LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

Praktikum Autonome Systeme

Function Approximation

Prof. Dr. Claudia Linnhoff-Popien
Thomy Phan, Andreas Sedlmeier, Fabian Ritz
http://www.mobile.ifi.lmu.de

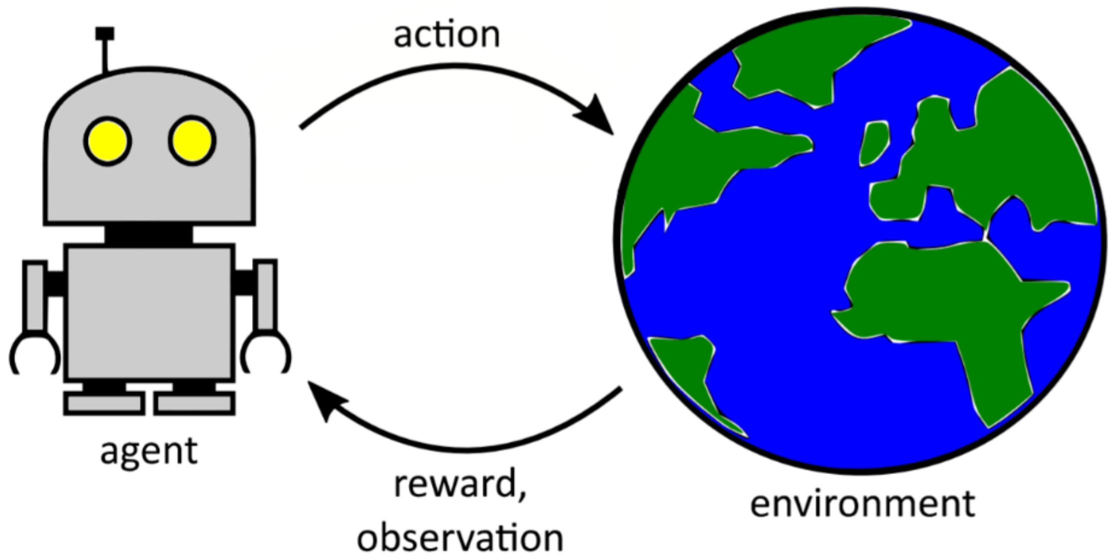
WiSe 2019/20

https://openmunich.eu/

→ Recaps

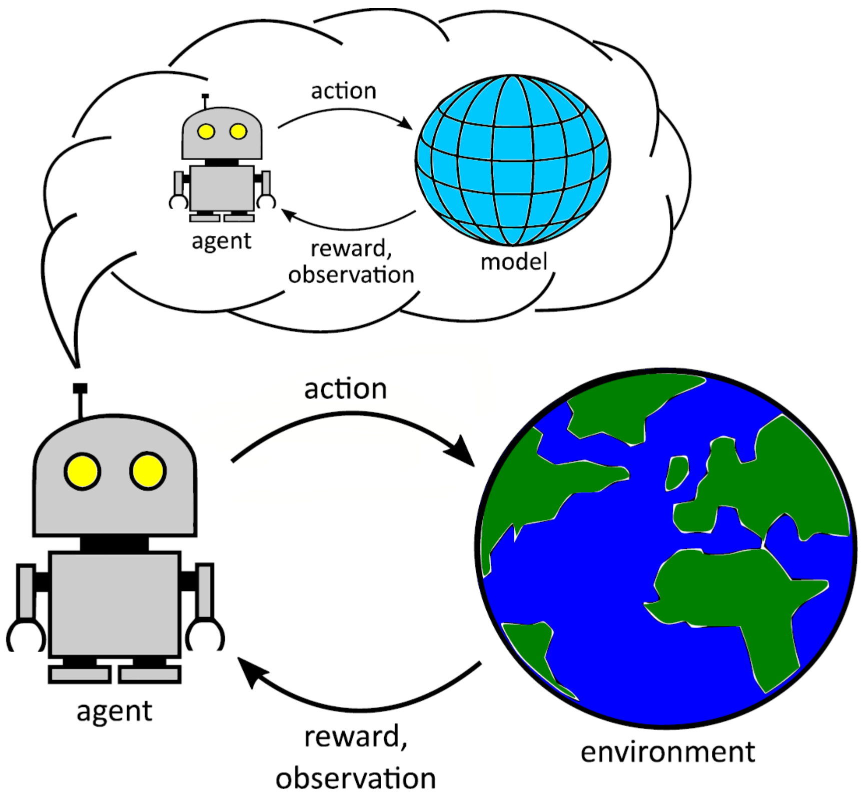
Recap: Sequential Decision Making

- **Goal:** Autonomously select actions to solve a (complex) task
  - time is important (actions might have **long term** consequences)
  - maximize the **expected cumulative reward** for each state



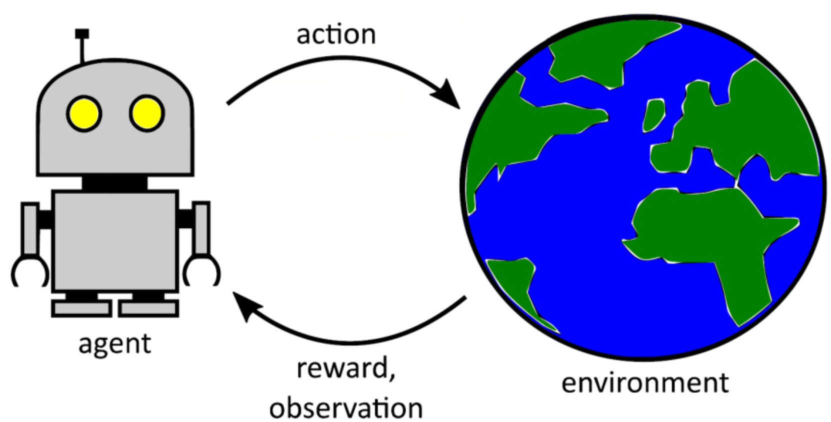
Recap: Automated Planning

- **Goal:** Find (near-)**optimal policies** π^* to solve complex problems
- Use (heuristic) lookahead search on a **given model** $\widehat{M} \approx M$ of the problem



Recap: Reinforcement Learning (1)

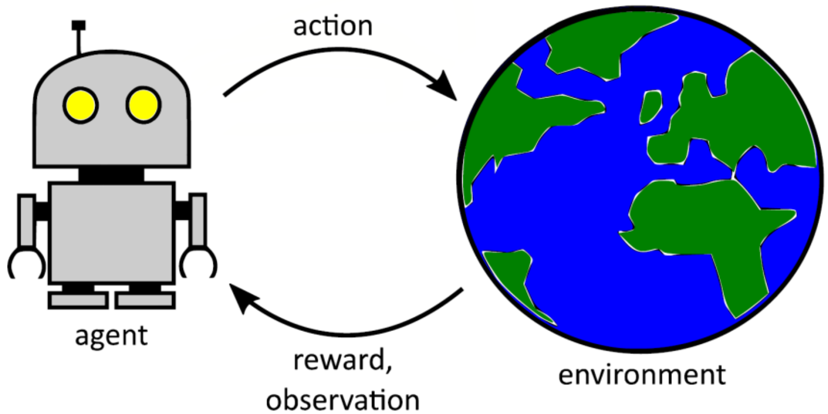
- **Goal:** Find an (near-)optimal policy to solve complex problems



- Learn via trial-error from (real) experience:
  - Model $\mathcal{P}(s_{t+1}|s_t, a_t)$ is unknown
  - **Experience samples** $e_t = \langle s_t, a_t, r_t, s_{t+1} \rangle$ are generated by interacting with a (real or simulated) environment
  - To obtain sufficient experience samples one has to balance between **exploration** and **exploitation** of actions

Recap: Reinforcement Learning (2)

