Reinforcement Learning Lecture 4b:

Deep Q-networks
[SutBar] Sec. 9.4, 9.7,
[Sze] Sec. 4.3.2

Outline

- Value Function Approximation
  - Linear approximation
  - Neural network approximation
    - Deep Q-network

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)

  $Q(s, a) \approx g(\boldsymbol{x}; \boldsymbol{w})$

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:

$\overline{w}$ fixed

$$Err(w) = \frac{1}{2}[Q_w(s, a) - r - \gamma \max_{a'} Q_{\overline{\boldsymbol{w}}}(s', a')]^2$$

- Gradient

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

Gradient Q-learning

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

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:

$$\sum_{\mathrm{n}=0}^{\infty} \alpha_n = \infty \text{ and } \quad \sum_{\mathrm{n}=0}^{\infty} \alpha_n^2 < \infty$$

- Let $\alpha_n(s,a) = 1/n(s,a)$
  - Where $n(s,a)$ is # of times that (s,a) is visited

- Q-learning

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

Convergence of Linear Gradient Q-Learning

- Linear Q-Learning converges under the same conditions:

$$\sum_{\mathrm{n}=0}^{\infty} \alpha_n = \infty \text{ and } \quad \sum_{\mathrm{n}=0}^{\infty} \alpha_n^2 < \infty$$

- Let $\alpha_n = 1/n$
- Let $Q_w(s,a) = \sum_i w_i x_i$

- Q-learning

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

Divergence of Non-linear Gradient Q-learning

- Even when the following conditions hold
  $\sum_{\mathrm{n}=0}^{\infty} \alpha_n = \infty$ and $\sum_{\mathrm{n}=0}^{\infty} \alpha_n^2 < \infty$
  non-linear Q-learning may diverge

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

Mitigating divergence

- Two simple tricks are often used in practice:

1. Experience replay
2. Use two networks:
   - Q-network
   - Target network

Experience Replay

- Idea: 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

- Advantages
  - Break correlations between successive updates (more stable learning)
  - Fewer interactions with environment needed to converge (greater 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:

$$\boldsymbol{w} \leftarrow \boldsymbol{w} - \alpha_t [\underbrace{Q_{\boldsymbol{w}}(s, a)}_{\text{update}} - r - \gamma \max_{a'} \underbrace{Q_{\overline{\boldsymbol{w}}}(s', a')}_{\text{target}}] \frac{\partial Q_{\boldsymbol{w}}(s, a)}{\partial \boldsymbol{w}}$$

$\overline{\boldsymbol{w}} \leftarrow \boldsymbol{w}$

- Advantage: mitigate divergence

Target Network

- Similar to value iteration:

repeat for all s

$$\underbrace{V(s)}_{\text{update}} \leftarrow \max_a R(s) + \gamma \sum_{s'} \Pr(s'|s,a)\underbrace{\bar{V}(s')}_{\text{target}} \quad \forall s$$

$\bar{V} \leftarrow V$

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

$$\boldsymbol{w} \leftarrow \boldsymbol{w} - \alpha_n[\underbrace{Q_{\boldsymbol{w}}(s,a)}_{\text{update}} - r - \gamma \max_{a'} \underbrace{Q_{\overline{\boldsymbol{w}}}(s',a')}_{\text{target}}] \frac{\partial Q_{\boldsymbol{w}}(s,a)}{\partial \boldsymbol{w}}$$

$\overline{\boldsymbol{w}} \leftarrow \boldsymbol{w}$

Deep Q-network

- Google 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

Initialize weights w and $\bar{w}$ random in $[-1,1]$

Observe current state s

Loop

Select action a and execute it

Receive immediate reward r

Observe new state s'

