Reinforcement Learning

Generalization and Function Approximation

Subramanian Ramamoorthy
School of Informatics

28 February, 2012
Function Approximation in RL: Why?

Tabular $V^\pi$:

What should we do when the state spaces are very large and this becomes computationally expensive?

Represent $V^\pi$ as a surface, use function approximation.
Generalization

- Large numbers of states/actions?
- Continuously-valued states/actions?
- Most states never experienced exactly before
  Memory
  Time
  Data

… how experience with small part of state space is used to produce good behaviour over large part of state space
Rich Toolbox

• Neural networks, decision trees, multivariate regression ...
• Use of statistical clustering for state aggregation
• Can swap in/out your favourite function approximation method as long as they can deal with:
  – learning while interacting, online
  – non-stationarity induced by policy changes
• So, combining gradient descent methods with RL requires care

• May use generalization methods to approximate states, actions, value functions, Q-value functions, policies
Value Prediction with FA

As usual: Policy Evaluation (the prediction problem):
for a given policy $\pi$, compute the state-value function $V^\pi$

The value function estimate at time $t$, $V_t$, depends on a parameter vector $\tilde{\theta}$ and only the parameter vector is updated.

e.g., $\theta_t$ could be the vector of connection weights of a neural network.
Adapt Supervised Learning Algorithms

Training Info = desired (target) outputs

Inputs ➔ Parameterized Function ➔ Outputs

Training example = \{input, target output\}

Error = (target output - actual output)
Backups as Training Examples

e.g., the TD(0) backup:

\[
V(s_t) \leftarrow V(s_t) + \alpha [r_{t+1} + \gamma V(s_{t+1}) - V(s_t)]
\]

As a training example:

\{ \text{description of } s_t, \ r_{t+1} + \gamma V(s_{t+1}) \}

input \quad \text{target output}
Feature Vectors

\[ \phi_s = \begin{pmatrix} \text{redness} \\ \text{greenness} \\ \text{roundness} \\ \text{starness} \\ \text{size} \end{pmatrix} = \begin{pmatrix} 25 \\ 3 \\ 2 \\ 15 \\ 25 \end{pmatrix} \]

- “Redness” = say closeness to 11111111000000000000000000 (RGB, R=255, G=0, B=0)
- “Roundness” = say distance of points from enclosing circle
- “Starness” = say some combination of number of points, template matching to a star shape, high spatial frequency components of boundary

*This gives you a summary of state,*
*e.g., state ⇔ weighted linear combination of features*
Linear Methods

Represent states as feature vectors:
for each $s \in S$:

$$\phi_s = (\phi_s(1), \phi_s(2), \ldots, \phi_s(n))^T$$

$$V_t(s) = \theta_t^T \phi_s = \sum_{i=1}^{\infty} \theta_t(i) \phi_s(i)$$

$$\nabla_{\theta} V_t(s) = \ ?$$
What are we learning? From what?

\[ \theta_t = \left( \theta_t(1), \theta_t(2), \ldots, \theta_t(n) \right)^T \]

Assume \( V_t \) is a (sufficiently smooth) differentiable function of \( \theta_t \), for all \( s \in S \).

Assume, for now, training examples of this form:

\[ \{ \text{description of } s_t, \ V^\pi(s_t) \} \]
Performance Measures

- Many are applicable but...
- a common and simple one is the mean-squared error (MSE) over a distribution $P$:

$$MSE(\theta_t) = \sum_{s \in S} P(s) [\pi(s) - V_t(s)]^2$$

- Why $P$?
- Why minimize MSE?
- Let us assume that $P$ is always the distribution of states at which backups are done.
- The on-policy distribution: the distribution created while following the policy being evaluated. Stronger results are available for this distribution.
RL with FA – Basic Setup

• For the policy evaluation (i.e., value prediction) problem, we are minimizing:

\[
MSE(\theta_t) = \sum_{s \in S} P(s) \left( \pi(s) - V_t(s) \right)^2
\]

Approximated ‘surface’

Value obtained, e.g., by backup updates

• This could be solved by gradient descent (updating the parameter vector \( \theta \) that specifies function \( V \)):

\[
\theta_{t+1} = \theta_t + \alpha [v_t - V_t(s_t)] \nabla_{\theta_t} V_t(s_t)
\]