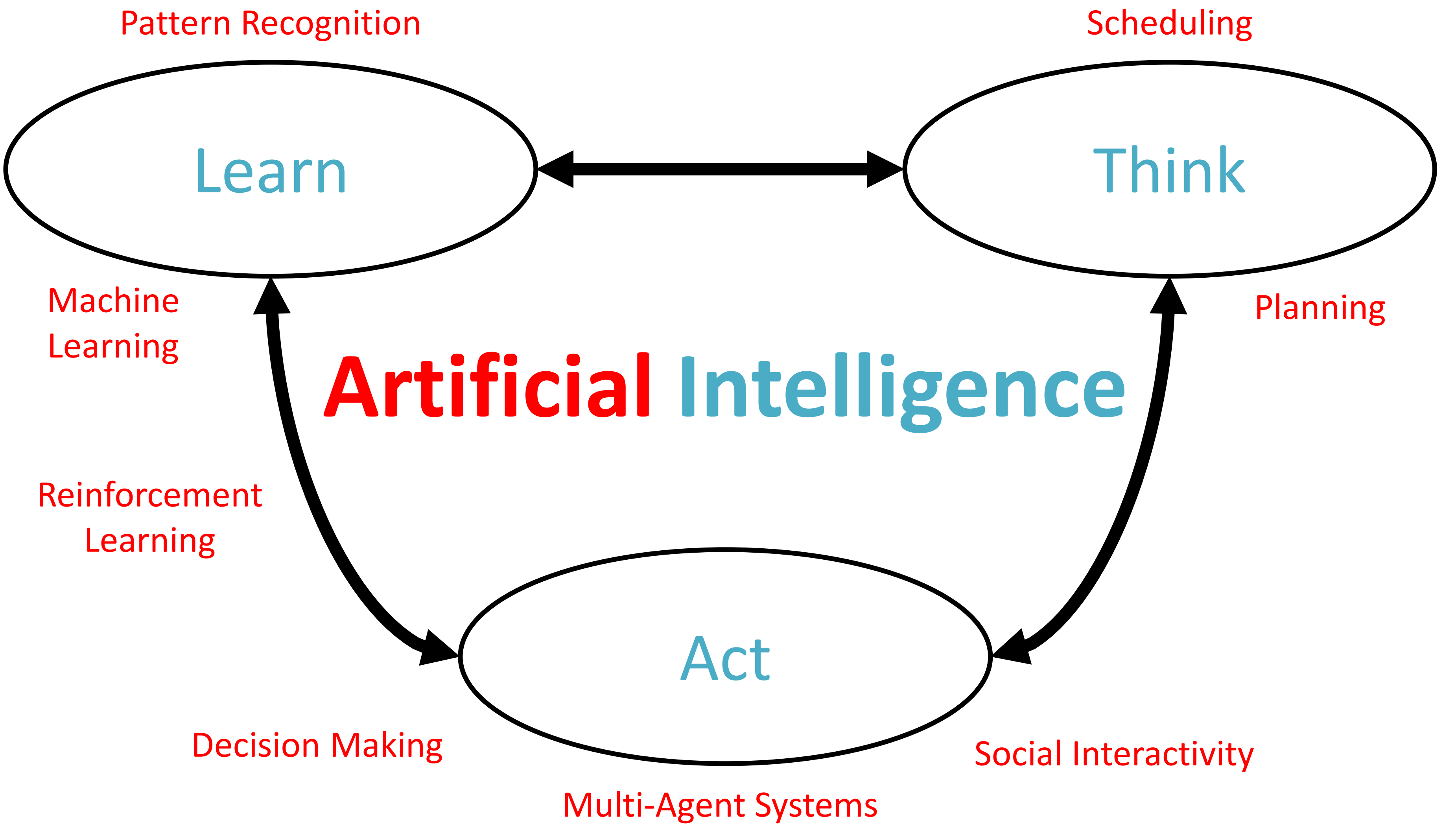
- **Goal:** Find an (near-)optimal policy to solve complex problems

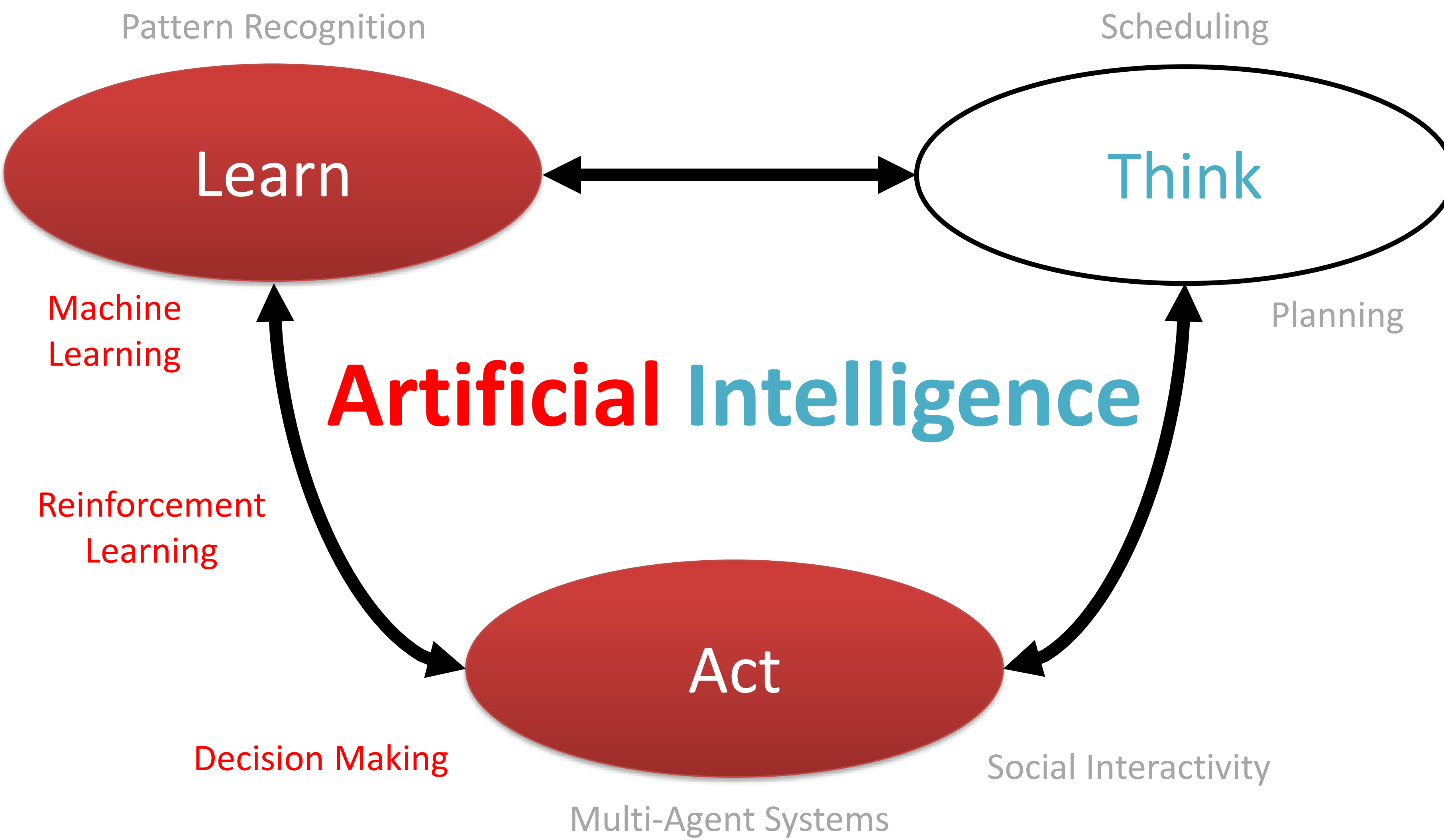


1. Model Free Prediction = Policy Evaluation (estimate V^{π} given π)
2. Model Free Control = Policy Improvement (improve π given V^{π})

- Temporal Difference vs. Monte Carlo Learning?
- On-Policy vs. Off-Policy Learning?

