Self-Driving Cars

Lecture 4 - Reinforcement Learning

Benjamin Coors

Autonomous Vision Group
MPI-IS / University of Tübingen

November 22, 2018

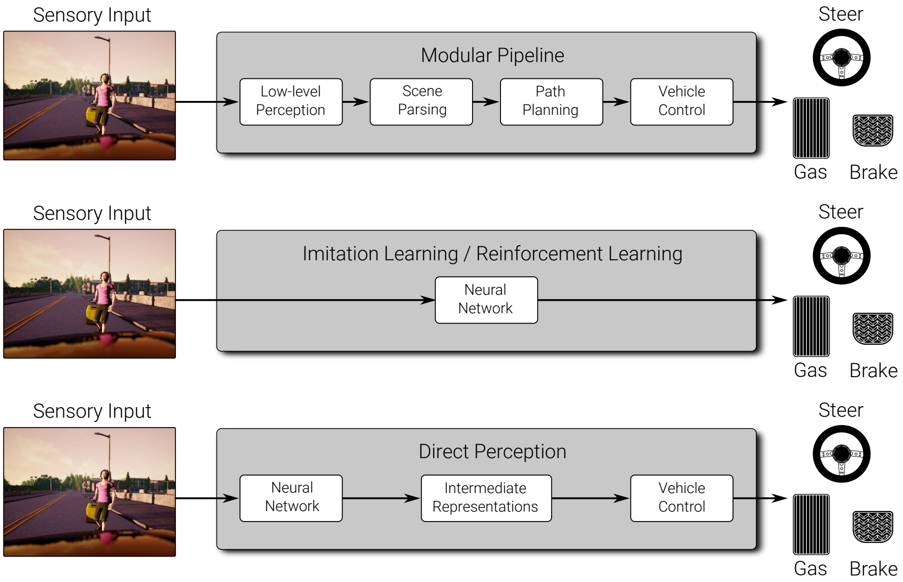
University of Tübingen
MPI for Intelligent Systems
Autonomous Vision Group

Agenda

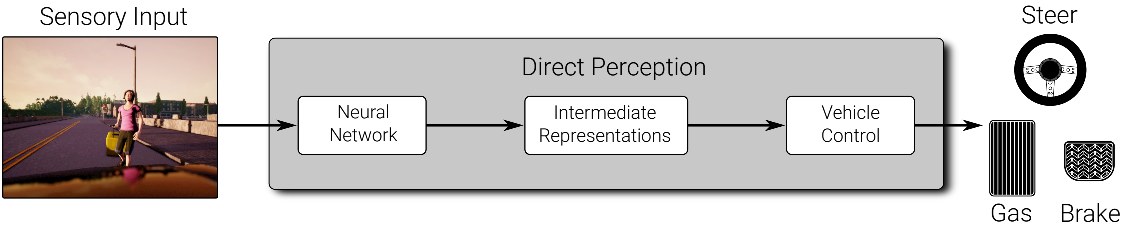
Date	Lecture (Thursday)	Date	Exercise (Friday)
18.10.	01 - Introduction to Self-Driving Cars	19.10.	00 - Introduction Pytorch & OpenAI Gym
25.10.	02 - DNNs, ConvNets, Imitation Learning	26.10.	01 - Intro: Imitation Learning
1.11.	none (Allerheiligen)	2.11.	
8.11.	03 - Direct Perception	9.11.	01 - Q&A
15.11.	none (CVPR Deadline)	16.11.	
22.11.	04 - Reinforcement Learning	23.11.	01 - Discussion & 02 - Intro: Reinforcement Learning
29.11.	05 - Vehicle Dynamics & Control	30.11.	
6.12.	06 - Localization & Visual Odometry	7.12.	02 - Q&A
13.12.	07 - Simultaneous Localization and Mapping (J. Stückler)	14.12.	
20.12.	08 - Road and Lane Detection	21.12.	02 - Discussion & 03 - Intro: Modular Pipeline
10.1.	09 - Reconstruction and Motion Estimation	11.1.	
17.1.	10 - Object Detection & Tracking	18.1.	
24.1.	11 - Scene Understanding	25.1.	03 - Q&A
31.1.	12 - Planning	1.2.	03 - Discussion & Announcement of Winners
7.2.	13 - Winner’s Presentations and Exam Q&A	8.2.	

