Lecture 18: Deep Reinforcement Learning CS486/686 Intro to Artificial Intelligence

2023-7-11

Sriram Ganapathi Subramanian,
Vector Institute

UNIVERSITY OF WATERLOO

Outline

- RL with function approximation
  - Linear approximation
  - Neural network approximation

- Algorithms:
  - Gradient Q-learning
  - Deep Q-Network (DQN)

Quick Recap

- Markov decision processes: value iteration

$$V(s) \leftarrow \max_a R(s,a) + \gamma \sum_{s'} Pr(s' \mid s,a) V(s')$$

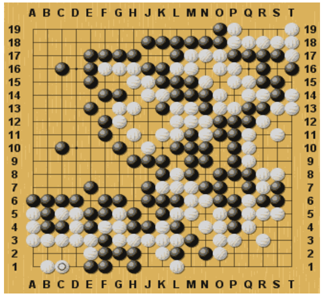
- Reinforcement learning: Q-learning

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[r + \gamma \max_{a'} Q(s',a') - Q(s,a) \right]$$

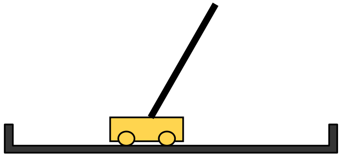
- Complexity depends on number of states and actions

Large State Spaces

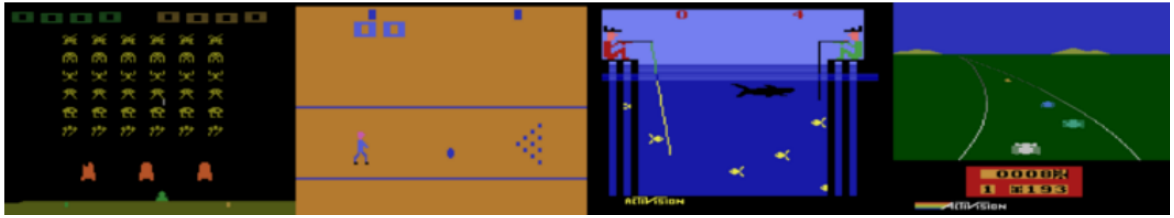
- Computer Go: 3^{361} states



- Inverted pendulum: $\langle x, x', \theta, \theta' \rangle$
  - 4-dimensional
    continuous state space



- Atari: 210 x 160 x 3 dimensions (pixel values)



UNIVERSITY OF WATERLOO

Function to be Approximated

- Policy: $\pi(s) \rightarrow a$

- Q-function: $Q(s, a) \in \Re$

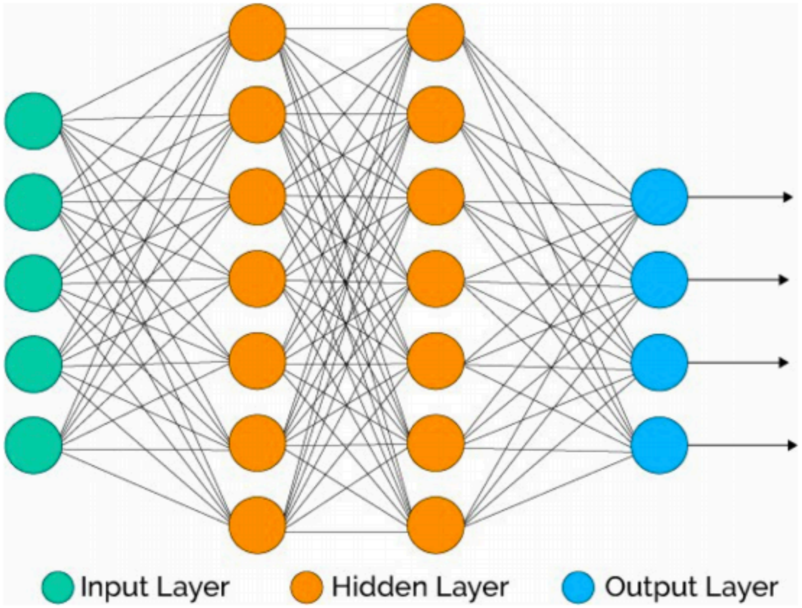
- Value function: $V(s) \in \Re$

Q-function Approximation

- Let $s = (x_1, x_2, \ldots, x_n)^T$
- Linear: $Q(s, a) \approx \sum_i w_{ai} x_i$
- Non-linear (e.g., neural network): $Q(s, a) \approx g(\boldsymbol{x}; \boldsymbol{w})$

