time and location.
  - Hierarchical aggregation accelerated the learning process
Driverless EV optimization

Step 1: Start with a pre-decision state $S^n_t$

Step 2: Solve the deterministic optimization using an approximate value function:

$$\hat{v}^n_t = \min_x \left( C_t(S^n_t, x_t) + \bar{V}^{n-1} (S^M, x) (S^n_t, x_t) \right)$$

to obtain $x^n$.

Step 3: Update the value function approximation

$$\bar{V}^n_{t-1}(S^{x,n}_{t-1}) = (1 - \alpha_{n-1})\bar{V}^{n-1}_{t-1}(S^{x,n}_{t-1}) + \alpha_{n-1}\hat{v}^n_t$$

Step 4: Obtain Monte Carlo sample of $W_t(\omega^n)$ and compute the next pre-decision state:

$$S^n_{t+1} = S^M (S^n_t, x^n_t, W_{t+1}(\omega^n))$$

Step 5: Return to step 1.

“on policy learning”

© 2019 Warren B. Powell
Driverless EV optimization

- 22000 zones
- 2000 cars
- Battery capacity: 50KWh
- 31560 Trips

© 2019 Warren B. Powell
Driverless EV optimization

- No aggregation – (!!) Solution gets worse!
Driverless EV optimization

- Three levels of aggregation - Better

Profit versus number of iterations

© 2019 Warren B. Powell
Driverless EV optimization

- Five levels of aggregation – old VFAs

Profit versus number of iterations
Driverless EV optimization

- Five levels of aggregation – new VFAs

Profit versus number of iterations
Driverless EV optimization

- Trip requests over time
  » Challenge is to recharge during off-peak periods
Driverless fleets of EVs using ADP

- Heuristic dispatch vs. ADP-based policies
  - Effect of value function approximations on recharging

Heuristic recharging logic

Recharging controlled by approximate dynamic programming

- Trip requests
- Actual trips
- No recharging during peak period
- Recharging during peak period

Total battery charge

Peak
The economics of driverless fleets

We can simulate different fleet sizes and battery capacities, properly modeling recharging behaviors given battery capacity.
Approximate dynamic programming

Exploiting convexity in an inventory problem
Exploiting convexity

- Single inventory problem
  - With pre-decision state:
    \[
    x_t^n = \arg \max_{x_t \in X} \left( C(S_t^n, x_t) + \gamma \mathbb{E}\left\{ \tilde{V}_{t+1}^{n-1} \left( S_{t+1}(S_t^n, x_t, W_{t+1}) \right) \mid S_t \right\} \right)
    \]
  - With post-decision state:
    \[
    x_t^n = \arg \max_{x_t \in X} \left( C(S_t^n, x_t) + \gamma \tilde{V}_t^{x,n-1} \left( S_t^{x \times} (S_t^n, x_t) \right) \right)
    \]
Exploiting convexity

Updating strategies

» Forward pass (approximate value iteration) updating as we go (also known as TD(0)):

• Compute the value of being in a state by “bootstrapping” the downstream value function approximation:

\[
\hat{V}_t^n(S_t^n) = \max_{x_t \in X} \left( C(S_t^n, x_t) + \gamma \hat{V}_t^{x,n-1}(S_t^x(S_t^n, x_t)) \right)
\]

• Now compute the marginal value using

\[
\hat{\nu}_t^n = \frac{\hat{V}_t^n(S_t^n + \delta) - \hat{V}_t^n(S_t^n)}{\delta}
\]

