CS 486/686: Introduction to Artificial Intelligence

Reinforcement Learning
Large State Spaces

• Computer Go: $3^{361}$ states

• Inverted pendulum:
  • 4 dimensional, continuous state space

• Atari: 210 x 160 x 3 dimensions (pixel values)
Value-Function Approximation

• So far we have represented value functions by a look-up table (tabular RL)
  • Every state $s$ has an entry $V(s)$
  • Every state-action pair $s,a$ has an entry $Q(s,a)$

• Issue
  • There are too many states or actions to store in memory
  • It is too slow to learn the value of each state individually
Value-Function Approximation

• Estimate value functions with function approximation

\[ V^{\pi}(s) \sim \hat{V}(s, \mathbf{w}) \]

\[ Q^{\pi}(s, a) \sim \hat{Q}(s, a, \mathbf{w}) \]

• Let \( s=(x_1, x_2, ..., x_n)^T \) or \( (s,a) = (x_1(s,a), ..., x_n(s,a))^T \)

  • Linear: \( V(s, \mathbf{w}) = \sum_i w_i x_i(s) \), \( Q(s,a,\mathbf{w}) = \sum_i w_i x_i(s, a) \)
  • Non-linear (e.g. neural networks): \( V(s,\mathbf{w}) \) (\( Q(s,a,\mathbf{w}) \)) = \( g(x;\mathbf{w}) \)
Recall: Neural Networks

• Network of units linked by weighted edges

• Each unit computes: \( z = h(w^T x + b) \)
  - Inputs: \( x \)
  - Outputs: \( z \)
  - Weights: \( w \)
  - Bias: \( b \)
  - Activation function: \( h \)

• Neural networks with at least one “large enough” hidden layer consisting of sigmoid/tanh/Gaussian units can approximate any function arbitrarily closely
Gradient Q-Learning

• Minimize the error between Q-value estimate and a target
  • Estimate: $Q(s,a,w)$
  • Target: $r+\gamma \max_{a'} Q(s',a',w')$

• Squared Error:
  $$\text{err}(w) = \frac{1}{2} [Q(s, a, w) - r - \gamma \max_{a'} Q(s', a', w')]^2$$

• Gradient:
  $$\frac{\partial \text{Err}}{\partial w} = [Q(s, a, w) - r - \gamma \max_{a'} Q(s', a', w')] \frac{\partial Q(s, a, w)}{\partial w}$$
Gradient Q-Learning

Initialize weights $w$ at random in $[-1,1]$
Observe current state $s$

Loop
   - Select action $a$ and execute it
   - Receive immediate reward $r$
   - Observe new state $s'$
   - Gradient: $\frac{\partial \text{Err}}{\partial w} = [Q_w(s, a) - r - \gamma \max_{a'} Q_w(s', a')] \frac{\partial Q_w(s, a)}{\partial w}$
   - Update weights: $w \leftarrow w - \alpha \frac{\partial \text{Err}}{\partial w}$
   - Update state: $s \leftarrow s'$
Convergence?

• Tabular Q-learning converges when

\[ \sum_{t=0}^{\infty} \alpha_t = \infty \quad \text{and} \quad \sum_{t=0}^{\infty} \alpha_t^2 < \infty \]

• We typically set \( \alpha_t(s,a) = 1/n(s,a) \) where \( n(s,a) \) is the number of times \( (s,a) \) is visited or \( 1/t \).

• Linear function approximation
  • Same convergence guarantees

• Non-linear function approximation
  • No convergence guarantee
Non-linear function approximation

Handling divergence:
• Experience replay
• Use two networks:
  • Q-network
  • Target network

Experience Replay
• Store previous experiences \((s,a,,s',r)\) into a buffer and sample a mini-batch of previous experiences at each step to learn by Q-learning
  • Break correlations between successive updates (more stable learning)
  • Less interactions with environment needed (better data efficiency)
Target Network

• Use a target network that is updated only occasionally

• Repeat for each \((s,a,s',r)\) in a mini-batch:

\[
\begin{align*}
\mathbf{w} &\leftarrow \mathbf{w} - \alpha_t [Q(s, a, \mathbf{w}) - r - \gamma \max_{a'} Q(s', a', \mathbf{w}')] \frac{\partial Q(s, a, \mathbf{w})}{\partial r} \\
\mathbf{w}' &\leftarrow \mathbf{w}
\end{align*}
\]

• Observe: similar in spirit to value iteration
Deep Q-network (DQN)

• Gradient Q-learning with
  • Deep neural networks
  • Experience replay
  • Target network
DQN

Initialize weights $\mathbf{w}$ and $\mathbf{w}^{\prime}$ at random in $[-1,1]$
Observe current state $s$
Loop
  Select action $a$ and execute it
  Receive immediate reward $r$
  Observe new state $s'$
  Add $\langle s, a, s', r \rangle$ to experience buffer
  Sample mini-batch of experiences from buffer
  For each experience $\langle \hat{s}, \hat{a}, \hat{s}', \hat{r} \rangle$ in mini-batch
    Gradient: $\frac{\partial \text{Err}}{\partial \mathbf{w}} = [Q_w(\hat{s}, \hat{a}) - \hat{r} - \gamma \max_{\hat{a}'} Q_w(\hat{s}', \hat{a}')] \frac{\partial Q_w(\hat{s}, \hat{a})}{\partial \mathbf{w}}$
    Update weights: $\mathbf{w} \leftarrow \mathbf{w} - \alpha \frac{\partial \text{Err}}{\partial \mathbf{w}}$
  Update state: $s \leftarrow s'$
  Every $c$ steps, update target: $\mathbf{w}^{\prime} \leftarrow \mathbf{w}$
DQN for Atari
DQN versus Linear Approximation