→ Large Scale Reinforcement Learning

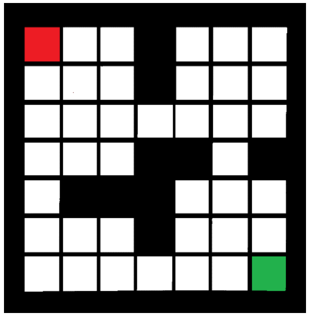

Pattern Recognition
Scheduling
Learn
Think
Machine Learning
Planning
Artificial Intelligence
Reinforcement Learning
Act
Decision Making
Social Interactivity
Multi-Agent Systems


Pattern Recognition
Scheduling
Learn
Think
Machine Learning
Planning
Artificial Intelligence
Reinforcement Learning
Act
Decision Making
Social Interactivity
Multi-Agent Systems

Motivation

- **So far:** Approximation of π^*, V^*, and Q^* using
  – Tables (e.g., Dynamic Programming, Q-Learning, SARSA, …)
  – Trees (e.g., MCTS)

- Works well for **small** and **discrete** problems, if sufficient memory and computational resources available



https://deepmind.com/blog/article/alphastar-mastering-real-time-strategy-game-starcraft-ii

- **Our goal is to solve large (and continuous) decision making problems!**

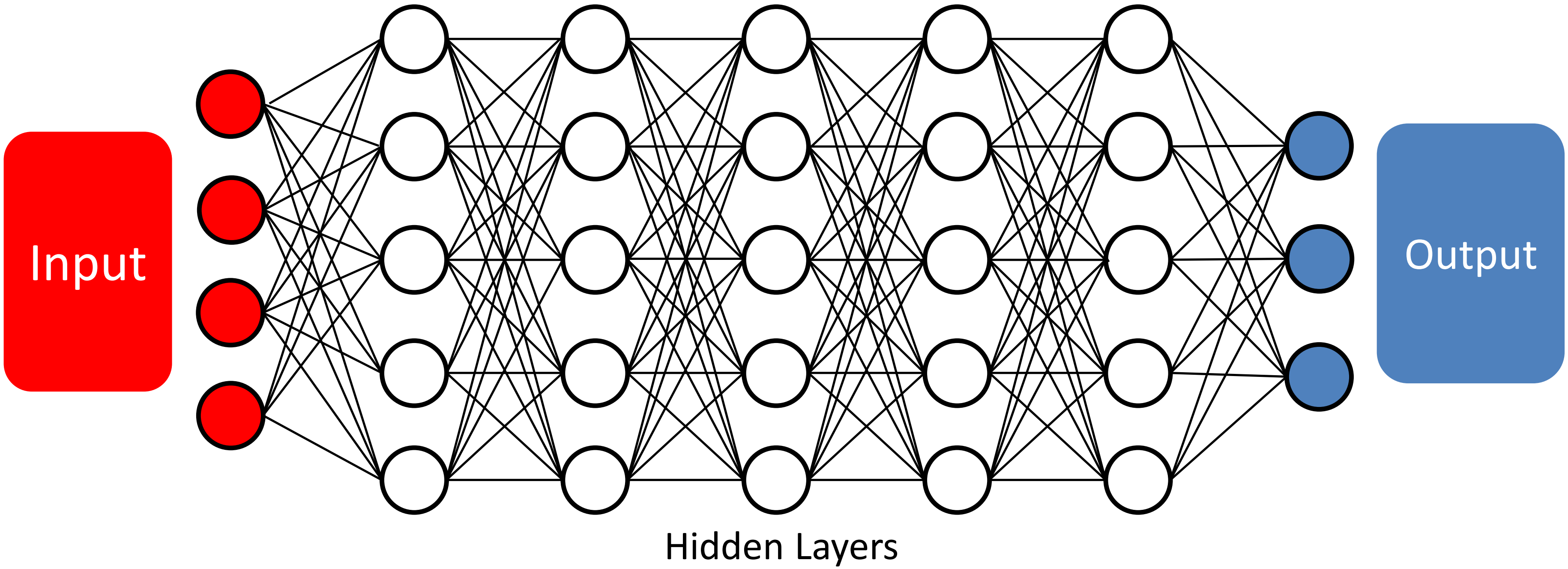
Motivation

- **Idea:** Use Function Approximation (Machine Learning) to approximate π^*, V^*, and Q^* using
  - **Gradient-based approximators (e.g., neural networks)**
  - Decision trees
  - Nearest neighbors
  - …

This is what we are focusing on …

Deep Learning

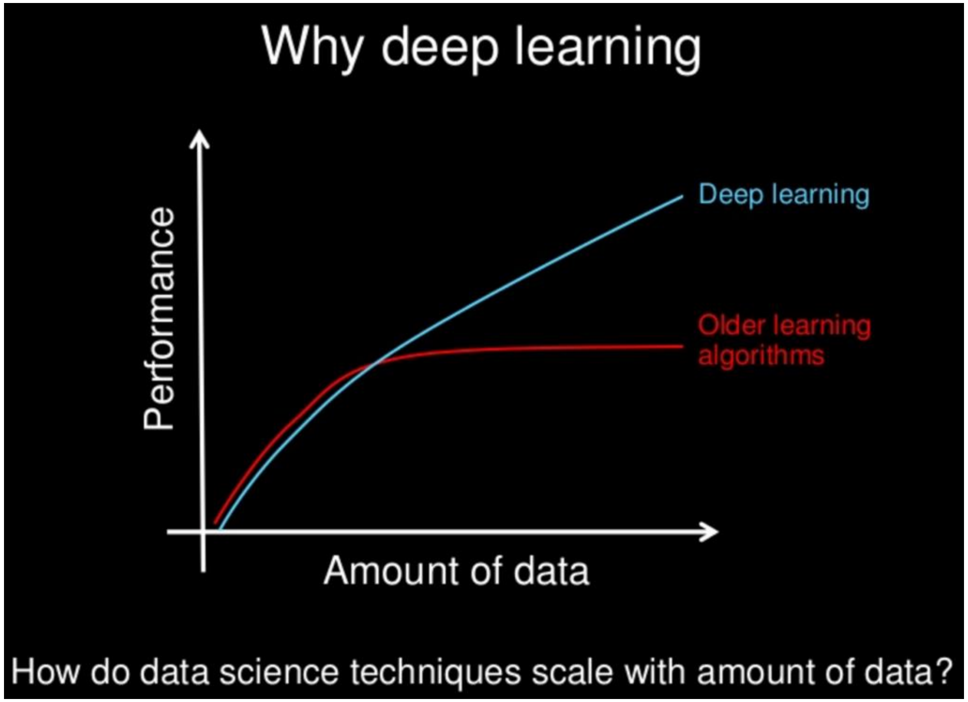
- **Deep Learning:** Neural Network with multiple hidden layers



- Enables **end-to-end learning** (feature learning + mapping) of tasks
- Typically trained with Stochastic Gradient Descent
- Works well for many complex tasks – but hard to interpret

Why Deep Learning for Reinforcement Learning?