• Use this to update the piecewise linear value functions.
Exploiting convexity

Updating strategies

» Double pass:
  • Perform forward pass simulating the policy for $t = 0, ..., T$:
    \[
    x_t^n = \arg \max_{x_t \in \mathcal{X}} \left( C(S_t^n, x_t) + \gamma \bar{V}_{t}^{x,n-1} \left( S_t^x(S_t^n, x_t) \right) \right)
    \]
  • Now perturb state by $\delta$ and solve again:
    \[
    x_t^{+\delta,n} = \arg \max_{x_t \in \mathcal{X}} \left( C(S_t^n + \delta, x_t) + \gamma \bar{V}_{t}^{x,n-1} \left( S_t^x(S_t^n + \delta, x_t) \right) \right)
    \]
    Update $S_{t+1}^n = S_t^n + x_t^n$ (remember $x_t^n$ may be negative). Store $S_t^n$ as you proceed. Also store incremental cost
    \[
    \delta C_t^n = C(S_t^n + \delta, x_t^{+\delta,n}) - C(S_t^n, x_t^n)
    \]
  • Perform backward pass, computing marginal values (and performing updates):
    \[
    \hat{v}_t^n = \delta C(S_t^n, x_t^n) + \hat{v}_{t+1}^n
    \]
    \[
    \bar{V}_{t-1}^n(S_{t-1}^n) \leftarrow U^V (\bar{V}_{t-1}^{n-1}(S_{t-1}^n), S_{t-1}^n, \hat{v}_t^n)
    \]
Exploiting convexity

- We update the piecewise linear value functions by computing estimates of slopes using a backward pass:

  The cost along the marginal path is the derivative of the simulation with respect to the flow perturbation.

- The cost along the marginal path is the derivative of the simulation with respect to the flow perturbation.
Exploiting convexity

- From pre- to post-decision state

  » Get marginal value of inventory, $\hat{v}_t^n$, at time $t$ for inventory level $s_t^n$ during iteration $n$.

  » Use this to update the value function around the previous post-decision state $s_{t-1}^{x,n}$ to obtain $\bar{V}_{t-1}^{x,n}(s)$. 
Exploiting convexity

Updating strategies

» The choice between a pure forward pass (TD(0)) versus a double pass (TD(1)) is highly problem dependent. Every problem has a natural “horizon.”

» We have found that in our fleet management problems, the pure forward pass works fine and is much easier to implement.

» For energy storage problems, it is essential that we use a double pass, since a decision at time $t$ can have an impact hundreds of time periods into the future.
Exploiting convexity

- It is important to maintain concavity:

\[ k \text{tv} + \ast k \text{tv} \]
Exploiting convexity

A concave function…

… has monotonically decreasing slopes. But updating the function with a stochastic gradient may violate this property.
Exploiting convexity
Exploiting convexity
Exploiting convexity

\[ \tilde{v}_it^n = (1 - \alpha)\tilde{v}_it^{n-1} + \alpha\tilde{v}_it^n \]
Exploiting convexity

\[ \bar{v}_{it}^n = (1 - \alpha)\bar{v}_{it}^{n-1} + \alpha \hat{v}_{it}^n \]
Exploiting convexity

$$\overline{v}_{it}^n = (1 - \alpha)\overline{v}_{it}^{n-1} + \alpha \hat{v}_{it}^n$$

$$v_0$$

$$v_1$$

$$v_2$$

$$u_0$$

$$u_1$$

$$R_{it}^k$$

$$u_2$$
Exploiting convexity

Ways for maintaining concavity (monotonicity in the slopes):

» CAVE algorithm – Use updated slope at one point to update over a range $+/−\delta^n$ which shrinks with iterations (shrinking factor is tunable parameter). Expand $\delta$ if needed to ensure concavity is maintained.
  • Works well! But we could never prove convergence.