Add (s, a, s', r) to experience buffer

Sample mini-batch of experiences from buffer

For each experience $(\hat{s}, \hat{a}, \hat{s}', \hat{r})$ in mini-batch

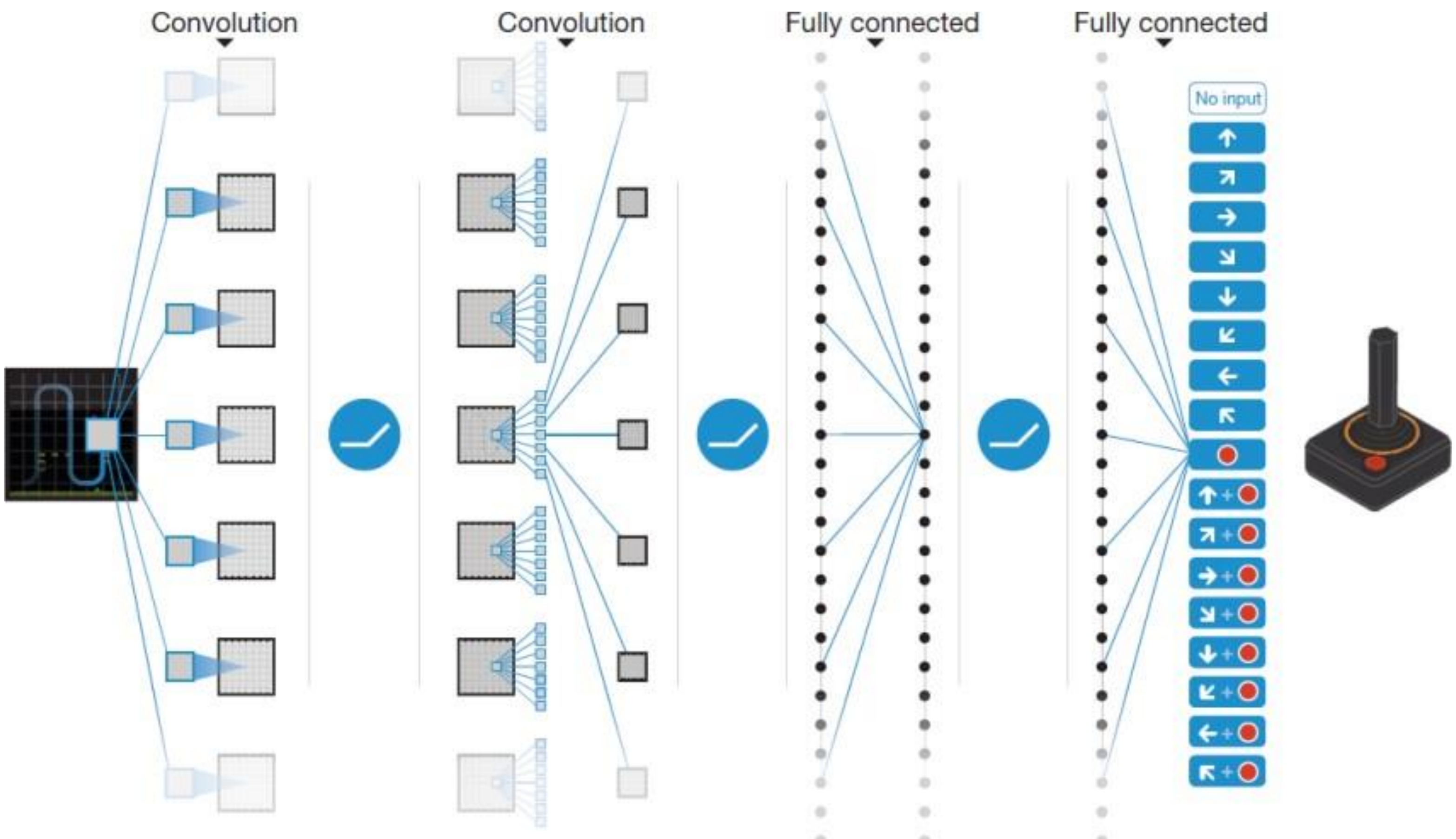
Gradient: $\frac{\partial Err}{\partial \boldsymbol{w}} = [Q_{\boldsymbol{w}}(\hat{s}, \hat{a}) - \hat{r} - \gamma \max_{a'} Q_{\overline{\boldsymbol{w}}}(\hat{s}', \hat{a}')] \frac{\partial Q_{\boldsymbol{w}}(\hat{s},\hat{a})}{\partial \boldsymbol{w}}$

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

Update state: $s \leftarrow s$'

Every c steps, update target: $\overline{\boldsymbol{w}} \leftarrow \boldsymbol{w}$

Deep Q-Network for Atari



DQN versus Linear approx.