Unbiased estimator, \( E\{v_t\} = V^\pi(s_t) \)
Gradient Descent

Let $f$ be any function of the parameter space. Its gradient at any point $\theta_t$ in this space is:

$$\nabla_{\theta} f(\theta_t) = \left( \frac{\partial f(\theta_t)}{\partial \theta(1)}, \frac{\partial f(\theta_t)}{\partial \theta(2)}, \ldots, \frac{\partial f(\theta_t)}{\partial \theta(n)} \right)^T.$$ 

Iteratively move down the gradient:

$$\theta_{t+1} = \theta_t - \alpha \nabla_{\theta} f(\theta_t)$$
Gradient Descent

For the MSE given earlier and using the chain rule:

\[
\theta_{t+1} = \theta_t - \frac{1}{2} \alpha \nabla_{\theta} \text{MSE}(\theta_t) \\
= \theta_t - \frac{1}{2} \alpha \nabla_{\theta} \sum_{s \in S} \pi(s) - V_t(s) \right]^2 \\
= \theta_t + \alpha \sum_{s \in S} \pi(s) - V_t(s) \nabla_{\theta} V_t(s)
\]
Gradient Descent

Use just the **sample gradient** instead:

\[
\theta_{t+1} = \theta_t - \frac{1}{2} \alpha \nabla_\theta \left[ \pi(s_t) - V_t(s_t) \right]^2 \\
= \theta_t + \alpha \left[ \pi(s_t) - V_t(s_t) \right] \nabla_\theta V_t(s_t),
\]

Since each sample gradient is an **unbiased estimate** of the true gradient, this converges to a local minimum of the MSE if \(\alpha\) decreases appropriately with \(t\).

\[
E \left[ \pi(s_t) - V_t(s_t) \right] \nabla_\theta V_t(s_t) = \sum_{s \in S} \pi(s) - V_t(s) \nabla_\theta V_t(s)
\]
But We Don’t have these Targets

Suppose we just have targets \( v_t \) instead:

\[
\theta_{t+1} = \theta_t + \alpha \left[ t - V_t(s_t) \right] \nabla_\theta V_t(s_t)
\]

If each \( v_t \) is an unbiased estimate of \( V^\pi(s_t) \), i.e., \( E \{ v_t \} = V^\pi(s_t) \), then gradient descent converges to a local minimum (provided \( \alpha \) decreases appropriately).

e.g., the Monte Carlo target \( v_t = R_t \):

\[
\theta_{t+1} = \theta_t + \alpha \left[ t - V_t(s_t) \right] \nabla_\theta V_t(s_t)
\]
RL with FA, TD style

Taking the ‘backward’ view of Temporal Difference learning,

$$\vec{\theta}_{t+1} = \vec{\theta}_t + \alpha \delta_t \vec{e}_t$$

where $\delta_t$ is the TD error,

$$\delta_t = r_{t+1} + \gamma V_t(s_{t+1}) - V_t(s_t)$$

and $\vec{e}_t$ is a column vector of eligibility traces, one for each component of $\theta_t$,

$$e_t = \gamma \lambda \vec{e}_{t-1} + \nabla_{\vec{\theta}_t} V_t(s_t)$$

In practice, how do we obtain this?
Nice Properties of Linear FA Methods

• The gradient is very simple: $\nabla_\theta V_t(s) = \phi_s$

• For MSE, the error surface is simple: quadratic surface with a single minimum.

• Linear gradient descent TD($\lambda$) converges:
  – Step size decreases appropriately
  – On-line sampling (states sampled from the on-policy distribution)
  – Converges to parameter vector $\theta_\infty$ with property:

$$MSE(\theta_\infty) \leq \frac{1 - \gamma \lambda}{1 - \gamma} MSE(\theta^*)$$

best parameter vector
On-Line Gradient-Descent TD($\lambda$)

Initialize $\vec{\theta}$ arbitrarily
Repeat (for each episode):
  $\vec{e} = 0$
  $s \leftarrow$ initial state of episode
Repeat (for each step of episode):
  $a \leftarrow$ action given by $\pi$ for $s$
  Take action $a$, observe reward, $r$, and next state, $s'$
  $\delta \leftarrow r + \gamma V(s') - V(s)$
  $\vec{e} \leftarrow \gamma \lambda \vec{e} + \nabla_{\vec{\theta}} V(s)$
  $\vec{\theta} \leftarrow \vec{\theta} + \alpha \delta \vec{e}$
  $s \leftarrow s'$
until $s$ is terminal
On Basis Functions: Coarse Coding
Shaping Generalization in Coarse Coding

a) Narrow generalization
b) Broad generalization
c) Asymmetric generalization
Learning and Coarse Coding

