Introduction to Reinforcement Learning

Part 8: RL with Approximations
Finite Markov decision processes (MDPs) with feature vectors

states $S_t \in \mathcal{S}$
actions $A_t \in \mathcal{A}$
rewards $R_t \in \mathcal{R}$
feature vectors $\mathbf{x}_t \in \mathbb{R}^n$

$$\mathbf{x}_t = \mathbf{x}(S_t) \quad \mathbf{x} : \mathcal{S} \rightarrow \mathbb{R}^n$$

policy $\pi : \mathcal{A} \times \mathcal{S} \rightarrow [0, 1]$ $\pi(a|s) = \Pr\{A_t = a|S_t = s\}$

dynamics $p : \mathcal{S} \times \mathcal{R} \times \mathcal{S} \times \mathcal{A} \rightarrow [0, 1]$ $p(s', r|s, a) = \Pr\{S_{t+1} = s', R_{t+1} = r|S_t = s, A_t = a\}$
Value Prediction with FA

As usual: Policy Evaluation (the prediction problem):
for a given policy $\pi$, estimate the state-value function $v_{\pi}$

In earlier chapters, value functions were stored in lookup tables.

Here, the value function estimate at time $t$, $V_t$, depends on a weight vector $w_t$, and only the weight vector is updated.

e.g., $w_t$ could be the vector of connection weights of a neural network.
Linear Methods

Represent states as feature vectors:

for each \( s \in S \):

\[
\hat{v}(s, \mathbf{w}) = \mathbf{w}^\top \mathbf{x}(s) = \sum_{i=1}^{n} w_i x_i(s)
\]

\[
\nabla_{\mathbf{w}} \hat{v}(s, \mathbf{w}) = ?
\]
Nice Properties of Linear FA Methods

- The gradient is very simple: \( \nabla_w \hat{v}(s, w) = x(s) \)
- 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 weight vector \( w_\infty \) with property:

\[
RMSE(w_\infty) \leq \frac{1 - \gamma \lambda}{1 - \gamma} RMSE(w^*)
\]

(Tsitsiklis & Van Roy, 1997)
Learning and Coarse Coding

- Narrow features
- Medium features
- Broad features

#Examples:
- 10
- 40
- 160
- 640
- 2560
- 10240

desired function
approximation

feature width
Tile Coding

- Binary feature for each tile
- Number of features present at any one time is constant
- Binary features means 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 Cont.

Irregular tilings

- a) Irregular
- b) Log stripes
- c) Diagonal stripes

Hashing

CMAC
“Cerebellar model arithmetic computer”
Albus 1971
Coarse Coding

original representation \rightarrow \text{expanded representation, many features}

\theta_t \rightarrow \Sigma \rightarrow \text{approximation}
Shaping Generalization in Coarse Coding

a) Narrow generalization

b) Broad generalization

c) Asymmetric generalization
Gradient-based TD(\(\lambda\)), backwards view

\[ \delta_t = R_{t+1} + \gamma \hat{v}(S_{t+1}, w_t) - \hat{v}(S_t, w_t) \]

\[ e_t = \gamma \lambda e_{t-1} + \nabla w_t \hat{v}(S_t, w_t) \]

\[ w_{t+1} = w_t + \alpha \delta_t e_t \]
Control with FA

- Learning state-action values
  Training examples of the form:
  \[
  \{ \text{description of } (S_t, A_t), Q_t \} 
  \]
- The general gradient-descent rule:
  \[
  w_{t+1} = w_t + \alpha \left[ Q_t - \hat{q}(S_t, A_t, w_t) \right] \nabla w_t \hat{q}(S_t, A_t, w_t) 
  \]
- Gradient-descent Sarsa(\(\lambda\)) (backward view):
  \[
  w_{t+1} = w_t + \alpha \delta_t e_t 
  \]
  where:
  \[
  \delta_t = R_{t+1} + \gamma \hat{q}(S_{t+1}, A_{t+1}, w_t) - \hat{q}(S_t, A_t, w_t) 
  \]
  \[
  e_t = \gamma \lambda e_{t-1} + \nabla w_t \hat{q}(S_t, A_t, w_t) 
  \]
Approx Value Functions on Mountain-Car Task
Mountain-Car Results
Should We Bootstrap?

- **Mountain Car**
  - Steps per episode vs. $\lambda$
  - Accumulating traces vs. Replacing traces

- **Random Walk**
  - RMS error vs. $\lambda$
  - Accumulating traces vs. Replacing traces

- **Puddle World**
  - Cost per episode vs. $\lambda$
  - Replacing traces

- **Cart and Pole**
  - Failures per 100,000 steps vs. $\lambda$
  - Accumulating traces vs. Replacing traces
Summary

- Generalization
- Adapting supervised-learning function approximation methods
- Linear gradient-descent methods
  - Tile coding