

Lecture 3b: Deep Q-networks

CS885 Reinforcement Learning

2022-09-16

Complementary readings: [GBC] Chap. 6, 7, 8, [SutBar] Sec. 9.4, 9.7, [Sze] Sec. 4.3.2

Pascal Poupart
David R. Cheriton School of Computer Science



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

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

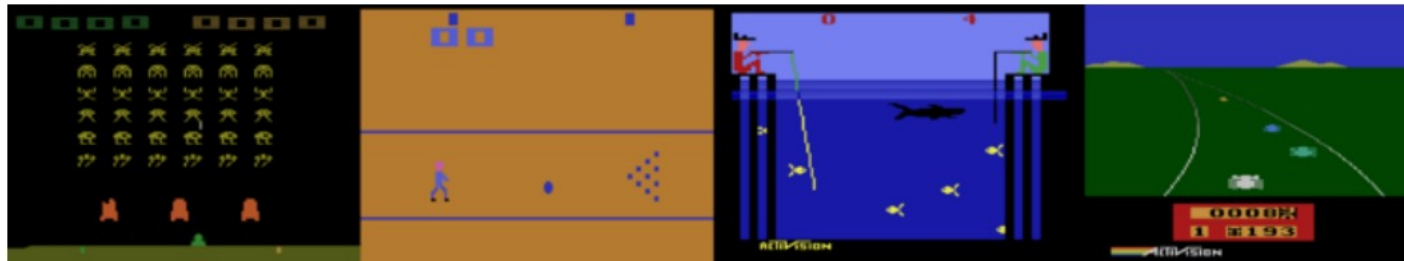
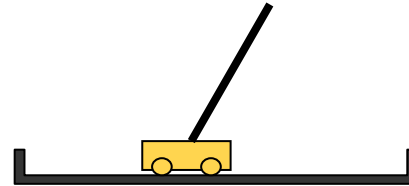
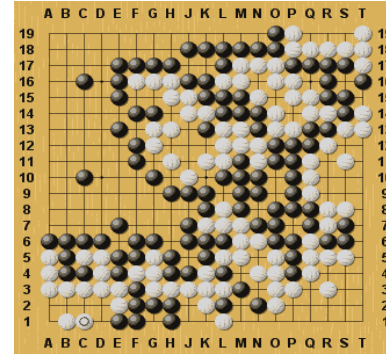
- Reinforcement learning: Q-learning

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

- Complexity depends on number of states and actions

Large State Spaces

- Computer Go: 3^{361} states
- Inverted pendulum: (x, x', θ, θ')
 - 4-dimensional continuous state space
- Atari: 210 x 160 x 3 dimensions (pixel values)

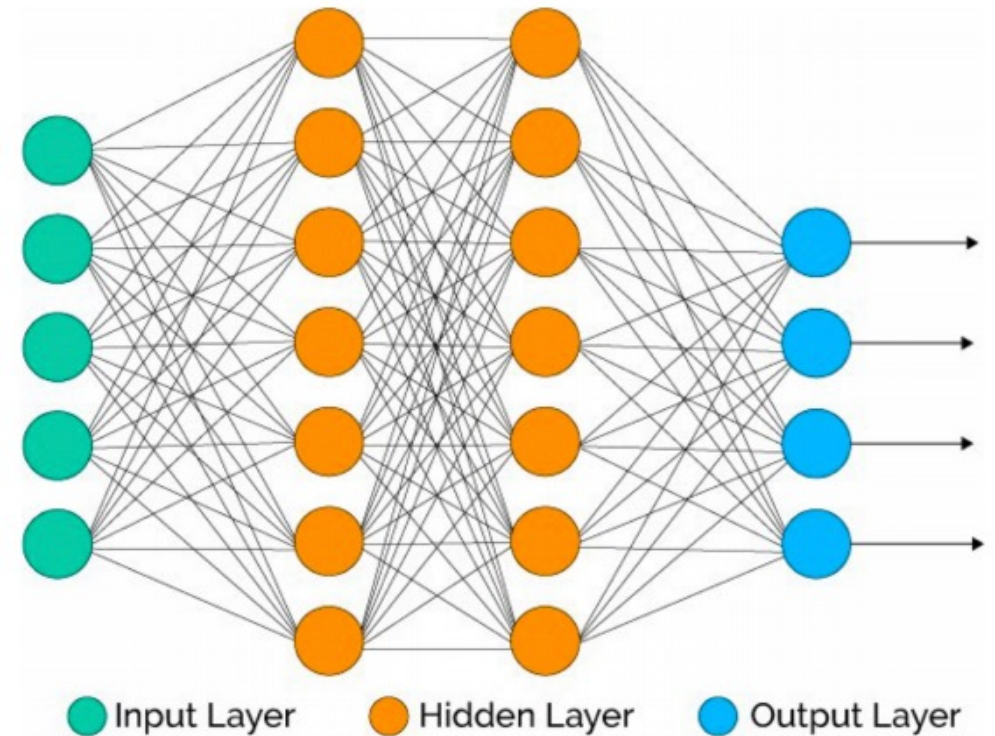


Functions to be Approximated

- Policy: $\pi(s) \rightarrow a$
- Q-function: $Q(s, a) \in \mathfrak{R}$
- Value function: $V(s) \in \mathfrak{R}$