#Examples
10
40
160
640
2560
10240

Narrow features
Medium features
Broad features

desired function
approximation

feature width
Tile Coding

- Binary feature for each tile
- Number of features present at any one time is constant
- Binary features mean weighted sum easy to compute
- Easy to compute indices of the features present

Shape of tiles $\Rightarrow$ Generalization

$\#$Tilings $\Rightarrow$ Resolution of final approximation
Tile Coding, Contd.

Irregular tilings

Hashing
Radial Basis Functions (RBFs)

- e.g., Gaussians

\[
\phi_s(i) = \exp\left( -\frac{\|s - c_i\|^2}{2\sigma_i^2} \right)
\]
Beating the “Curse of Dimensionality”

• Can you keep the number of features from going up exponentially with the dimension?
• Function complexity, not dimensionality, is the problem.
• Kanerva coding:
  – Select a set of binary prototypes
  – Use Hamming distance as distance measure
  – Dimensionality is no longer a problem, only complexity
• “Lazy learning” schemes:
  – Remember all the data
  – To get new value, find nearest neighbors & interpolate
  – e.g., (nonparametric) locally-weighted regression
Going from Value Prediction to GPI

- So far, we’ve only discussed policy evaluation where the value function is represented as an approximated function.

- In order to extend this to a GPI-like setting,
  1. Firstly, we need to use the action-value functions.
  2. Combine that with the policy improvement and action selection steps.
  3. For exploration, we need to think about on-policy vs. off-policy methods.
Gradient Descent Update for Action-Value Function Prediction

\[ \theta_{t+1} = \theta_t + \alpha[v_t - Q_t(s_t, a_t)] \nabla_{\theta_t} Q_t(s_t, a_t) \]

e.g., \( R_t = r_{t+1} + \gamma Q_t(s_{t+1}, a_{t+1}) \)

One could implement this in the TD(\(\lambda\)) fashion as,

\[ \theta_{t+1} = \theta_t + \alpha \delta_t e_t \]

where, \( \delta_t = r_{t+1} + \gamma Q_t(s_{t+1}, a_{t+1}) - Q_t(s_t, a_t) \)

and \( e_t = \gamma \lambda e_{t-1} + \nabla_{\theta_t} Q_t(s_t, a_t) \)
How to Plug-in Policy Improvement or Action Selection?

• If spaces are very large, or continuous, this is an active research topic and there are no conclusive answers

• For small-ish discrete spaces,
  – For each action, $a$, available at a state, $s_t$, compute $Q_t(s_t, a)$ and find the greedy action according to it

$$a_t^* = \arg \max_a Q_t(s_t, a)$$

  – Then, one could use this as part of an $\varepsilon$-greedy action selection or as the estimation policy in off-policy methods
On Eligibility Traces with Function Approx.

• The formulation of control with function approx., so far, is mainly assuming accumulating traces
• Replacing traces have advantages over this
• However, they do not directly extend to case of function approximation
  – Notice that we now maintain a column vector of eligibility traces, one trace for each component of parameter vector
  – So, can’t update separately for a state (as in tabular methods)
• A practical way to get around this is to do the replacing traces procedure for features rather than states
  – Could also utilize an optional clearing procedure, over features
On-policy, SARSA(\(\lambda\)), control with function approximation

```
Initialize \(\vec{\theta}\) arbitrarily
Repeat (for each episode):
    \(\vec{e} = 0\)
    \(s, a \leftarrow\) initial state and action of episode
    \(\mathcal{F}_a \leftarrow\) set of features present in \(s, a\)
Repeat (for each step of episode):
    For all \(i \in \mathcal{F}_a\):
        \(e(i) \leftarrow e(i) + 1\) (accumulating traces)
        or \(e(i) \leftarrow 1\) (replacing traces)
    Take action \(a\), observe reward, \(r\), and next state, \(s\)
    \(\delta \leftarrow r - \sum_{i \in \mathcal{F}_a} \theta(i)\)
    With probability \(1 - \varepsilon\):
        For all \(a \in \mathcal{A}(s)\):
            \(\mathcal{F}_a \leftarrow\) set of features present in \(s, a\)
            \(Q_a \leftarrow \sum_{i \in \mathcal{F}_a} \theta(i)\)
            \(a \leftarrow \arg\max_a Q_a\)
        else
            \(a \leftarrow\) a random action \(\in \mathcal{A}(s)\)
            \(\mathcal{F}_a \leftarrow\) set of features present in \(s, a\)
            \(Q_a \leftarrow \sum_{i \in \mathcal{F}_a} \theta(i)\)
            \(\delta \leftarrow \delta + \gamma Q_a\)
            \(\vec{\theta} \leftarrow \vec{\theta} + \alpha \delta \vec{e}\)
            \(\vec{e} \leftarrow \gamma \lambda \vec{e}\)
    until \(s\) is terminal
```
Off-policy, Q-learning, control with function approximation