Recall: Traditional Neural Network

- Network of units (computational neurons) linked by weighted edges
- Each unit computes: $z = h(\boldsymbol{w}^T\boldsymbol{x} + b)$
  - Inputs: $\boldsymbol{x}$
  - Outputs: z
  - Weights (parameters): $\boldsymbol{w}$
  - Bias: b
  - Activation function (usually non-linear): h

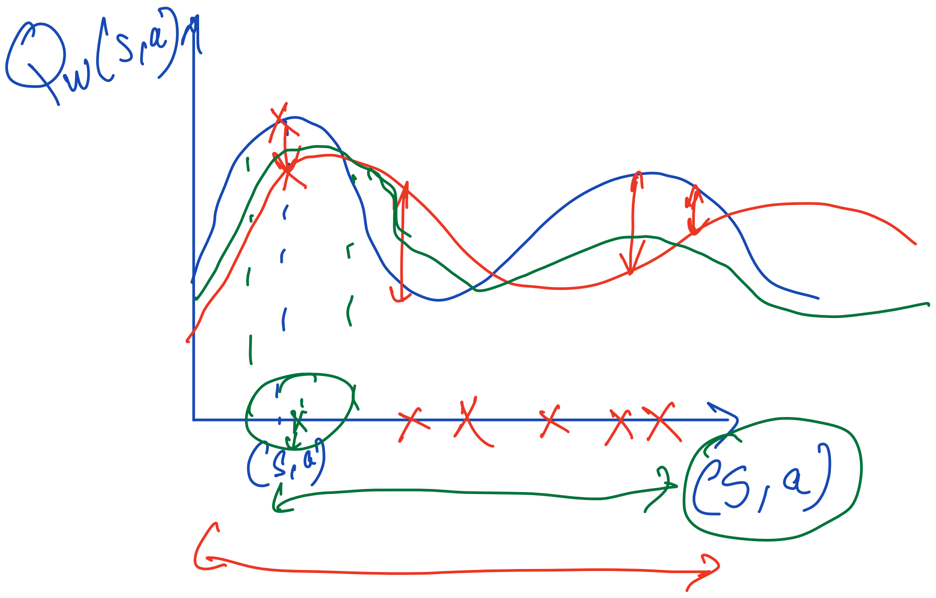


Recall: Universal Function Approximator

Theorem: Neural networks with at least one hidden layer of sufficiently many sigmoid/tanh/Gaussian units can approximate any function arbitrarily closely.

Gradient Q-learning

- Minimize squared error between Q-value estimate and target
  - Q-value estimate: $Q_{\boldsymbol{w}}(s, a)$
  - Target: $r + \gamma \max_{a'} Q_{\overline{\boldsymbol{w}}}(s', a')$
- Squared error: $Err(\boldsymbol{w}) = \frac{1}{2}[Q_{\boldsymbol{w}}(s, a) - r - \gamma \max_{a'} Q_{\overline{\boldsymbol{w}}}(s', a')]^2$
- Gradient: $\frac{\partial Err}{\partial \boldsymbol{w}} = [Q_{\boldsymbol{w}}(s, a) - r - \gamma \max_{a'} Q_{\overline{\boldsymbol{w}}}(s', a')] \frac{\partial Q_{\boldsymbol{w}}}{\partial \boldsymbol{w}}$



Qw(s,a)
(s,a)
(s,a)

Gradient Q-learning

Initialize weights $\boldsymbol{w}$ 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: $\frac{\partial Err}{\partial \boldsymbol{w}} = [Q_{\boldsymbol{w}}(s,a) - r - \gamma \max_{a'} Q_{\boldsymbol{w}}(s',a')] \frac{\partial Q_{\boldsymbol{w}}(s,a)}{\partial \boldsymbol{w}}$