Traditional Neural Network

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



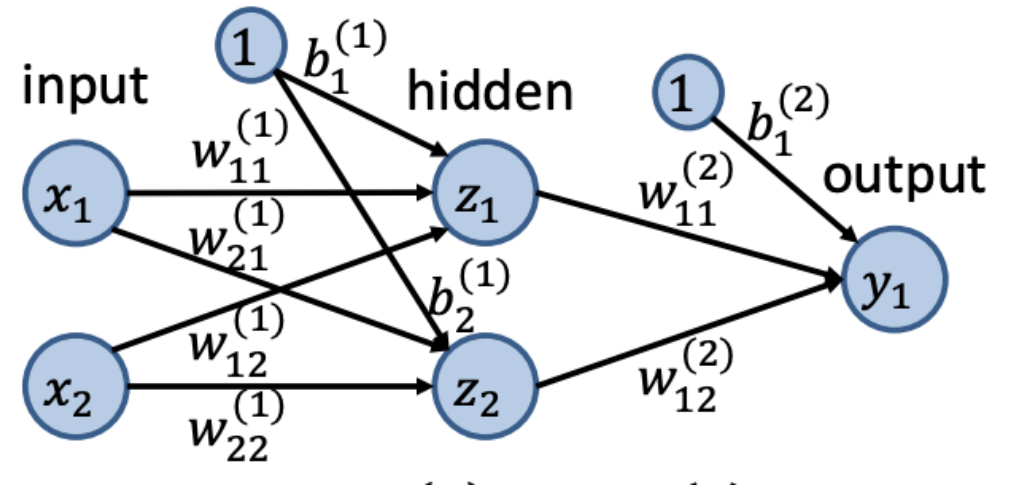
One Hidden Layer Architecture

- Feed-forward neural network

- Hidden units: $z_j = h_1(\mathbf{w}_j^{(1)} \mathbf{x} + b_j^{(1)})$

- Output units: $y_k = h_2(\mathbf{w}_k^{(2)} \mathbf{z} + b_k^{(2)})$

- Overall: $y_k = h_2 \left(\sum_j w_{kj}^{(2)} h_1 \left(\sum_i w_{ji}^{(1)} x_i + b_j^{(1)} \right) + b_k^{(2)} \right)$



Common Activation Functions

- Sigmoid: $h(a) = \sigma(a) = \frac{1}{1+e^{-a}}$
- Softmax: $h(\mathbf{a})_i = \frac{e^{a_i}}{\sum_j e^{a_j}}$
- Tanh (hyperbolic tangent): $h(a) = \frac{e^a - e^{-a}}{e^a + e^{-a}}$
- Gaussian: $h(a) = e^{-0.5\left(\frac{a-\mu}{\sigma}\right)^2}$
- ReLU (Rectified Linear Unit): $h(a) = \max(a, 0)$
- Identity: $h(a) = a$

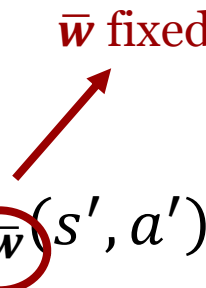
Universal Function Approximator

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

Q-function Approximation

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

Gradient Q-learning

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

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

Gradient Q-learning

Initialize weights \mathbf{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'

$$\text{Gradient: } \frac{\partial \text{Err}}{\partial \mathbf{w}} = \left[Q_{\mathbf{w}}(s, a) - r - \gamma \max_{a'} Q_{\mathbf{w}}(s', a') \right] \frac{\partial Q_{\mathbf{w}}(s, a)}{\partial \mathbf{w}}$$

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

$$\text{Update state: } s \leftarrow s'$$

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 $\alpha(s, a) = 1/n(s, a)$
 - Where $n(s, a)$ is # of times that (s, a) is visited
- Q-learning: $Q(s, a) \leftarrow Q(s, a) + \alpha(s, a)[r + \gamma \max_{a'} Q(s', a') - Q(s, a)]$

Convergence of Linear Gradient Q-Learning

- Linear Q-Learning converges under the same conditions:

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

- Let $\alpha_t = 1/t$

- Let $Q_w(s, a) = \sum_i w_i x_i$

- Q-learning: $\mathbf{w} \leftarrow \mathbf{w} - \alpha_t \left[Q_w(s, a) - r - \gamma \max_{a'} Q_w(s', a') \right] \frac{\partial Q_w(s, a)}{\partial \mathbf{w}}$

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 (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**)
 - Less interactions with environment needed to converge (**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:

$$\begin{aligned} \mathbf{w} &\leftarrow \mathbf{w} - \alpha_t \left[\underbrace{Q_{\mathbf{w}}(s, a)}_{\text{update}} - r - \gamma \max_{a'} \underbrace{Q_{\bar{\mathbf{w}}}(s', a')}_{\text{target}} \right] \frac{\partial Q_{\mathbf{w}}(s, a)}{\partial \mathbf{w}} \\ \bar{\mathbf{w}} &\leftarrow \mathbf{w} \end{aligned}$$