Let’s discuss it using the same grid world...

Initialize $\tilde{\theta}$ arbitrarily

Repeat (for each episode):

$\bar{e} = \tilde{\theta}$

$s, a \leftarrow$ initial state and action of episode

$\mathcal{F}_a \leftarrow$ set of features present in $s, a$

Repeat (for each step of episode):

For all $i \in \mathcal{F}_a$: $e(i) \leftarrow e(i) + 1$

Take action $a$, observe reward $r$, and next state $s$

$\delta \leftarrow r - \sum_{i \in \mathcal{F}_a} \theta(i)$

For all $a \in \mathcal{A}(s)$:

$\mathcal{F}_a \leftarrow$ set of features present in $s, a$

$Q_a \leftarrow \sum_{i \in \mathcal{F}_a} \theta(i)$

$\delta \leftarrow \delta + \gamma \max_a Q_a$

$\tilde{\theta} \leftarrow \tilde{\theta} + \alpha \delta \bar{e}$

With probability $1 - \varepsilon$:

For all $a \in \mathcal{A}(s)$:

$Q_a \leftarrow \sum_{i \in \mathcal{F}_a} \theta(i)$

$a \leftarrow \arg \max_a Q_a$

$\bar{e} \leftarrow \gamma \lambda \bar{e}$

else

$a \leftarrow$ a random action $\in \mathcal{A}(s)$

$\bar{e} \leftarrow 0$

until $s$ is terminal
Example: Mountain-car Task

- Drive an underpowered car up a steep mountain road
- Gravity is stronger than engine (like in cart-pole example)
- Example of a continuous control task where system must move away from goal first, then converge to goal
- Reward of -1 until car ‘escapes’
- Actions: +\( \tau \), -\( \tau \), 0

\[
x_{t+1} = \text{bound}[x_t + \dot{x}_{t+1}]
\]
\[
\dot{x}_{t+1} = \text{bound}[\ddot{x}_t + 0.001a_t - 0.0025 \cos(3x_t)]
\]
Mountain-car Example: Cost-to-go Function (SARSA(\(\lambda\)) solution)
Mountain Car Solution with RBFs

[Computed by M. Kretchmar]
Parameter Choices: Mountain Car Example

The effect of $\alpha$, $\lambda$, and the kind of traces on early performance on the mountain-car task. This study used five $9 \times 9$ tilings.
Off-Policy Bootstrapping

• All methods except MC make use of an existing value estimate in order to update the value function
• If the value function is being separately approximated, what is the combined effect of the two iterative processes?
• As we already saw, value prediction with linear gradient-descent function approximation converges to a sub-optimal point (as measured by MSE)
  – The further $\lambda$ strays from 1, the more the deviation from optimality
• Real issue is that $V$ is learned from an on-policy distribution
  – Off-policy bootstrapping with function approximation can lead to divergence (MSE tending to infinity)
Some Issues with Function Approximation: Baird’s Counterexample

Reward = 0, on all transitions
True $V^\pi(s) = 0$, for all $s$
Parameter updates, as below.
If we backup according to an uniform off-policy distribution (as opposed to the on-policy distribution), the estimates **diverge** for some parameters!