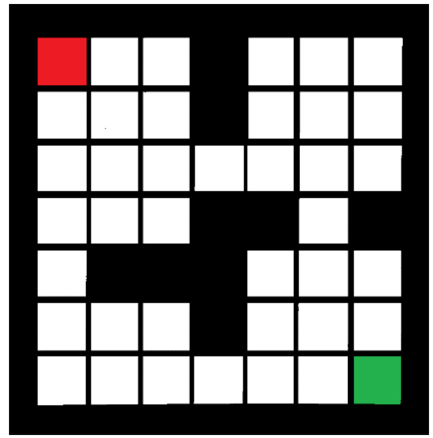
- Reinforcement Learning typically requires large amount of experience / data to solve complex problems (with large state and action spaces)
- Deep Learning scales well with large amount of data



Andrew Ng, What data scientistis should know about deep learning, 2015

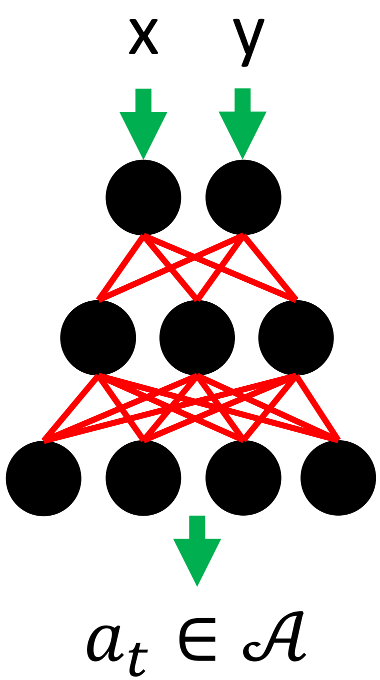
Motivation of Function Approximation

- Rooms Example:



40 States

(requires 40*4 = 160 table entries in theory)



- Approximator $\hat{f}_\theta$ (e.g., a neural network):
  - learns **unknown** function f (e.g., π^*, V^*, Q^*)
  - has parameters / weights $\theta \in \Theta$

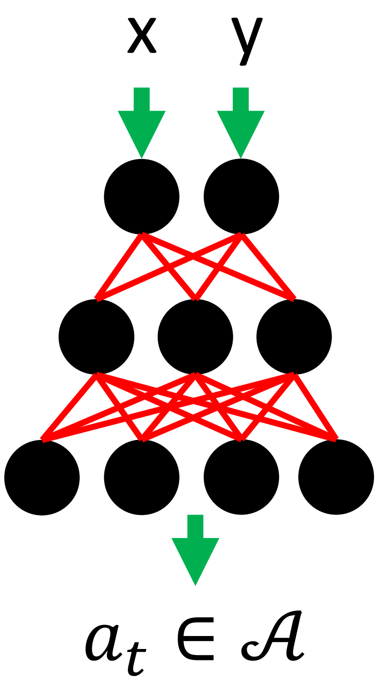
18 Weights (< 160)

Limitations of Function Approximation

- Modifying θ to update $\hat{f}_\theta(x)$ **will affect** $\hat{f}_\theta(x')$ even if x' is completely independent of x

- The more updates to $\hat{f}_\theta(x)$ the **better** the estimation will be (and the **worse** the estimation some other x' might become)

- Instead of directly seeking perfect approximations of π^*, V^*, and Q^*, we seek for appropriate weights θ, which minimize some **error / loss function**

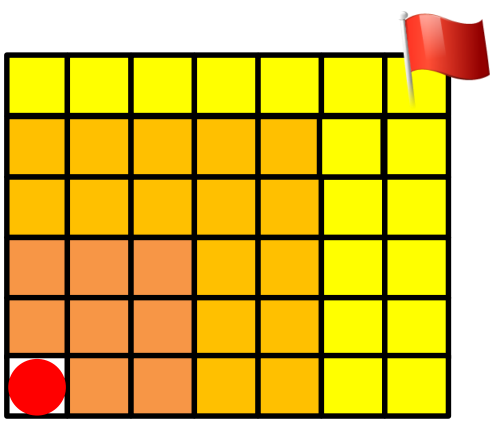


Our goal is a good **generalization** for the **most** „relevant“ inputs!

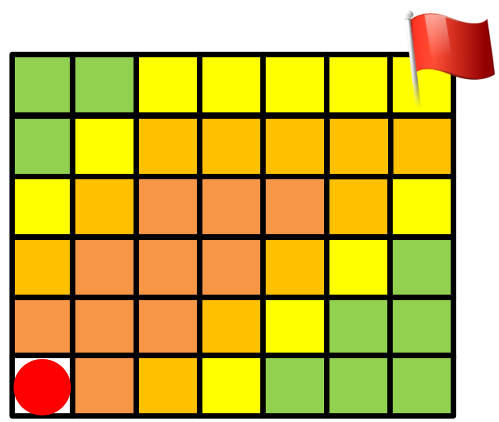
„Relevant“ Input and Non-Stationarity

- As our agents evolves, the experience it generates becomes non-stationary:
  - Current policy π^n improves over time
  - „Bad“ states are visited less over time

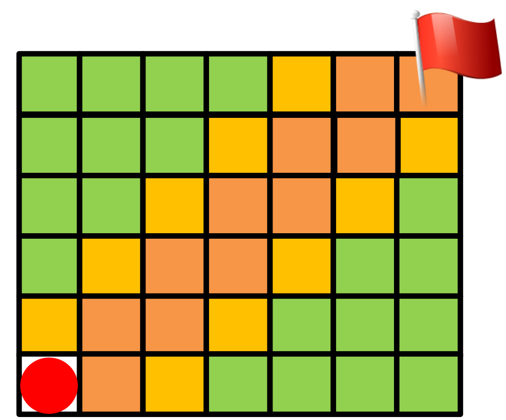
- **Example:**



initial exploration



focused exploration

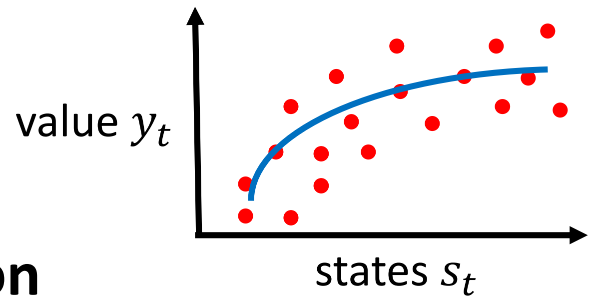


focused exploitation

→ Value Function Approximation

Value Function Approximation

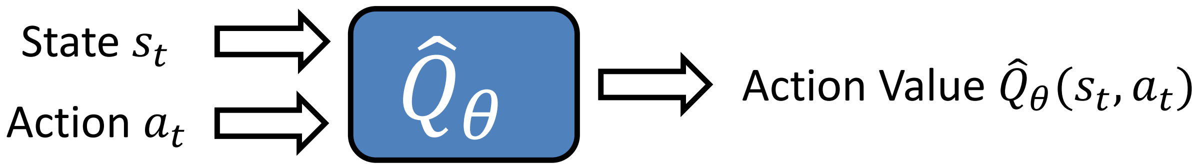
- **Goal:** approximate $Q(s_t, a_t)^*$ using $\hat{Q}_\theta(s_t, a_t)$
  - $\hat{Q}_\theta(s_t, a_t)$ is represented by a neural network
  - θ are the weights / learnable parameters



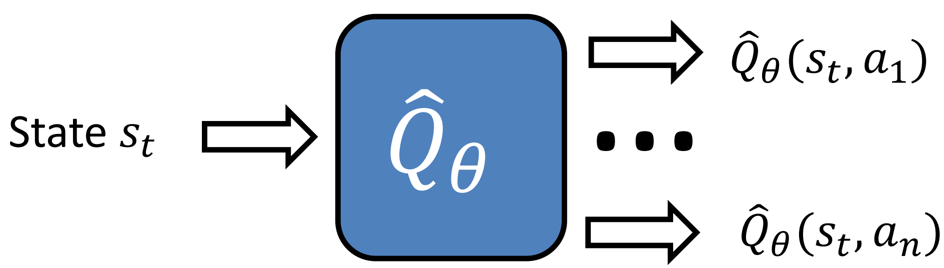
- $Q(s_t, a_t)^*$ is approximated via **regression**
  - Given experience samples $e_t = \langle s_t, a_t, r_t, s_{t+1} \rangle$
  - Regression target y_t is defined by Bellman equation or sample return G_t (or some combination)
  - θ is optimized via (stochastic) gradient descent on $\langle s_t, y_t \rangle$-pairs

Value Network Architectures

- Possible architectures:



Which one makes more sense to you?



