# Lecture 11: Deep Reinforcement Learning
# CS486/686 Intro to Artificial Intelligence

Pascal Poupart
David R. Cheriton School of Computer Science
CIFAR AI Chair at 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

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

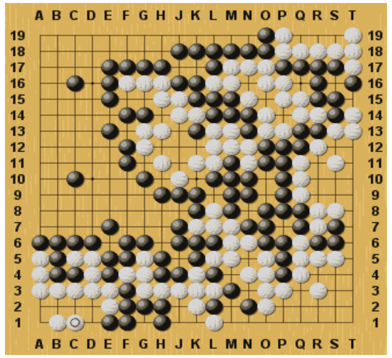
- Reinforcement learning: Q-learning

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

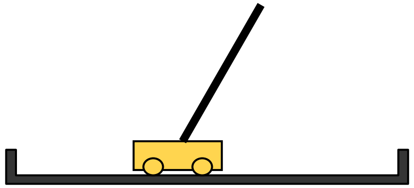
- Complexity depends on number of states and actions

# Large State Spaces

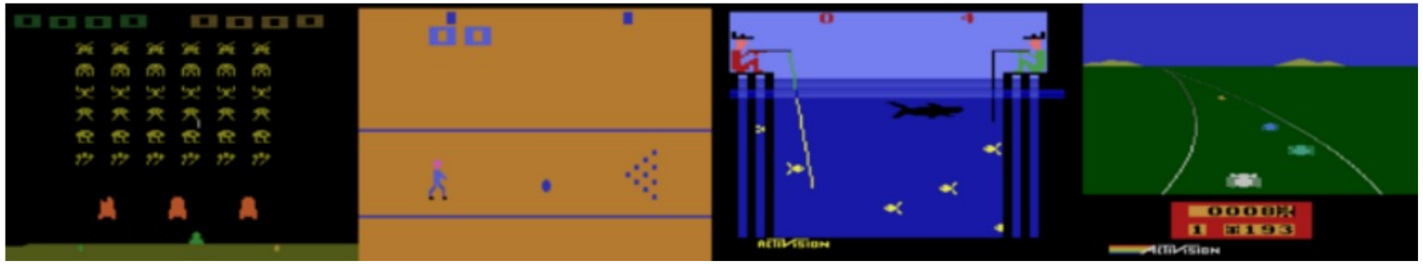
- Computer Go: $3^{361}$ states



- Inverted pendulum: $(x, x', \theta, \theta')$
  - 4-dimensional continuous state space