Recap

Approaches to Self-Driving



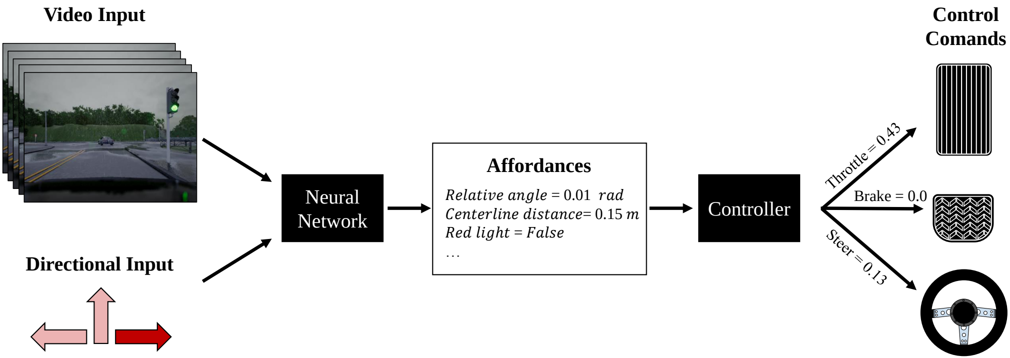
Direct Perception



Idea of Direct Perception:

- Learn to predict interpretable low-dimensional intermediate representation
- Exploit classical controllers and finite state machines
- Hybrid model between imitation learning and modular pipelines

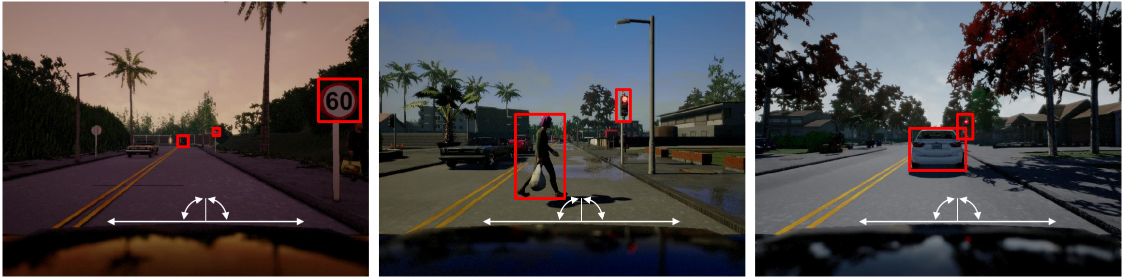
Conditional Affordance Learning



Affordances: [Gibson, 1966]

- Attributes of the environment which limit the space of allowed actions

Conditional Affordance Learning: Affordances



Affordances:

- ▶ Distance to centerline
- ▶ Relative angle to road
- ▶ Distance to lead vehicle
- ▶ Speed signs
- ▶ Traffic lights
- ▶ Hazard stop

Sauer, Savinov and Geiger: Conditional Affordance Learning for Driving in Urban Environments. CoRL, 2018.

Reinforcement Learning

Reinforcement Learning

So far:

- Supervised learning, lots of data-label pairs required
- Use of auxiliary, short-term loss functions
  - Imitation learning: per-frame loss on action
  - Direct perception: per-frame loss on affordance indicators

Now:

- Learning of models based on the loss that we actually care about, e.g.:
  - Minimize time to target location
  - Minimize number of collisions
  - Minimize risk
  - Maximize comfort
  - etc.

Types of Learning

Supervised Learning:

- Dataset: $\{(x_i, y_i)\}$ (x_i = data, y_i = label) Goal: Learn mapping $x \mapsto y$
- Examples: Classification, regression, imitation learning, affordance learning, etc.

Unsupervised Learning:

- Dataset: $\{(x_i)\}$ (x_i = data) Goal: Discover structure underlying data
- Examples: Clustering, dimensionality reduction, feature learning, etc.