Monte Carlo Approximation

- **Idea:** Learn $Q(s_t, a_t)^*$ from sample returns
  1) Run multiple episodes $s_1, a_1, r_1, \dots, s_{T-1}, a_{T-1}, r_{T-1}, s_T$
  2) For each episode compute the return G_t for each state s_t

$$G_t = \sum_{k=0}^{h-1} \gamma^k \mathcal{R}\big(s_{t+k}, \pi(s_{t+k})\big), \gamma \in [0,1]$$

  3) Perform regression on each $\langle s_t, G_t \rangle$-pair to adapt θ
     – Typically one gradient descent step (**why only one?**)
     – Minimize the mean squared error δ_t w.r.t. to θ

$$\delta_t = (G_t - \hat{Q}_\theta(s_t, a_t))^2$$

  4) Repeat all steps starting from 1.

- **How shall we „run" these episodes? → exploration?**

Exploration-Exploitation Dilemma

- **Goal:** To ensure good generalization of $\hat{Q}_\theta(s_t, a_t)$, we need to explore various states sufficiently
  - Otherwise overfitting on „well-known" states
  - Unexpected / Undesirable behaviour on „new" states
  - Detect / Adapt to changes in the environment

- **Approach:** Use multi-armed bandit based exploration
  - Example: ϵ-greedy ($\epsilon > 0$)

With probability $\begin{cases} \epsilon\text{, select randomly} \\ 1-\epsilon\text{, select action } a_t \text{ with highest } \hat{Q}_\theta(s_t, a_t) \end{cases}$

Monte Carlo Approximation Summary

- Advantages:
  - Simple method for value function approximation
  - Garantueed convergence given sufficient time and data

- Disadvantages:
  - Only offline learning (task needs to be episodic)
  - High variance in return estimation

Temporal Difference (TD) Learning

- **Idea:** Learn $Q(s_t, a_t)^*$ from Bellman Updates
  1) Run multiple episodes $s_1, a_1, r_1, \ldots, s_{T-1}, a_{T-1}, r_{T-1}, s_T$
  2) For each episode compute the TD target $\hat{G}_t$ for each state

$$\hat{G}_t = r_t + \gamma \max_{a_{t+1} \in \mathcal{A}} \hat{Q}_\theta(s_{t+1}, a_{t+1})$$

  3) Perform regression on each $\langle s_t, G_t \rangle$-pair to adapt θ
     – Typically one gradient descent step
     – Minimize the mean squared error δ_t w.r.t. to θ

$$\delta_t = (\hat{G}_t - \hat{Q}_\theta(s_t, a_t))^2$$

  4) Repeat all steps starting from 1.

- **Similarly to Monte Carlo Approximation, we need sufficient exploration (e.g., ϵ-greedy)**

TD Learning Summary

- Advantages:
  - Can be applied online $\hat{G}_t = r_t + \gamma \max_{a_{t+1} \in \mathcal{A}} \hat{Q}_\theta(s_{t+1}, a_{t+1})$
  - Reuse / Bootstrapping of successor values

- Disadvantages:
  - No convergence garantuees (except linear function approximation)
  - High bias in return estimation

TD Learning Issues with Deep Learning

- Deep Learning is highly sensitive to correlation in data
  - Possible overfitting / Hard generalization
  - Experience / Data generated via RL is highly correlated
    (1) w.r.t. successing states within the same episode
    (2) w.r.t. action value prediction $\hat{Q}$ and the TD target:

$$\hat{G}_t = r_t + \gamma \max_{a_{t+1} \in \mathcal{A}} \hat{Q}_\theta(s_{t+1}, a_{t+1})$$

TD Learning Issue: Correlation of Successing States

- Successing States can be highly correlated due to
  - Similarity (small differences in input data)
  - Policy (only certain policies lead to certain trajectories)

- **Naive fitting / regression leads to Overfitting!**

- **Solution:** Experience Replay
  - Sample small subset (**minibatch**) of experience buffer
  - Calculate TD-Targets/-Losses of minibatch
  - Perform regression on sampled minibatch
  - **Only possible for Off-Policy Reinforcement Learning!**

TD Learning Issue: Correlation of Prediction and Target

- Prediction of $\hat{Q}_\theta(s_t, a_t)$ is highly correlated to $\hat{Q}_\theta(s_{t+1}, a_{t+1})$:

$$\boldsymbol{\hat{Q}_\theta(s_t, a_t)} \approx r_t + \gamma \max_{a_{t+1} \in \mathcal{A}} \boldsymbol{\hat{Q}_\theta(s_{t+1}, a_{t+1})}$$

- Small changes to θ may result in huge changes of $\hat{Q}_\theta(s_t, a_t)$ **and** $\hat{Q}_\theta(s_{t+1}, a_{t+1})$

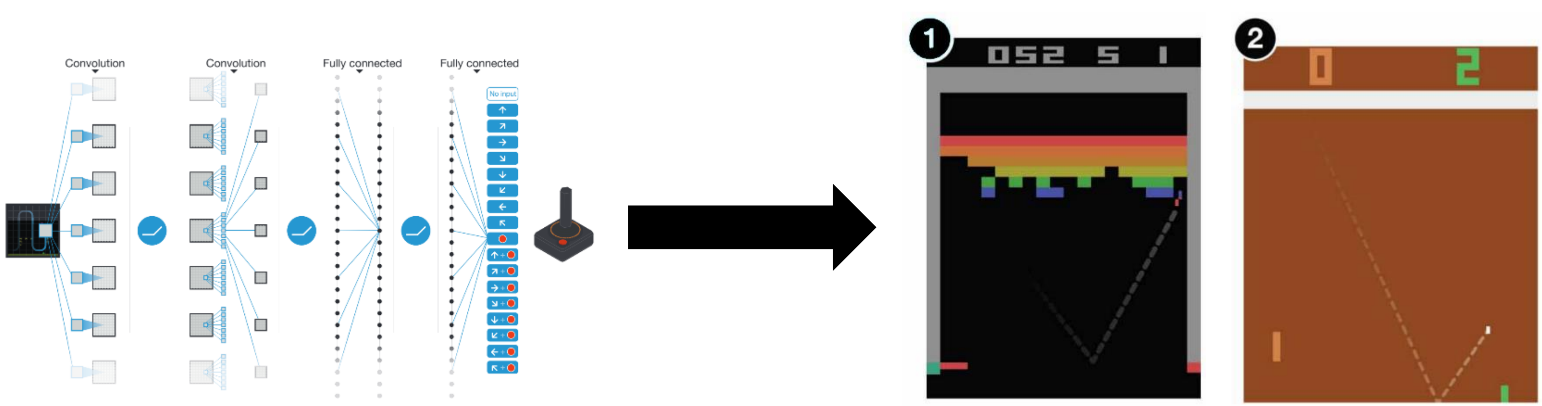
- **Solution:** Target Network
  - Use an older copy θ^- of θ to compute the TD target

$$\hat{G}_t = r_t + \gamma \max_{a_{t+1} \in \mathcal{A}} \boldsymbol{\hat{Q}_{\theta^-}(s_{t+1}, a_{t+1})}$$

  - Periodically set θ^- to θ (θ^- is freezed, while θ is adapting)
  - Possible extension: soft updates of θ^- using a weighting factor α ($\theta^- \leftarrow (1-\alpha)\theta^- + \alpha\theta$)

DQN – Value-based Deep Reinforcement Learning

- Deep Q-Networks (DQN):
  - Q-Learning implemented with deep neural networks
  - Uses **experience replay** and **target networks**
  - Successfully applied to multiple Atari Games using end-to-end learning (no handcrafted features for state descriptions)



…

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

Value Function Approximation Summary

- Monte Carlo Approximation (offline, high variance)
- Temporal Difference Learning (online, high bias)
- Deep RL suited for high-dimensional state spaces
- Action space must be discrete for model-free control