Update weights: $\boldsymbol{w} \leftarrow \boldsymbol{w} - \alpha \frac{\partial Err}{\partial \boldsymbol{w}}$

Update state: $s \leftarrow s'$

Recap: Convergence of Tabular Q-learning

- Tabular Q-Learning converges to optimal Q-function under the following conditions:

$$\sum_{t=0}^{\infty} \alpha_t = \infty \text{ and } \sum_{t=0}^{\infty} \alpha_t^2 < \infty$$

- Let $\alpha(s, a) = \dfrac{1}{n(s, a)}$
  - Where $n(s, a)$ is # of times that (s, a) is visited

- Q-learning: $Q(s, a) \leftarrow Q(s, a) + \alpha(s, a)[r + \gamma \max_{a'} Q(s', a') - Q(s, a)]$

UNIVERSITY OF WATERLOO

Convergence of Linear Function Approximation Q-Learning

- Linear Q-Learning converges under the same conditions:

$$\sum_{t=0}^{\infty} \alpha_t = \infty \text{ and } \sum_{t=0}^{\infty} \alpha_t^2 < \infty$$

- Let $\alpha_t = \frac{1}{t}$
- Let $Q_{\boldsymbol{w}}(s, a) = \sum_i w_i x_i$
- Q-learning: $\boldsymbol{w} \leftarrow \boldsymbol{w} - \alpha_t \left[Q_{\boldsymbol{w}}(s, a) - r - \gamma \max_{a'} Q_{\boldsymbol{w}}(s', a') \right] \frac{\partial Q_{\boldsymbol{w}}(s, a)}{\partial \boldsymbol{w}}$

Divergence of Non-linear Gradient Q-learning

- Even when the following conditions hold

$$\sum_{t=0}^{\infty} \alpha_t = \infty \text{ and } \sum_{t=0}^{\infty} \alpha_t^2 < \infty$$

non-linear Q-learning may diverge

- Intuition:
  - Adjusting w to increase Q at (s, a) might introduce errors at nearby state-action pairs.

Mitigating divergence

- Two tricks are often used in practice:
  1. Experience replay
  2. Use two networks:
     - Q-network
     - Target network

Experience Replay

- Idea: store previous experiences $\langle s, a, s', r \rangle$ into a buffer and sample a mini-batch of previous experiences at each step to learn by Q-learning

- Advantages
  - Break correlations between successive updates (more stable learning)
  - Less interactions with environment needed (better data efficiency)

Target Network

- Idea: Use a separate target network that is updated only periodically

  repeat for each (s, a, s', r) in mini-batch:

$$\boldsymbol{w} \leftarrow \boldsymbol{w} - \alpha_t \left[Q_{\boldsymbol{w}}(s,a) - r - \gamma \max_{a'} Q_{\overline{\boldsymbol{w}}}(s',a') \right] \frac{\partial Q_{\boldsymbol{w}}(s,a)}{\partial \boldsymbol{w}}$$

$$\bar{\boldsymbol{w}} \leftarrow \boldsymbol{w}$$

- Advantage: mitigate divergence

UNIVERSITY OF WATERLOO

Target Network

- Similar to value iteration:

  repeat for all s

  $$V(s) \leftarrow \max_a R(s) + \gamma \sum_{s'} Pr(s'|s,a)\overline{V}(s') \qquad \forall s$$

  $$\overline{V} \leftarrow V$$

  repeat for each $\langle s, a, s', r \rangle$ in mini-batch:

  $$\boldsymbol{w} \leftarrow \boldsymbol{w} - \alpha_t \left[Q_{\boldsymbol{w}}(s,a) - r - \gamma \max_{a'} Q_{\overline{\boldsymbol{w}}}(s',a') \right] \frac{\partial Q_{\boldsymbol{w}}(s,a)}{\partial \boldsymbol{w}}$$