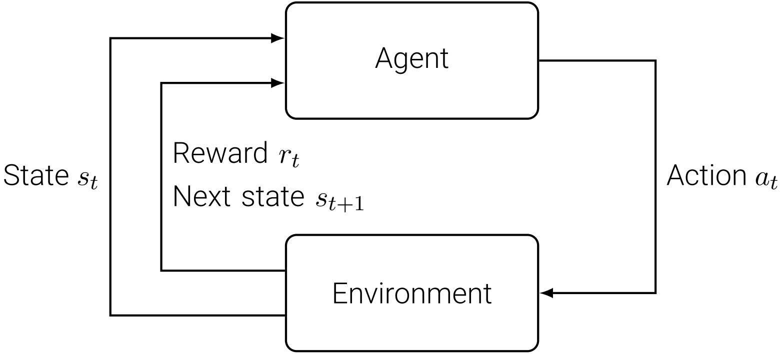
Reinforcement Learning:

- Agent interacting with environment which provides numeric reward signals
- Goal: Learn how to take actions in order to maximize reward
- Examples: Learning of manipulation or control tasks (everything that interacts)

Reinforcement Learning

Introduction

Reinforcement Learning Overview



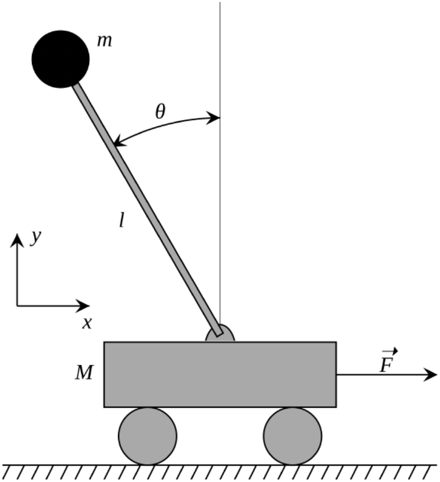
- Agent oberserves environment state s_t at time t
- Agent sends action a_t at time t to the environment
- Environment returns the reward r_t and its new state s_{t+1} to the agent

Reinforcement Learning Overview

Sequential decision making:

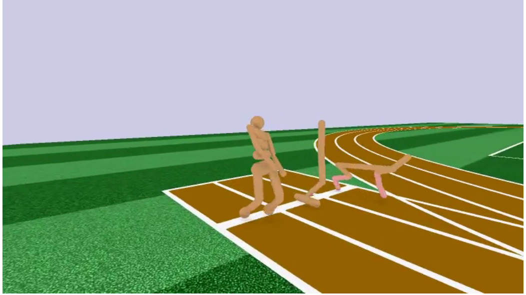
- Goal: Select actions to maximize total future reward
- Actions may have long term consequences
- Reward may be delayed, not instantaneous
- It may be better to sacrifice immediate reward to gain more long-term reward
- Examples:
  - Financial investment (may take months to mature)
  - Refuelling a helicopter (might prevent crash in several hours)
  - Blocking opponent moves (might help winning chances in the future)

Example: Pole-Balancing



- **Objective:** Balance pole on moving cart
- **State:** Angle, angular vel., position, vel.
- **Action:** Horizontal force applied to cart
- **Reward:** 1 if pole is upright at time t

Example: Robot Locomotion



http://blog.openai.com/roboschool/

- **Objective:** Make robot move forward
- **State:** Position and angle of joints
- **Action:** Torques applied on joints
- **Reward:** 1 if upright and forward moving

Example: Atari Games

http://blog.openai.com/gym-retro/

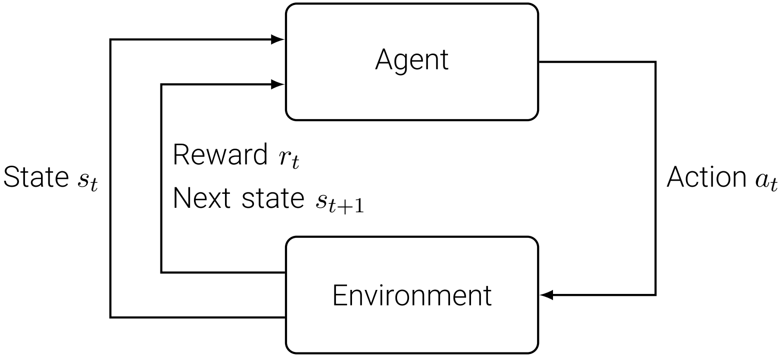
- **Objective:** Maximize score
- **State:** Raw pixels of screen (210x160)
- **Action:** Left, right, up, down
- **Reward:** Score increase/decrease at t

Example: Go

www.deepmind.com/research/alphago/

- **Objective:** Win the game!
- **State:** Position of all pieces
- **Action:** Location of next piece
- **Reward:** 1 if game won, 0 otherwise

Reinforcement Learning: Overview



- How can we mathematically formalize the RL problem?

