Assignment #1

Recap RL so far

- When model is available:
  - VI, PI, GPI, LP

- When model is not available:
  - Model-based RL: collect data, estimate model, run one of the above methods for estimated model
  - Model-free RL: learn V, Q directly from experience:
    - TD(\(\lambda\)), sarsa(\(\lambda\)): on policy updates
    - Q: off policy updates

- What about large MDPs for which we cannot represent all states in memory or cannot collect experience from all states?
  → Function Approximation

Generalization and function approximation

- Represent the value function using a parameterized function \(V_\theta(s)\), e.g.:
  - Neural network: \(\theta\) is a vector with the weights on the connections in the network
  - Linear function approximator: \(V_\theta(s) = \theta^T \phi(s)\)
    - Radial basis functions: \(\phi_i(s) = \exp\left(\frac{1}{\alpha}(s - s_i)^T \Sigma^{-1}(s - s_i)\right)\)
    - Tilings: (often multiple) partitions of the state space
    - Polynomials: \(\phi_i(s) = x_1^{j_1} x_2^{j_2} \ldots x_n^{j_n}\)
    - Fourier basis: \(\{1, \sin(2\pi \frac{x_1}{L_1}), \cos(2\pi \frac{x_1}{L_1}), \sin(2\pi \frac{x_2}{L_2}), \cos(2\pi \frac{x_2}{L_2}), \ldots\}\)
      - [Note: most algorithms kernelizable]
  - Often also specifically designed for the problem at hand
**Example: tetris**

- **state**: board configuration + shape of the falling piece \( \sim 2^{200} \) states!
- **action**: rotation and translation applied to the falling piece

\[
V(s) = \sum_{i=1}^{22} \theta_i \phi_i(s)
\]

- **22 features aka basis functions** \( \phi_i \)
  - Ten basis functions, 0, \ldots, 9, *mapping the state to the height* \( h[k] \) *of each of the ten columns.*
  - Nine basis functions, 10, \ldots, 18, *each mapping the state to the absolute difference* between heights of successive columns: \( |h[k+1] - h[k]|, k = 1, \ldots, 9. \)
  - One basis function, 19, that maps state to the maximum column height: \( \max_k h[k] \)
  - One basis function, 20, that maps state to the number of ‘holes’ in the board.
  - One basis function, 21, that is equal to 1 in every state.

[Bertsekas & Ioffe, 1996 (TD); Bertsekas & Tsitsiklis1996 (TD); Kakade 2002 (policy gradient); Farias & Van Roy, 2006 (approximate LP)]

---

**Objective**

- A standard way to find \( \theta \) in supervised learning, optimize MSE:

\[
\min_\theta \sum_s P(s) \left( V(s) - V_\theta(s) \right)^2
\]

- Is this the correct objective?
When performing policy evaluation, we can obtain samples by simply executing the policy and recording the empirical (discounted) sum of rewards.

\[
\min_\theta \sum_{\text{encountered states } s} \left( \hat{V}_\pi(s) - V^\pi_\theta(s) \right)^2
\]

**TD methods:** use \( v_t = r_t + \gamma V^\pi_\theta(s_{t+1}) \) as a substitute for \( V^\pi(s) \)

---

**Stochastic gradient descent**

- Stochastic gradient descent to optimize MSE objective:
  - Iterate
    - Draw a state \( s \) according to \( P \)
    - Update:
      \[
      \theta \leftarrow \theta - \frac{1}{2\alpha} \nabla_{\theta} \left( V^\pi(s) - V^\pi_\theta(s) \right)^2 = \theta + \alpha (V^\pi(s) - V^\pi_\theta(s)) \nabla_{\theta} V^\pi_\theta(s)
      \]
  - TD(0): use \( v_t = r_t + \gamma V^\pi_\theta(s_{t+1}) \) as a substitute for \( V^\pi(s) \)
    \[
    \theta_{t+1} \leftarrow \theta_t + \alpha \left( R(s_t, a_t, s_{t+1}) + \gamma V^\pi_\theta(s_{t+1}) - V^\pi_\theta(s_t) \right) \nabla_{\theta} V^\pi_\theta(s_t)
    \]
TD(\(\lambda\)) with function approximation