  $$\bar{\boldsymbol{w}} \leftarrow \boldsymbol{w}$$

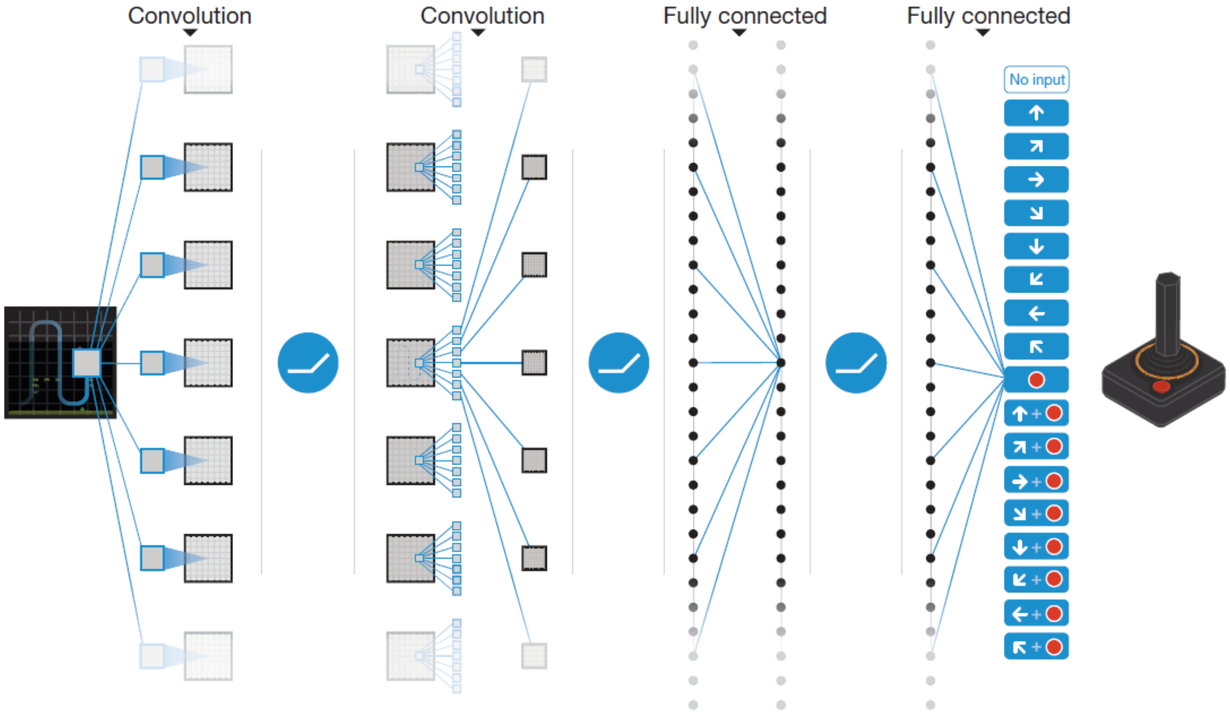
Deep Q-network (DQN)

- Deep Mind
- Deep Q-network: Gradient Q-learning with
  - Deep neural networks
  - Experience replay
  - Target network
- Breakthrough: human-level play in many Atari video games

Deep Q-network (DQN)

Initialize weights $\boldsymbol{w}$ and $\overline{\boldsymbol{w}}$ at random in $[-1,1]$
Observe current state s
Loop
    Select action a and execute it
    Receive immediate reward r
    Observe new state s'
    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: $\dfrac{\partial Err}{\partial \boldsymbol{w}} = [Q_{\boldsymbol{w}}(\hat{s}, \hat{a}) - \hat{r} - \gamma \max_{\hat{a}'} Q_{\overline{\boldsymbol{w}}}(\hat{s}', \hat{a}')] \dfrac{\partial Q_{\boldsymbol{w}}(\hat{s}, \hat{a})}{\partial \boldsymbol{w}}$
        Update weights: $\boldsymbol{w} \leftarrow \boldsymbol{w} - \alpha \dfrac{\partial Err}{\partial \boldsymbol{w}}$
    Update state: $s \leftarrow s'$
    Every c steps, update target: $\overline{\boldsymbol{w}} \leftarrow \boldsymbol{w}$

Deep Q-Network for Atari



UNIVERSITY OF WATERLOO

DQN versus Linear Approximation