Markov Decision Process

Markov Decision Process (MDP) defined by tuple:

$$(\mathcal{S}, \mathcal{A}, \mathcal{R}, P, \gamma)$$

- $\mathcal{S}$: set of possible states
- $\mathcal{A}$: set of possible actions
- $\mathcal{R}$: distribution of reward given (state,action) pair
- P: distribution over next state given (state,action) pair
- γ: discount factor

Almost all reinforcement learning problems can be formalized as MDPs

Markov Decision Process

Markov property: Current state completely characterizes state of the world

- A state s_t is *Markov* if and only if

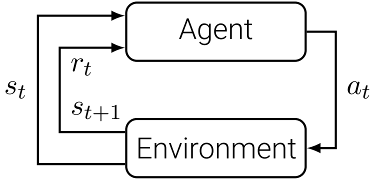
$$P(s_{t+1}|s_t) = P(s_{t+1}|s_1, ..., s_t)$$

- ”The future is independent of the past given the present”
- The state captures all relevant information from the history
- Once the state is known, the history may be thrown away
- i.e. the state is a sufficient statistic of the future

Markov Decision Process

Reinforcement learning interaction loop:

- At time $t = 0$:
  - Environment samples initial state $s_0 \sim P(s_0)$
- Then, for $t = 0$ until done:
  - Agent selects action a_t
  - Environment samples reward $r_t \sim \mathcal{R}(\cdot|s_t, a_t)$
  - Environment samples next state $s_{t+1} \sim P(\cdot|s_t, a_t)$
  - Agent receives reward r_t and next state s_{t+1}



How do we select an action?

Policy

A **policy** π is a function from $\mathcal{S}$ to $\mathcal{A}$ that specifies what action to take in each state:

- A policy fully defines the behavior of an agent
- Deterministic policy: $a = \pi(s)$
- Stochastic policy: $\pi(a|s) = P(a_t = a|s_t = s)$
- MDP policies depend on the current state (not the history)

Policy

How do we learn a policy?

Imitation Learning: Learn a policy from expert demonstrations

- Expert demonstrations are provided
- Supervised learning problem

Reinforcement Learning: Learn a policy through *trial-and-error*

- No expert demonstrations given
- Agent has to discover itself which actions maximize its reward
  - The agent interacts with the environment and obtains reward
  - The agent discovers good actions and improves its policy π
- **Goal:** Learn a policy which maximizes the total future reward

Exploration and Exploitation

How do we discover good actions?

Answer: We need to explore the action space

- **Exploration:** Try a novel action a in state s , observe reward r_t
  - Discovers more information about the environment
  - Game-playing example: Play a novel experimental move
- **Exploitation:** Use a previously discovered good action a
  - Exploits known information to maximize reward
  - Game-playing example: Play the move you believe is best

Trade-off: It is usually important to explore as well as exploit

Exploration and Exploitation

How to balance exploration and exploitation?

ϵ**-greedy** exploration algorithm:

- All m actions are tried with non-zero probability
- With probability ϵ choose an action at random (**exploration**)
- With probability $1 - \epsilon$ choose the greedy action (**exploitation**)
- Greedy action is defined as best action which was discovered so far
- ϵ is gradually annealed over time

Value Functions

How good is a state?

The **state-value function** V^{π} at state s_t is the expected cumulative discounted reward $(r_t \sim \mathcal{R}(\cdot|s_t, a_t))$ when following the policy π from state s_t:

$$\begin{aligned} V^{\pi}(s_t) &= \mathbb{E}[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + ...|s_t, \pi] \\ &= \mathbb{E}\left[\sum_{t \geq 0} \gamma^t r_t \middle| s_t, \pi\right] \end{aligned}$$

- The discount $\gamma \in [0, 1]$ is the present value of future rewards
    - Weights immediate reward higher than future reward
    - Determines agent's far/short-sightedness
    - Avoids infinite returns in cyclic Markov processes

Value Functions

How good is a state-action pair?