- 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:

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

$$\bar{\mathbf{w}} \leftarrow \mathbf{w}$$

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 \mathbf{w} and $\bar{\mathbf{w}}$ 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 (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

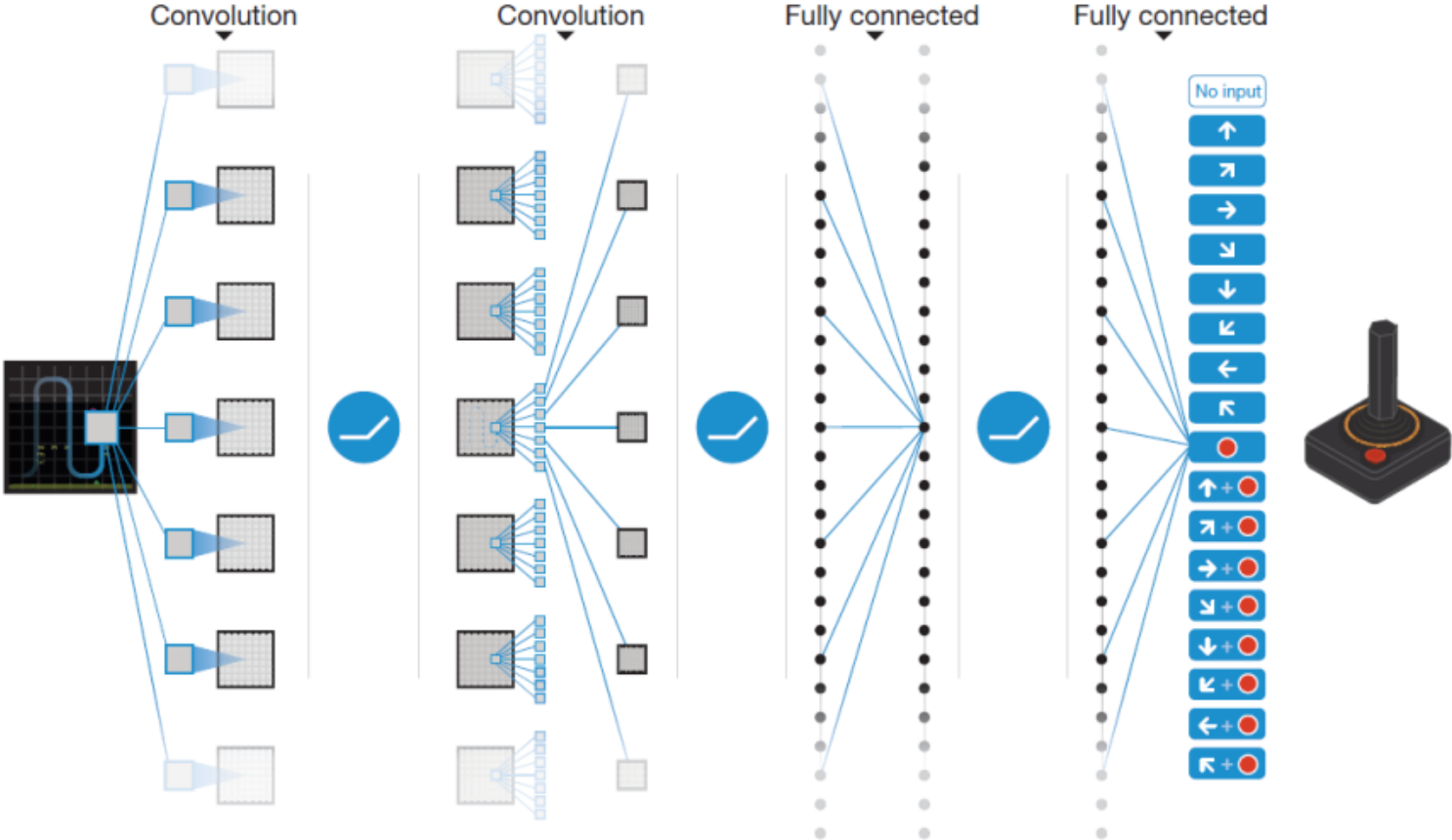
$$\text{Gradient: } \frac{\partial \text{Err}}{\partial \mathbf{w}} = \left[Q_{\mathbf{w}}(\hat{s}, \hat{a}) - \hat{r} - \gamma \max_{\hat{a}'} Q_{\bar{\mathbf{w}}}(\hat{s}', \hat{a}') \right] \frac{\partial Q_{\mathbf{w}}(\hat{s}, \hat{a})}{\partial \mathbf{w}}$$

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

 Update state: $s \leftarrow s'$

 Every c steps, update target: $\bar{\mathbf{w}} \leftarrow \mathbf{w}$

Deep Q-Network for Atari



DQN versus Linear Approximation