→ **Policy Approximation**

Policy Approximation

- **Goal:** approximate π^* using $\hat{\pi}_\theta(a_t|s_t) \in [0,1]$
  - $\hat{\pi}_\theta(a_t|s_t)$ is represented by a neural network
  - θ are the weights / learnable parameters

- Why approximating π^* instead of Q^*?
  - Stochastic policies
  - Continuous action spaces
  - Convergence properties

Policy Gradients

- **Goal:** approximate π^* using $\hat{\pi}_\theta(a_t|s_t) \in [0,1]$
  - $\hat{\pi}_\theta(a_t|s_t)$ is represented by a neural network
  - θ are the weights / learnable parameters

- In episodic tasks, $\hat{\pi}_\theta(a_t|s_t)$ is evaluated with its **start value** $J(\theta)$:

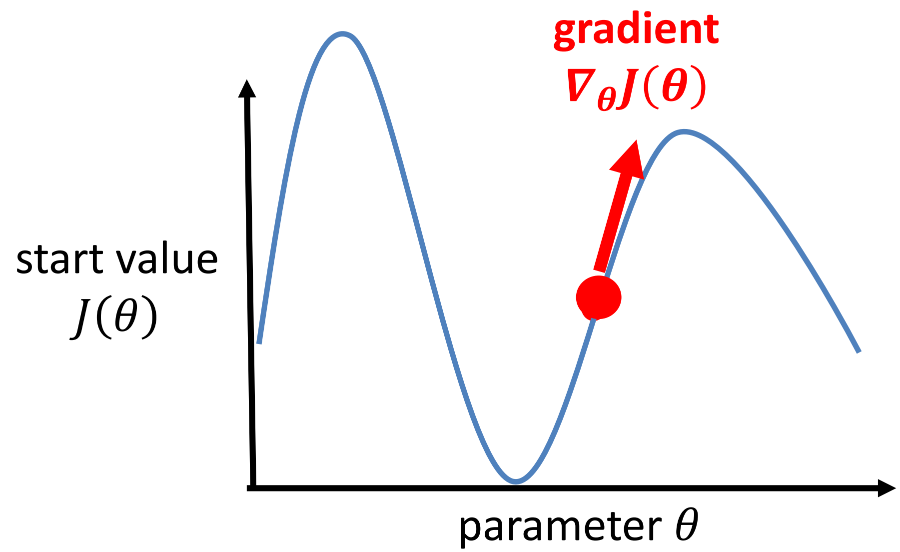
$$J(\theta) = V^{\pi_\theta}(s_1) = \mathbb{E}[G_1|s_1, \pi_\theta]$$

- To learn π^*, we have to optimize θ to **maximize** $J(\theta)$
  - E.g., with gradient ascent

Policy Gradients

- To perform gradient ascent w.r.t. θ, we have to estimate the gradient of $J(\theta)$:

$$\nabla_\theta J(\theta) = \begin{bmatrix} \frac{\delta J(\theta)}{\delta \theta_1} \\ \dots \\ \frac{\delta J(\theta)}{\delta \theta_n} \end{bmatrix}$$



- Given a differentiable policy $\hat{\pi}_\theta(a_t|s_t)$, the gradient $\nabla_\theta J(\theta)$ can be estimated with:

$$A^{\pi_\theta}(s_t, a_t)\nabla_\theta \log \hat{\pi}_\theta(a_t|s_t)$$

Policy Gradients

- The policy gradient

$$A^{\pi_\theta}(s_t, a_t)\nabla_\theta \log \hat{\pi}_\theta(a_t|s_t)$$

with advantage $A^{\pi_\theta}(s_t, a_t)$ can be expressed in different ways:

- $A^{\pi_\theta}(s_t, a_t) = G_t = \sum_{k=0}^{h-1} \gamma^k r_t$ (REINFORCE)
- $A^{\pi_\theta}(s_t, a_t) = G_t - \hat{V}_\omega(s_t)$ (Advantage Actor-Critic)
- $A^{\pi_\theta}(s_t, a_t) = \hat{Q}_\omega(s_t, a_t)$ (Q Actor-Critic)
- $A^{\pi_\theta}(s_t, a_t) = r_t + \gamma \hat{V}_\omega(s_{t+1}) - \hat{V}_\omega(s_t)$ (TD Actor-Critic)
- $A^{\pi_\theta}(s_t, a_t) = \sum_{k=0}^{n-1} \gamma^k r_t + \gamma^n \hat{V}_\omega(s_{t+n})$ (n-step Actor-Critic)

Implementation Details

- Variant 1:
  - Modify classification loss $\mathcal{L}_{ce}$ (e.g., cross entropy loss)
  - Given episodes with experience samples $e_t = \langle s_t, a_t, r_t, s_{t+1} \rangle$
  - Compute $\mathcal{L}_{ce}(s_t) = a_t \log \hat{\pi}_\theta(a_t|s_t)$ for each e_t
  - Multiply loss $\mathcal{L}_{ce}(s_t)$ with $\boldsymbol{A^{\pi_\theta}(s_t, a_t)}$ (see slide before)
  - Minimize $\mathcal{L}_{ce}$ = $\mathbb{E}[\mathcal{L}_{ce}(s_t)\boldsymbol{A^{\pi_\theta}(s_t, a_t)}]$ with gradient descent

- **Important Note:** No experience replay used here! Gradient descent has to be performed on all experience samples (which are discarded afterwards) ➡ Can you guess why?

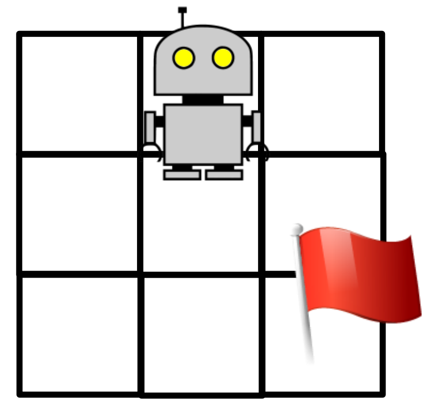
Implementation Details

- Variant 2:
  - Modify gradient of classification loss $\mathcal{L}_{ce}$
  - Given episodes with experience samples $e_t = \langle s_t, a_t, r_t, s_{t+1} \rangle$
  - Compute $\mathcal{L}_{ce}(s_t) = a_t \log \hat{\pi}_\theta(a_t|s_t)$ for each e_t
  - Compute gradients $\nabla_\theta \log \hat{\pi}_\theta(a_t|s_t)$
  - Multiply gradients $\nabla_\theta \log \hat{\pi}_\theta(a_t|s_t)$ with $\boldsymbol{A^{\pi_\theta}(s_t, a_t)}$
  - Apply accumulated gradients to $\theta \leftarrow \theta + \alpha \sum_{e_t} \boldsymbol{A^{\pi_\theta}(s_t, a_t)} \nabla_\theta \log \hat{\pi}_\theta\, (a_t|s_t)$

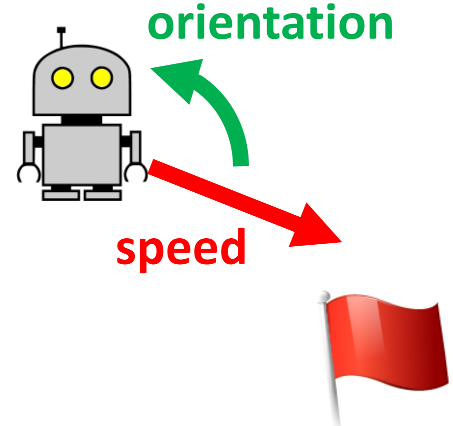
- **Important Note:** No experience replay used here! Gradient descent has to be performed on <u>all experience samples</u> (which are discarded afterwards) ➡ Can you guess why?