The **action-value function** Q^{π} at state s_t and action a_t is the expected cumulative discounted reward from taking action a_t in state s_t and then following the policy π:

$$Q^{\pi}(s_t, a_t) = \mathbb{E}\left[\sum_{t \geq 0} \gamma^t r_t \middle| s_t, a_t, \pi\right]$$

- The discount $\gamma \in [0, 1]$ is the present value of future rewards
  - Weights immediate reward higher than future reward
  - Determines agent's far/short-sightedness
  - Avoids infinite returns in cyclic Markov processes

Optimal Value Functions

The **optimal state-value function** $V^*(s_t)$ is the best $V^\pi(s_t)$ over all policies π:

$$V^*(s_t) = \max_\pi V^\pi(s_t) \qquad V^\pi(s_t) = \mathbb{E}\left[\sum_{t\geq 0} \gamma^t r_t \middle| s_t, \pi\right]$$

The **optimal action-value function** $Q^*(s_t, a_t)$ is the best $Q^\pi(s_t, a_t)$ over all policies π:

$$Q^*(s_t, a_t) = \max_\pi Q^\pi(s_t, a_t) \qquad Q^\pi(s_t, a_t) = \mathbb{E}\left[\sum_{t\geq 0} \gamma^t r_t \middle| s_t, a_t, \pi\right]$$

- The optimal value functions specify the best possible performance in the MDP
- However, optimizing over all possible policies π is computationally intractable

Optimal Policy

An **optimal policy** can be found by maximizing over $Q^*(s_t, a_t)$:

$$\pi^*(a_t|s_t) = \begin{cases} 1 & \text{if } a_t = \underset{a' \in \mathcal{A}}{\operatorname{argmax}} \; Q^*(s_t, a') \\ 0 & \text{otherwise} \end{cases}$$

- The optimal policy is better than or equal to all other policies:

$$\pi^* \geq \pi, \; \forall \pi$$

where $\pi \geq \pi'$ if $V^\pi(s_t) \geq V^{\pi'}(s_t)$ for all $s_t \in \mathcal{S}$

A Simple Grid World Example

actions = $\{$

1. right $\rightarrow$
2. left $\leftarrow$
3. up $\uparrow$
4. down $\downarrow$

$\}$

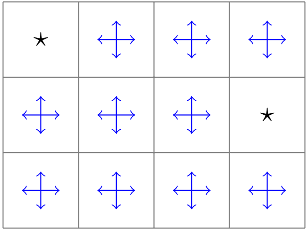
states

$\star$			
			$\star$

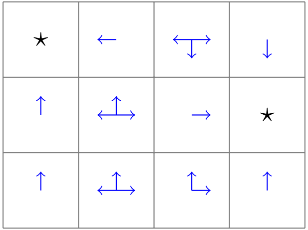
reward: $r = -1$ for each transition

Objective: Reach one of terminal states (marked by a $\star$) in least number of actions

A Simple Grid World Example



Random Policy



Optimal Policy

Solving for the Optimal Policy

Bellman Optimality Equation

The **Bellman optimality equation** decomposes Q^* into two parts:

$$\begin{aligned} Q^*(s_t, a_t) &= \mathbb{E}\left[r_t + \gamma r_{t+1} + ... + \gamma^n r_n | s_t, a_t\right] \\ &\overset{BOE}{=} \mathbb{E}\left[r_t + \gamma \max_{a' \in \mathcal{A}} Q^*(s_{t+1}, a') \middle| s_t, a_t\right] \end{aligned}$$

Recursive formulation comprises:

- Current reward: r_t
- Discounted optimal action-value of successor: $\gamma \max_{a' \in \mathcal{A}} Q^*(s_{t+1}, a')$

Solving the Bellman optimality equation:

- Bellman Optimality Equation is non-linear
- No closed form solution (in general)
- Many iterative solution methods

Q-learning

Q-learning algorithm: Iteratively solve for Q^*

$$Q_{\color{red}{i+1}}(s_t, a_t) = \mathbb{E}\left[r_t + \gamma \max_{a' \in \mathcal{A}} Q_{\color{red}{i}}(s_{t+1}, a') \middle| s_t, a_t\right]$$

- Approximate the expectation with a single-sample iterative update:

$$Q_{i+1}(s_t, a_t) \leftarrow Q_i(s_t, a_t) + \alpha \underbrace{\Big(\underbrace{r_t + \gamma \max_{a' \in \mathcal{A}} Q_i(s_{t+1}, a')}_{\text{target}} - \underbrace{Q_i(s_t, a_t)}_{\text{prediction}}\Big)}_{\text{temporal difference (TD) error}}$$

  with learning rate α

- Q_i will converge to Q^* as $i \to \infty$

Watkins and Dayan: Technical Note Q-Learning. Machine Learning, 1992.