» Leveling algorithm – Update at a point $x^n$.
  • Force monotonicity in slopes by increasing/decreasing slopes farther from $x^n$ as needed.
  • Works fine, without tunable parameters

» SPAR algorithm – Perform nearest point projection onto space of monotone functions.
  • Nice theoretical convergence proof – could never get it to work.
Exploiting convexity

Derivatives are used to estimate a piecewise linear approximation
Exploiting convexity

With luck, your objective function improves
Exploiting convexity

Testing on a time-dependent, deterministic problem

- Blue = optimal (found by solving a single linear program over entire horizon)
- Black = ADP solution
Exploiting convexity

Stochastic, time-dependent problems
Exploiting convexity

Stochastic, time-dependent problems
Approximate dynamic programming

Resource allocation (freight cars) for two-stage problem
Forecasts of Car Demands
Two-stage problems

Two-stage resource allocation under uncertainty
Optimization frameworks

We obtain piecewise linear recourse functions for each region.
Optimization frameworks

- The function is piecewise linear on the integers.

![Graph showing profits vs. number of vehicles at a location.](image)
Optimization frameworks

Using standard network transformation:

Each link captures the marginal reward of an additional car.
Two-stage problems
Two-stage problems
Two-stage problems

$R^n_1 \rightarrow$

$R^n_2 \rightarrow$

$R^n_3 \rightarrow$

$R^n_4 \rightarrow$

$R^n_5 \rightarrow$

© 2004 Warren B. Powell
Two-stage problems

We estimate the functions by sampling from our distributions.

Marginal value:

\[ v_1(\omega^n) \rightarrow R_1^n \rightarrow D_1(\omega^n) \]
\[ v_2(\omega^n) \rightarrow R_2^n \rightarrow D_2(w^n) \]
\[ v_3(\omega^n) \rightarrow R_3^n \rightarrow \cdots \]
\[ v_4(\omega^n) \rightarrow R_4^n \rightarrow \cdots \]
\[ v_5(\omega^n) \rightarrow R_5^n \rightarrow D_C(w^n) \]
Two-stage problems

- The time $t$ subproblem:

$$\tilde{V}_{ta}^n (R_{t1}, R_{t2}, R_{t3})$$

Gradients:

$$(\hat{v}_{t1}^{-}, \hat{v}_{t1}^{+})$$

$$(\hat{v}_{t2}^{-}, \hat{v}_{t2}^{+})$$

$$(\hat{v}_{t3}^{-}, \hat{v}_{t3}^{+})$$
Two-stage problems

Left and right gradients are found by solving flow augmenting path problems.

Gradients:

The right derivative (the value of one more unit of that resource) is a flow augmenting path from that node to the supersink.
Two-stage problems

- Left and right derivatives are used to build up a nonlinear approximation of the subproblem.
Two-stage problems

Left and right derivatives are used to build up a nonlinear approximation of the subproblem.
Two-stage problems

Each iteration adds new segments, as well as refining old ones.
What we can prove

Convergence proven assuming that all points are sampled infinitely often:

Convergence also proven for variations which maintain concavity:

But so far, our fastest convergence is with the original version, called the CAVE algorithm, for which convergence has not been proven.
Approximate dynamic programming

Blood management problem
Management multiple resources
Blood management

- Managing blood inventories over time

Week 0

Week 1

Week 2

Week 3

© 2018 Warren B. Powell
\[
\begin{align*}
\mathcal{R}_t & \quad \mathcal{R}^x_t & \quad \hat{\mathcal{R}}_{t+1, AB+} \\
\mathcal{R}_{t,(AB+,0)} & \quad \text{AB+,0} & \quad \text{AB+,0} \\
\mathcal{R}_{t,(AB+,1)} & \quad \text{AB+,1} & \quad \text{AB+,1} \\
\mathcal{R}_{t,(AB+,2)} & \quad \text{AB+,2} & \quad \text{AB+,2} \\
\mathcal{R}_{t,(AB+,3)} & \quad \text{AB+,3} & \quad \text{AB+,3} \\
\mathcal{R}_{t,(O-,0)} & \quad \text{O-,0} & \quad \hat{\mathcal{R}}_{t+1, O-} \\
\mathcal{R}_{t,(O-,1)} & \quad \text{O-,1} & \quad \text{O-,1} \\
\mathcal{R}_{t,(O-,2)} & \quad \text{O-,2} & \quad \text{O-,2} \\
\mathcal{R}_{t,(O-,3)} & \quad \text{O-,3} & \quad \text{O-,3} \\
\end{align*}
\]