- time t:
  \[ \theta_{t+1} \leftarrow \theta_t + \alpha \left( R(s_t, a_t, s_{t+1}) + \gamma \nabla_{\theta_t} V^\pi_{\theta_t}(s_t) \cdot \nabla_{\theta_t} V^\pi_{\theta_t}(s_t) \right) \]

- time t+1:
  \[ \theta_{t+2} \leftarrow \theta_{t+1} + \alpha \left( R(s_{t+1}, a_{t+1}, s_{t+2}) + \gamma \nabla_{\theta_{t+1}} V^\pi_{\theta_{t+1}}(s_{t+1}) \cdot \nabla_{\theta_{t+1}} V^\pi_{\theta_{t+1}}(s_{t+1}) \right) \]

\[ \theta_{t+2} \leftarrow \theta_{t+2} + \alpha \delta_{t+1} \lambda \epsilon_t \] [“improving previous update”]

Combined:
\[ \delta_{t+1} = R(s_t, a_t, s_{t+1}) + \gamma \nabla_{\theta_t} V^\pi_{\theta_t}(s_t) \cdot \nabla_{\theta_t} V^\pi_{\theta_t}(s_t) \]
\[ \epsilon_{t+1} = \gamma \lambda \epsilon_t + \nabla_{\theta_{t+1}} V^\pi_{\theta_{t+1}}(s_{t+1}) \]
\[ \theta_{t+2} = \theta_{t+1} + \alpha \delta_{t+1} \epsilon_{t+1} \]

---

Can similarly adapt sarsa(\(\lambda\)) and Q(\(\lambda\)) eligibility vectors for function approximation
Guarantees

- Monte Carlo based evaluation:
  - Provides unbiased estimates under current policy
  - Will converge to true value of current policy

- Temporal difference based evaluations:
  - TD(λ) w/linear function approximation: [Tsitsiklis and Van Roy, 1997]
    - “If” samples are generated from traces of execution of the policy π, and for appropriate choice of step-sizes α, TD(λ) converges and at convergence: [D = expected discounted state visitation frequencies under policy π]
    \[
    \|V_\theta - V^*\|_D \leq \frac{1 - \lambda \gamma}{1 - \gamma} \|V^* - V^*\|_D
    \]
  - Sarsa(λ) w/linear function approximation: same as TD
  - Q w/linear function approximation: [Melo and Ribeiro, 2007] Convergence to “reasonable” Q value under certain assumptions, including: \(\forall s, a \|\phi(s, a)\|_1 \leq 1\)
  - [Could also use infinity norm contraction function approximators to attain convergence --- see earlier lectures. However, this class of function approximators tends to be more restrictive.]

Off-policy counterexamples

- Baird’s counterexample for off policy updates:

[Diagram of Baird’s counterexample]
Off-policy counterexamples

- Tsitsiklis and Van Roy counterexample: complete back-up with “off-policy” linear regression [i.e., uniform least squares, rather than waits by state visitation rates]

\[
\begin{align*}
\text{Intuition behind TD(0) with linear function approximation guarantees} \\
\text{Stochastic approximation of the following operations:} \\
\quad \text{Back-up: } (T^n V)(s) = \sum_s T(s, \pi(s), s') [R(s, \pi(s), s') + \gamma V(s')] \\
\quad \text{Weighted linear regression: } \min_\theta \sum_s D(s)((T^n V)(s) - \theta^T \phi(s))^2 \\
\quad \text{with solution: } \Phi \theta = \Phi (\Phi^T D \Phi)^{-1} \Phi^T D (T^n V) \\
\text{Key observations:} \\
\forall V_1, V_2 : \|T^n V_1 - T^n V_2\|_D \leq \gamma \|V_1 - V_2\|_D, \text{ here: } \|x\|_D = \sqrt{\sum_i D(i)x(i)^2} \\
\forall V_1, V_2 : \|\Pi_D V_1 - \Pi_D V_2\|_D \leq \|V_1 - V_2\|_D
\end{align*}
\]
Intuition behind TD(\(\lambda\)) guarantees