Q-learning

Implementation:

- Initialize a Q-table with all zero entries
- Repeat:
  - Observe state s_t, choose action a_t according to ϵ-greedy strategy
  - Observe reward r_t and next state s_{t+1}
  - Compute TD error: $r_t + \gamma \max_{a' \in \mathcal{A}} Q_i(s_{t+1}, a') - Q_i(s_t, a_t)$
  - Update Q-table

What's the problem with using Q-tables?

Not scalable: Q-tables don't scale to high dimensional state or action spaces

Solution: Use a function approximator to estimate $Q(s, a)$, e.g. a neural network!

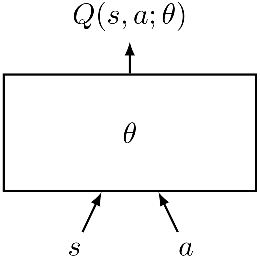
Watkins and Dayan: Technical Note Q-Learning. Machine Learning, 1992.

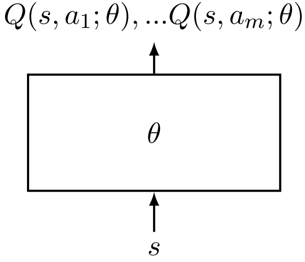
Deep Reinforcement Learning

Deep Q-Learning

Use a **deep neural network** with weights θ as function approximator to estimate Q:

$$Q(s, a; \theta) \approx Q^*(s, a)$$





Mnih et al.: Human-level control through deep reinforcement learning. Nature, 2015.

Deep Q-Learning

Forward Pass:

Loss function is the mean-squared error in Q-values:

$$L_i(\theta_i) = \mathbb{E}\left[\left((r + \gamma \max_{a'} Q(s', a'; \theta_i)) - Q(s, a; \theta_i)\right)^2\right]$$

Backward Pass:

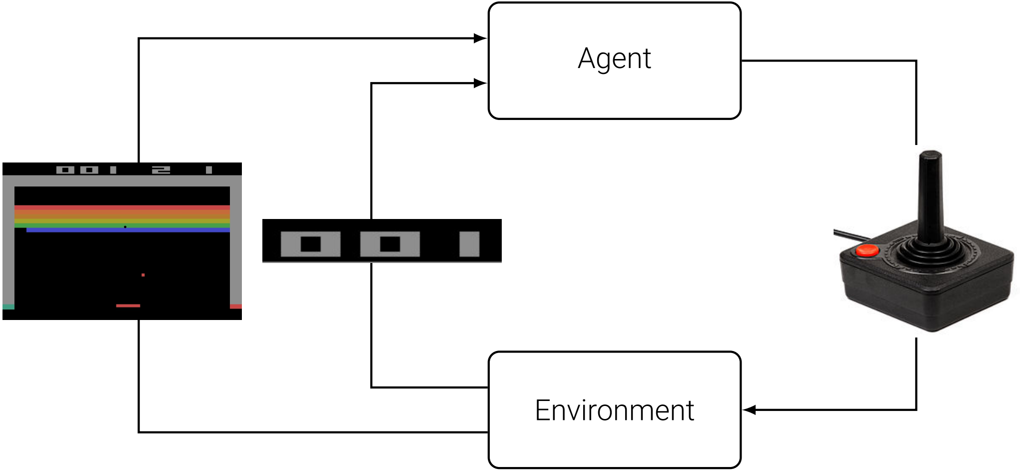
Gradient update with respect to Q-function parameters θ:

$$\nabla_{\theta_i} L_i(\theta_i) = \mathbb{E}\left[\left(r + \gamma \max_{a'} Q(s', a'; \theta_i) - Q(s, a; \theta_i)\right) \nabla_{\theta_i} Q(s, a; \theta_i)\right]$$

Optimize objective end-to-end with stochastic gradient descent (SGD) using $\nabla_{\theta_i} L_i(\theta_i)$

Mnih et al.: Human-level control through deep reinforcement learning. Nature, 2015.