Discrete vs. Continuous Action Spaces

- Discrete Action Space example:
  - Enumerable actions (e.g., move north, south, west, east)
  - Actions representable as **Integer**

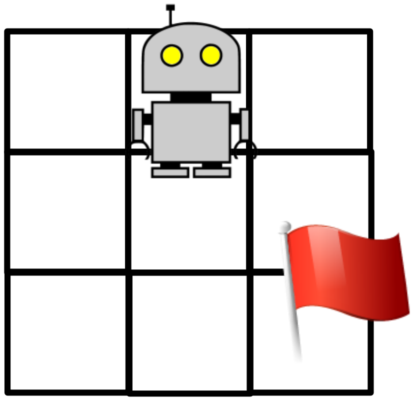


- Continuous Action Space example:
  - Multiple degrees of freedom (e.g., position, orientation, speed)
  - Actions representable as **Vector of Real Values** (<u>Infinite</u> Action Space!)

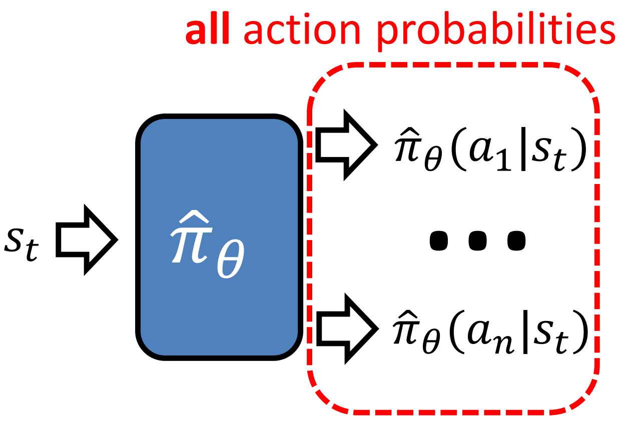


Discrete vs. Continuous Action Spaces

- Discrete Action Space example:
  - Enumerable actions (e.g., move north, south, west, east)
  - Actions representable as **Integer**

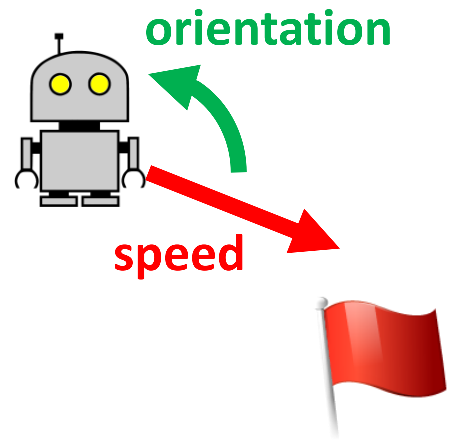


- Discrete Policy Approximation:
  - Output is a probability vector (e.g., softmax)
  - **Each vector element** $\hat{\pi}_\theta(a_t|s_t)$ corresponds to the probability of a **single action** $\boldsymbol{a_t}$
  - Implicit exploration: sample $a_t \sim \hat{\pi}_\theta(a_t|s_t)$

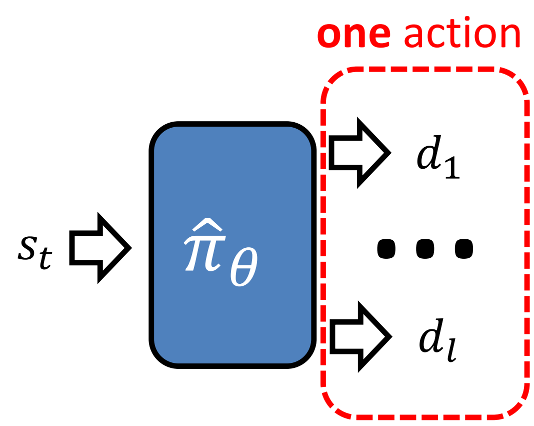


Discrete vs. Continuous Action Spaces

- Continuous Action Space example:
  - Multiple degrees of freedom (e.g., position, orientation, speed)
  - Actions representable as **Vector of Real Values** (Infinite Action Space!)

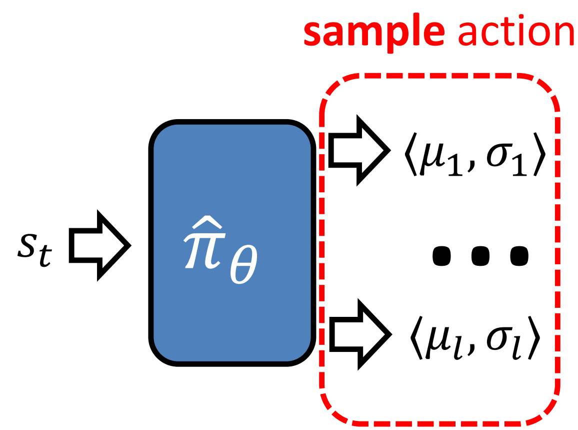


- Continuous action spaces:
  - Output is a vector of real values (which could be bounded)
  - Each vector element corresponds to a degree of freedom d_i (e.g., acceleration, rotation, …)
  - The **whole vector** represents a **single action a_t**
  - Needs additional exploration mechanism

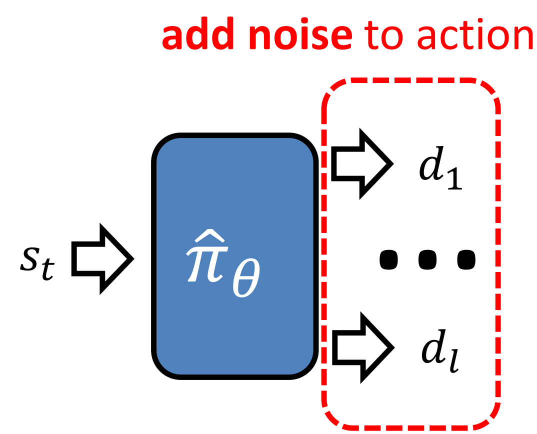


Exploration in Continuous Action Spaces

- On-Policy:
  - Learn distribution (e.g., normal distribution $\mathcal{N}_{\mu_i,\sigma_i}(s_t)$) for each degree of freedom d_i
  - Sample action $a_t = \langle d_1 \sim \mathcal{N}_{\mu_1,\sigma_1}(s_t), \dots, d_l \sim \mathcal{N}_{\mu_l,\sigma_l}(s_t) \rangle$
  - Example algorithm: A2C, A3C