- Similar counterexamples exist for Q-learning as well…
Another Example

Baird’s counterexample has a simple fix: Instead of taking small steps towards expected one step returns, change value function to the best least-squares approximation.

$$\tilde{\theta}_{k+1} = \tilde{\theta}_k + \alpha \sum_s \left[ E \{ r_{t+1} + \gamma V_t(s_{t+1}) \mid s_t = s \} - V_k(s) \right] \nabla \tilde{\theta}_k V_k(s).$$

However, even this is not a general rule!

Reward = 0, on all transitions
True $V^\pi(s) = 0$, for all $s$

Tsitsiklis and Van Roy’s counterexample to DP policy evaluation with least-squares linear function approximation.
Tsitsiklis and Van Roy’s Counterexample

\[
\theta_{k+1} = \arg \min_{\theta \in \mathbb{R}} \sum_{s \in S} \left[ V_\theta(s) - E_\pi \{ r_{t+1} + \gamma V_\theta_k(s_{t+1}) | s_t = s \} \right]^2
\]

\[
= \arg \min_{\theta \in \mathbb{R}} \left[ \theta - \gamma 2 \theta_k \right]^2 + \left[ 2 \theta - (1 - \varepsilon) \gamma 2 \theta_k \right]^2
\]

\[
= \frac{6 - 4\varepsilon}{5} \gamma \theta_k,
\]

\(V_\theta\) denotes the value function given \(\theta\). The sequence \(\{\theta_k\}\) diverges when \(\gamma > \frac{5}{6 - 4\varepsilon}\) and \(\theta_0 \neq 0\).

Some function approximation methods (that do not extrapolate), such as nearest neighbors or locally weighted regression, can avoid this. - another solution is to change the objective (e.g., min 1-step expected error)
Some Case Studies

Positive examples (instead of just counterexamples)...

TD Gammon

Tesauro 1992, 1994, 1995, ...

• White has just rolled a 5 and a 2 so can move one of his pieces 5 and one (possibly the same) 2 steps
• Objective is to advance all pieces to points 19-24
• Hitting
• Doubling
• 30 pieces, 24 locations implies enormous number of configurations
• Effective branching factor of 400
A Few Details

• Reward: 0 at all times except those in which the game is won, when it is 1

• Episodic (game = episode), undiscounted

• Gradient descent TD(\(\lambda\)) with a multi-layer neural network
  – weights initialized to small random numbers
  – backpropagation of TD error
  – four input units for each point; unary encoding of number of white pieces, plus other features

• Use of afterstates, learning during self-play
Multi-layer Neural Network

predicted probability of winning, $V_t$

TD error, $V_{t+1} - V_t$

hidden units (40-80)

backgammon position (198 input units)
## Summary of TD-Gammon Results

<table>
<thead>
<tr>
<th>Program</th>
<th>Hidden Units</th>
<th>Training Games</th>
<th>Opponents</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>TD-Gam 0.0</td>
<td>40</td>
<td>300,000</td>
<td>other programs</td>
<td>tied for best</td>
</tr>
<tr>
<td>TD-Gam 1.0</td>
<td>80</td>
<td>300,000</td>
<td>Robertie, Magriel, ...</td>
<td>−13 pts / 51 games</td>
</tr>
<tr>
<td>TD-Gam 2.0</td>
<td>40</td>
<td>800,000</td>
<td>various Grandmasters</td>
<td>−7 pts / 38 games</td>
</tr>
<tr>
<td>TD-Gam 2.1</td>
<td>80</td>
<td>1,500,000</td>
<td>Robertie</td>
<td>−1 pt / 40 games</td>
</tr>
<tr>
<td>TD-Gam 3.0</td>
<td>80</td>
<td>1,500,000</td>
<td>Kazaros</td>
<td>+6 pts / 20 games</td>
</tr>
</tbody>
</table>
Samuel’s Checkers Player

Arthur Samuel 1959, 1967

• Score board configurations by a “scoring polynomial” (after Shannon, 1950)
• Minimax to determine “backed-up score” of a position
• Alpha-beta cutoffs
• Rote learning: save each board config encountered together with backed-up score
  – needed a “sense of direction”: like discounting
• Learning by generalization: similar to TD algorithm
Samuel’s Backups

actual events

hypothetical events

backup
The Basic Idea

“... we are attempting to make the score, calculated for the current board position, look like that calculated for the terminal board positions of the chain of moves which most probably occur during actual play.”

A. L. Samuel

Some Studies in Machine Learning Using the Game of Checkers, 1959