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



# Functions to be Approximated

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

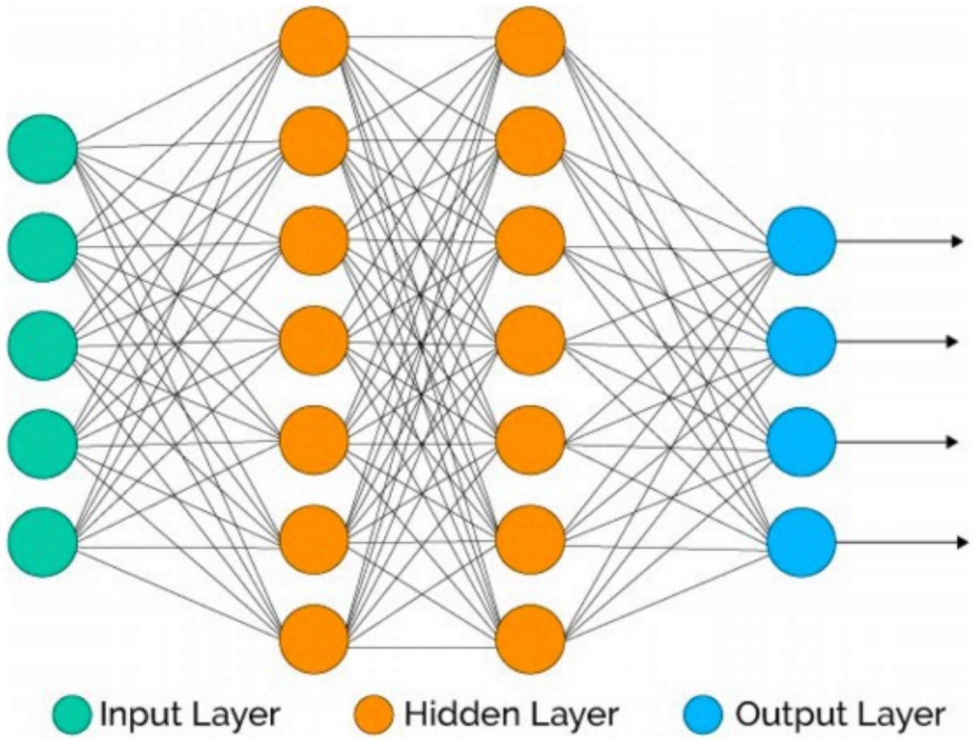
- Q-function: $Q(s,a) \in \Re$

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

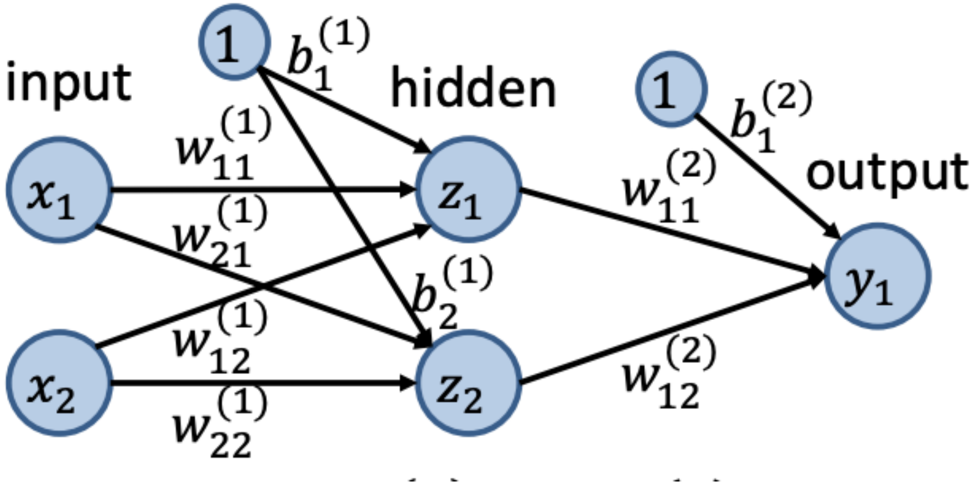
# Traditional Neural Network

- 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): $h$



# One Hidden Layer Architecture




- Feed-forward neural network
  - Hidden units: $z_j = h_1(\boldsymbol{w}_j^{(1)}\boldsymbol{x} + b_j^{(1)})$
  - Output units: $y_k = h_2(\boldsymbol{w}_k^{(2)}\boldsymbol{z} + b_k^{(2)})$
  - 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: $h(\boldsymbol{a})_i = \frac{e^{a_i}}{\sum_j e^{a_j}}$
- 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 $s = (x_1, x_2, \ldots, x_n)^T$

- 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: $r + \gamma \max_{a'} Q_{\overline{\boldsymbol{w}}}(s', a')$

$\overline{\boldsymbol{w}}$ fixed

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

# 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: $\frac{\partial Err}{\partial \boldsymbol{w}} = \left[ Q_{\boldsymbol{w}}(s,a) - r - \gamma \max_{a'} Q_{\boldsymbol{w}}(s',a') \right] \frac{\partial Q_{\boldsymbol{w}}(s,a)}{\partial \boldsymbol{w}}$
- 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 $\alpha(s,a) = 1/n(s,a)$
  - 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:

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

- Let $\alpha_t = 1/t$
- 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

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

  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 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:

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

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

- 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:

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

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

# Deep Q-network (DQN)

- Deep Mind

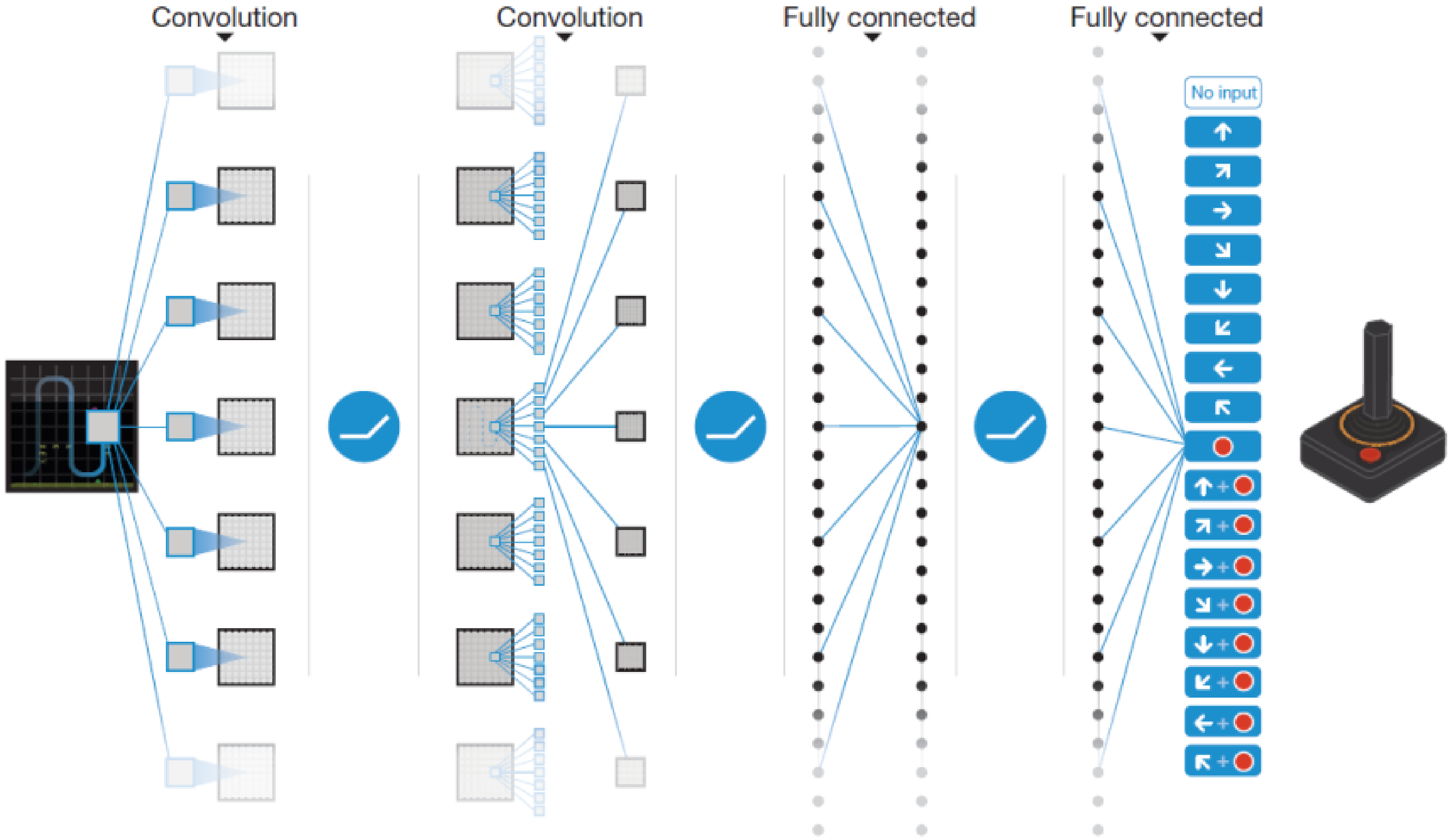
- Deep Q-network: Gradient Q-learning with
  - Deep neural networks
  - Experience replay
  - Target network

- 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 $(s, a, s', r)$ 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: $\frac{\partial Err}{\partial \boldsymbol{w}} = \left[Q_{\boldsymbol{w}}(\hat{s}, \hat{a}) - \hat{r} - \gamma \max_{\hat{a}'} Q_{\overline{\mathbf{w}}}(\hat{s}', \hat{a}')\right] \frac{\partial Q_{\boldsymbol{w}}(\hat{s},\hat{a})}{\partial \boldsymbol{w}}$
        Update weights: $\boldsymbol{w} \leftarrow \boldsymbol{w} - \alpha \frac{\partial Err}{\partial \boldsymbol{w}}$
    Update state: $s \leftarrow s$'
    Every $c$ steps, update target: $\overline{\boldsymbol{w}} \leftarrow \boldsymbol{w}$

# Deep Q-Network for Atari



# DQN versus Linear Approximation