Case Study: Playing Atari Games



Objective: Complete the game with the highest score

Mnih et al.: Human-level control through deep reinforcement learning. Nature, 2015.

Q-network Architecture

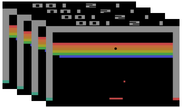
$Q(s, a; \theta)$: Neural network with weights θ

FC-Out (Q-values)

FC-256

32 4x4 conv, stride 2

16 8x8 conv, stride 2



Output: Q-values for all 4-18 Atari actions

Input: $84 \times 84 \times 4$ stack of last 4 frames
(after RGB to grayscale, downsampling, cropping)

Efficient: A single forward pass computes the Q-values for all actions!

Mnih et al.: Human-level control through deep reinforcement learning. Nature, 2015.

Training the Q-network: Loss Function (from before)

Forward Pass:

Loss function is the mean-squared error in Q-values:

$$L_i(\theta_i) = \mathbb{E}\left[\left((r + \gamma \max_{a'} Q(s', a'; \theta_i)) - Q(s, a; \theta_i)\right)^2\right]$$

Backward Pass:

Gradient update with respect to Q-function parameters θ:

$$\nabla_{\theta_i} L_i(\theta_i) = \mathbb{E}\left[\left(r + \gamma \max_{a'} Q(s', a'; \theta_i) - Q(s, a; \theta_i)\right) \nabla_{\theta_i} Q(s, a; \theta_i)\right]$$

Optimize objective end-to-end with stochastic gradient descent (SGD) using $\nabla_{\theta_i} L_i(\theta_i)$

Mnih et al.: Human-level control through deep reinforcement learning. Nature, 2015.

Training the Q-network: Experience Replay

Unlike in standard Q-learning, we now train on **mini-batches**:

- Problem: Learning from consecutive samples is inefficient
- Reason: Strong correlations between consecutive samples

Experience replay stores agent's experiences at each time-step

- Continually update a **replay memory** D with new experiences $e_t = (s_t, a_t, r_t, s_{t+1})$
- Train on samples $(s, a, r, s') \sim U(D)$ drawn uniformly at random from D
- Breaks correlations between samples
- Improves data efficiency as each sample can be used multiple times

In practice, a circular replay memory of finite memory size N is used

Mnih et al.: Human-level control through deep reinforcement learning. Nature, 2015.

Training the Q-network: Fixed Q-targets

Challenge: **Non-stationary targets**

- As the policy changes, so do our targets: $r + \gamma \max_{a'} Q(s', a'; \theta_i)$
- This may lead to oscillation or divergence

Fixed Q-targets are used to stabilize the training:

- A second target network $\hat{Q}$ with weights θ^- is used to generate the targets:

$$L_i(\theta_i) = \mathbb{E}_{(s,a,r,s')\sim U(D)} \left[\left(r + \gamma \max_{a'} \hat{Q}(s', a'; \theta_i^-) - Q(s, a; \theta_i) \right)^2 \right]$$

- Target network $\hat{Q}$ is only updated every C steps by cloning the Q-network
- Adds a delay between updating Q and updating the targets
- Effect: Oscillations or divergence of the policy is reduced

Mnih et al.: Human-level control through deep reinforcement learning. Nature, 2015.

Putting it together: Deep Q-Learning with Experience Replay

Training a deep Q-network using **experience replay** and **fixed Q-targets**

- Take action a_t according to ϵ-greedy policy
- Store transition (s_t, a_t, r_t, s_{t+1}) in replay memory D
- Sample random mini-batch of transitions (s, a, r, s') from D
- Compute Q-learning targets w.r.t. old, fixed parameters θ^-
- Optimize MSE between Q-network predictions and Q-learning targets:

$$L_i(\theta_i) = \mathbb{E}_{s,a,r,s' \sim D_i} \left[\left(r + \gamma \max_{a'} \hat{Q}(s', a'; \theta_i^-) - Q(s, a; \theta_i) \right)^2 \right]$$

using a variant of stochastic gradient descent

Mnih et al.: Human-level control through deep reinforcement learning. Nature, 2015.

Deep Q-Learning: Breakout

Mnih et al.: Human-level control through deep reinforcement learning. Nature, 2015.