© 2018 Warren B. Powell
Solve this as a linear program.
Dual variables give value additional unit of blood.
Updating the value function approximation

- Estimate the gradient at $R_t^n$
Updating the value function approximation

Update the value function at $R_{t-1}^{x,n}$
Updating the value function approximation

Update the value function at $R_{t-1}^{x,n}$
Updating the value function approximation

Update the value function at $R_{t-1}^{x,n}$
Updating the value function approximation

Notes:

» We get a marginal value for each supply node.
» These are provided automatically by linear programming solvers.
» Be careful when the supply at the node = 0. While we would like to have the marginal value of one more, if we use the marginal values produced by the linear programming code, it might be the slope to the left (where the “supply” would be negative).
Exploiting concavity

- Derivatives are used to estimate a piecewise linear approximation

\[ \bar{V}_t(R_t) \]

\[ R_t \]
Approximate value iteration

Step 1: Start with a pre-decision state $S_t^n$

Step 2: Solve the deterministic optimization using an approximate value function:
$$\max_x \left( C_t(S_t^n, x_t) + \bar{V}_t^{n-1}(S_t^n, x_t) \right)$$
to obtain $x_t^n$ and dual variables $(\hat{\nu}_t^n)$.

Step 3: Update the value function approximation
$$\bar{V}_{t-1}^n(S_{t-1}^{x,n}) = (1 - \alpha_{n-1})\bar{V}_{t-1}^{n-1}(S_{t-1}^{x,n}) + \alpha_{n-1}\hat{\nu}_t^n$$

Step 4: Obtain Monte Carlo sample of $W_t(\omega^n)$ and compute the next pre-decision state:
$$S_t^{n+1} = S_t^M(S_t^n, x_t^n, W_{t+1}(\omega^n))$$

Step 5: Return to step 1.
Approximate value iteration

Step 1: Start with a pre-decision state $S_t^n$

Step 2: Solve the deterministic optimization using an approximate value function:
$$\max_x \left( C_t(S_t^n, x_t) + \bar{V}_t^{n-1}(S_t^M, x(S_t^n, x_t)) \right)$$
to obtain $x_t^n$ and dual variables $(\hat{v}_t^n)$.

Deterministic optimization
Approximate value iteration

Step 1: Start with a pre-decision state $S_t^n$

Step 2: Solve the deterministic optimization using an approximate value function:

$$\max_x \left( C_t(S_t^n, x_t) + \bar{\eta}_t^{n-1}(S_t^M, x(S_t^n, x_t)) \right)$$

to obtain $x_t^n$ and dual variables ($\hat{\nu}_t^n$).
Approximate value iteration

Step 1: Start with a pre-decision state $S^n_t$

Step 2: Solve the deterministic optimization using an approximate value function:
$$
\max_x \left( C_t(S^n_t, x_t) + \bar{V}_t^{n-1}(S^{M_x}(S^n_t, x_t)) \right)
$$
to obtain $x^n_t$ and dual variables $(\hat{\nu}^n_i)$.

Step 3: Update the value function approximation

Deterministic optimization
Recursive statistics

© 2018 Warren B. Powell
Iterative learning
Iterative learning
Iterative learning
Approximate dynamic programming

Grid level storage
- Imagine 25 large storage devices spread around the PJM grid:
Optimizing battery storage
Value function approximations
Value function approximations
Value function approximations

© 2019 Warren B. Powell
Value function approximations

© 2019 Warren B. Powell
Exploiting concavity

- Derivatives are used to estimate a piecewise linear approximation
Approximate dynamic programming

… a typical performance graph.
Grid level storage control

- Without storage
Grid level storage control

- With storage