- Bellman operator:
  \[
  (T^\pi J)(s) = \sum_{s'} P(s'|s, \pi(s)) [g(s) + \gamma J(s')] = E[g(s) + \gamma J(s')]
  \]

- \(T^\lambda\) operator:
  \[
  T^\lambda J(s) = (1 - \lambda) \sum_{m=0}^{\infty} \lambda^m E \left[ \sum_{k=0}^{m} \gamma^k g(s_k) + \gamma^{m+1} J(s_{m+1}) \right]
  \]

- \(T^\lambda\) operator is contraction w.r.t. \(\|\cdot\|_D\) for all \(\lambda \in [0,1]\)

Should we use TD than well Monte Carlo?

- At convergence:
  \[
  \|V_\theta - V^*\|_D \leq \frac{1 - \lambda \gamma}{1 - \gamma} \|V^* - V^*_D\|_D
  \]
Empirical comparison
(See, Sutton&Barto p.221 for details)

Backgammon

- 15 pieces, try go reach “other side”
- Move according to roll of dice
- If hitting an opponent piece: it gets reset to the middle row
- Cannot hit points with two or more opponent pieces
Backgammon

- 30 pieces, 24+2 possible locations
- For typical state and dice roll: often 20 moves

TD Gammon  [Tesauro 92,94,95]

- Reward = 1 for winning the game
  = 0 other states
- Function approximator: 2 layer neural network
**Input features**

- For each point on the backgammon board, 4 input units indicate the number of white pieces as follows:
  - 1 piece \( \rightarrow \text{unit1}=1; \)
  - 2 pieces \( \rightarrow \text{unit1}=1, \text{unit2}=1; \)
  - 3 pieces \( \rightarrow \text{unit1}=1, \text{unit2}=1, \text{unit3}=1; \)
  - \( n>3 \) pieces \( \rightarrow \text{unit1}=1, \text{unit2}=1, \text{unit3}=1, \text{unit4} = (n-3)/2 \)
- Similarly for black

[This already makes for \( 2^4 \cdot 24 = 192 \) input units.]

- Last six: number of pieces on the bar (w/b), number of pieces that completed the game (w/b), white’s move, black’s move

---

**Neural net**

- Each hidden unit computes:
  \[ h(j) = \sigma(\sum_i w_{ij} \phi(i)) = \frac{1}{1+\exp\left(-\sum_i w_{ij} \phi(i)\right)} \]

- Output unit computes:
  \[ o = \sigma(\sum_j w_j h(j)) = \frac{1}{1+\exp\left(-\sum_j w_j h(j)\right)} \]

- Overall: \( o = f(\phi(1), \ldots, \phi(198); w) \)
Neural nets

- Popular at that time for function approximation / learning in general
- Derivatives/Gradients are easily derived analytically
  - Turns out they can be computed through backward error propagation --- name “error backpropagation”
- Susceptible to local optima!

Learning

- Initialize weights randomly
- TD(\(\lambda\)) \[\lambda = 0.7, \alpha = 0.1\]
- Source of games: self-play, greedy w.r.t. current value function [results suggest game has enough stochasticity built in for exploration purposes]
Results

- After 300,000 games as good as best previous computer programs
  - Neurogammon: highly tuned neural network trained on large corpus of exemplary moves
- TD Gammon 1.0: add Neurogammon features
  - Substantially better than all previous computer players; human expert level
- TD Gammon 2.0, 2.1: selective 2-ply search
- TD Gammon 3.0: selective 3-ply search, 160 hidden units