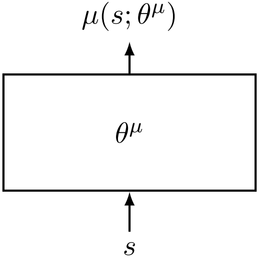
Deep Q-Learning Shortcomings

There are several **poinst for improvement** in the initial deep Q-learning approach:

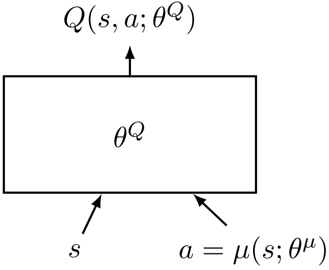
- Long training times
- Action space is limited to a discrete set of actions
- Uniform sampling from experience replay gives equal importance to all transitions
- Simplistic exploration strategy
- Overestimation of action-values
- Relies on fully-observable states

Continuous Control with Deep Reinforcement Learning

Use two networks: an **actor** and a **critic**



Actor



Critic

Lillicrap et al.: Continuous Control with Deep Reinforcement Learning. ICLR, 2016.

Continuous Control with Deep Reinforcement Learning

Use two networks: an **actor** and a **critic**

- **Actor** network with weights θ^μ estimates agent's deterministic policy $\mu(s;\theta^\mu)$
  - Update policy in direction that most improves Q
  - i.e. backpropagate critic through actor:

$$\begin{aligned}\nabla_{\theta^\mu} J_i &\approx \mathbb{E}_{s_i}[\nabla_{\theta^\mu} Q(s,a;\theta_i^Q)] \\ &= \mathbb{E}_{s_i}[\nabla_a Q(s,a;\theta_i^Q)\nabla_{\theta^\mu}\mu(s;\theta_i^\mu)]\end{aligned}$$

- **Critic** estimates value of current policy $Q(s,a;\theta^Q)$
  - Learned using the Bellman optimality equation as in Q-learning:

$$L_i(\theta_i^Q) = \mathbb{E}_{(s,a,r,s')}[(r + \gamma Q'(s,\mu'(s;\theta_i^{\mu'});\theta_i^{Q'}) - Q(s,a;\theta_i^Q))^2]$$

Lillicrap et al.: Continuous Control with Deep Reinforcement Learning. ICLR, 2016.

Continuous Control with Deep Reinforcement Learning

Experience replay and **target networks** are again used to stabilize the training:

- Replay memory D again stores transition tuples (s_t, a_t, r_t, s_{t+1})
- Target networks are updated using ”soft” target updates
  - Weights are not directly copied but slowly adapted:

$$\begin{aligned} \theta^{Q'} &\leftarrow \tau\theta^{Q} + (1-\tau)\theta^{Q'} \\ \theta^{\mu'} &\leftarrow \tau\theta^{\mu} + (1-\tau)\theta^{\mu'} \end{aligned}$$

where $0 < \tau << 1$ controls the tradeoff between speed and stability of learning

Exploration is performed by adding temporally correlated noise $\mathcal{O}$ to $\mu(s)$:

$$\mu'(s_t) = \mu(s_t; \theta_t^{\mu}) + \epsilon\mathcal{O}$$

Lillicrap et al.: Continuous Control with Deep Reinforcement Learning. ICLR, 2016.

Prioritized Experience Replay

Prioritize experience to replay important transitions more frequently

- Priority δ is measured by magnitude of temporal difference (TD) error:

$$\delta = |r + \gamma \max_{a'} Q(s', a'; \theta_i^-) - Q(s, a; \theta_i)|$$

- TD error measures how 'surprising' or unexpected the transition is
- Stochastic prioritization avoids overfitting due to lack of diversity
- Enables learning speed-up by a factor of 2 on Atari benchmarks

Schaul et al.: Prioritized Experience Replay. ICLR, 2016.

Faulty Reward Functions

https://blog.openai.com/faulty-reward-functions/

Further Readings

- Sutton and Barto: Reinforcement Learning: An Introduction. MIT Press, 2017.
- Watkins and Dayan: Technical Note Q-Learning. Machine Learning, 1992.
- Mnih et al.: Human-level control through deep reinforcement learning. Nature, 2015.
- Lillicrap et al.: Continuous Control with Deep Reinforcement Learning. ICLR, 2016.

Next Time:

Vehicle Dynamics & Control

Questions?