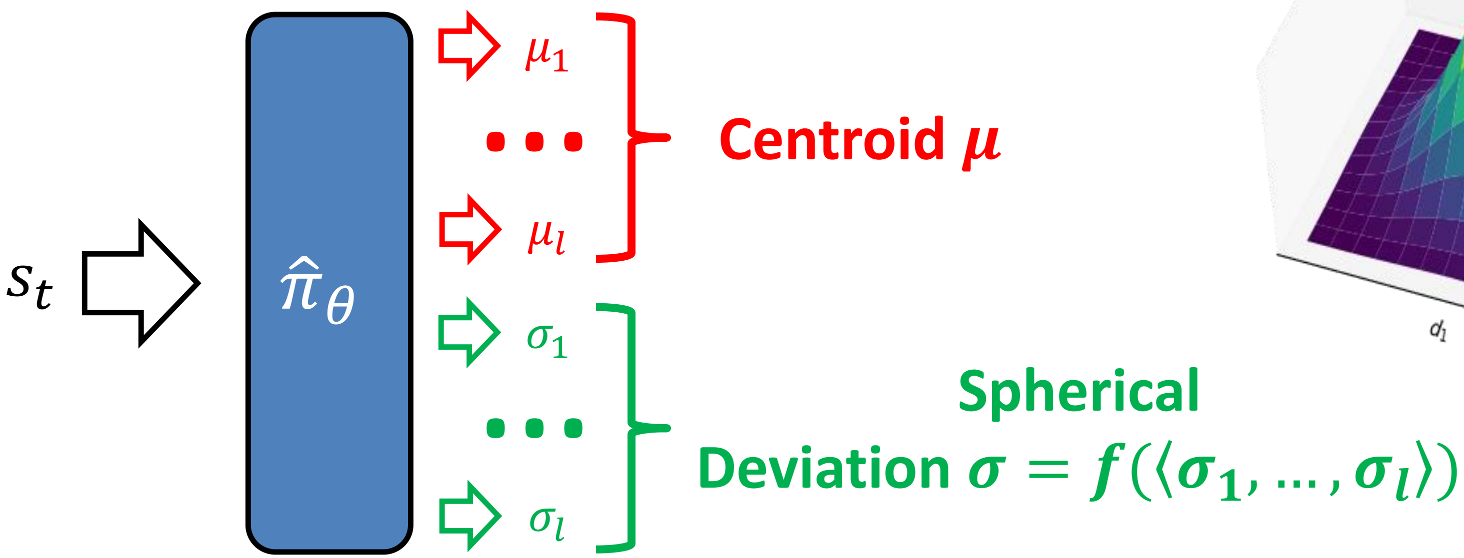


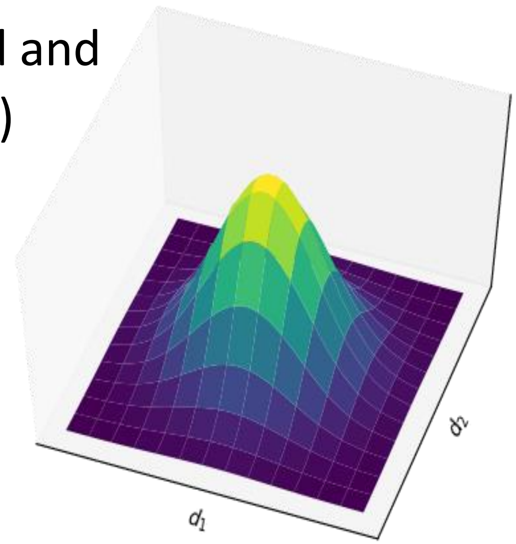
- Off-Policy:
  - Add noise to the degrees of freedom (e.g., by using an external normal distribution)
  - Requires further adjustments (most policy gradient algorithms are on-policy!)
  - Example algorithms: DDPG, PPO



On-Policy Exploration in Continuous Action Spaces

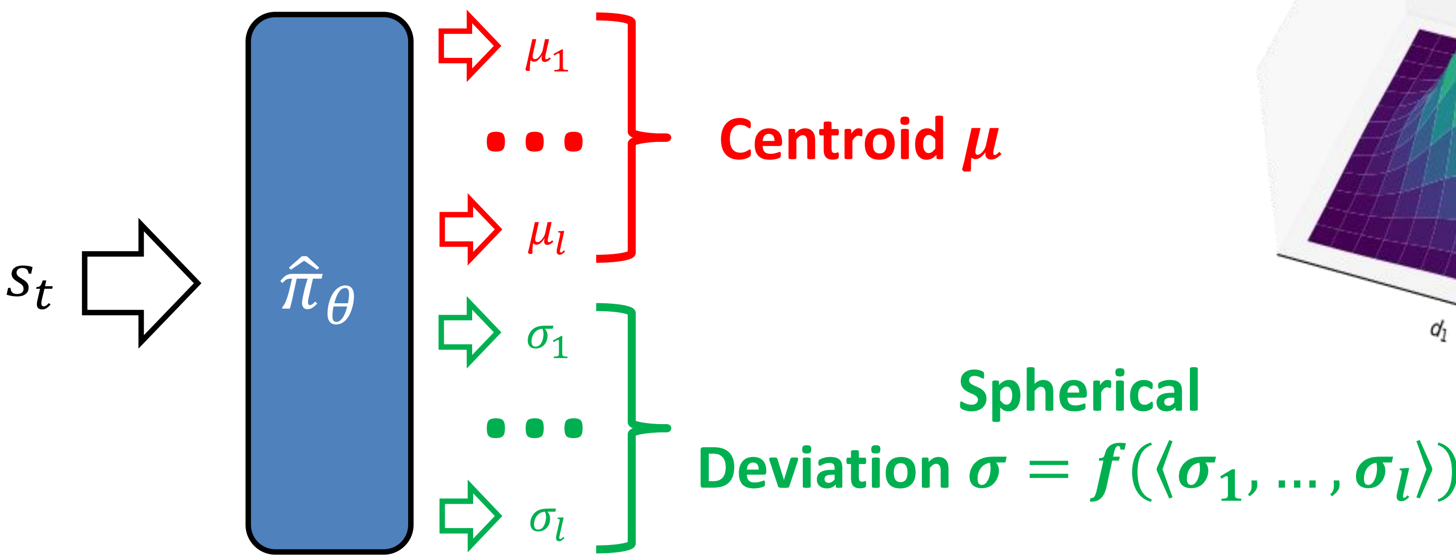
- **Approach:** Learn normal distribution $\mathcal{N}_{\mu_i,\sigma}(s_t)$ for each degree of freedom d_i using two output layers:
  - One layer $\mu = \langle \mu_1, \dots, \mu_l \rangle$
  - The other layer $\sigma = f(\langle \sigma_1, \dots, \sigma_l \rangle)$
  - After sampling a_t, losses $\mathcal{L}_\mu$ and $\mathcal{L}_\sigma$ are computed and multiplied with $A^{\pi_\theta}(s_t, a_t)$ (for joint minimization)

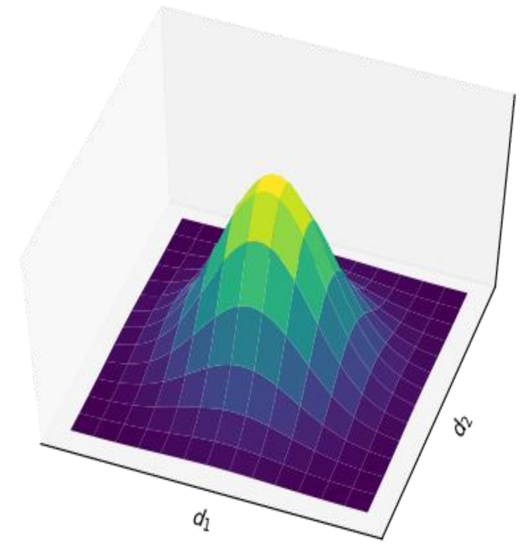




On-Policy Exploration in Continuous Action Spaces

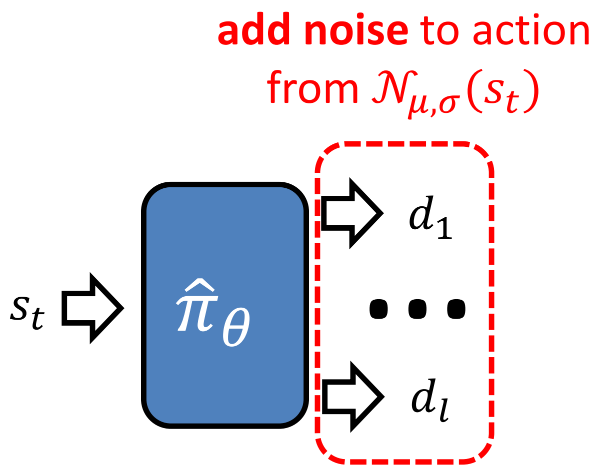
- **Example:** Assume μ is linear and f for σ is softplus.
  - Given an action a_t for state s_t and experience tuple $e_t = \langle s_t, a_t, r_t, s_{t+1} \rangle$
  - Use the mean squared loss between a_t and μ as $\mathcal{L}_\mu$
  - Use softplus as $\mathcal{L}_\sigma$
  - Minimize $\mathbb{E}[\big(\mathcal{L}_\mu(s_t) + \mathcal{L}_\sigma(s_t)\big) A^{\pi_\theta}(s_t, a_t)]$





Off-Policy Exploration in Continuous Action Spaces

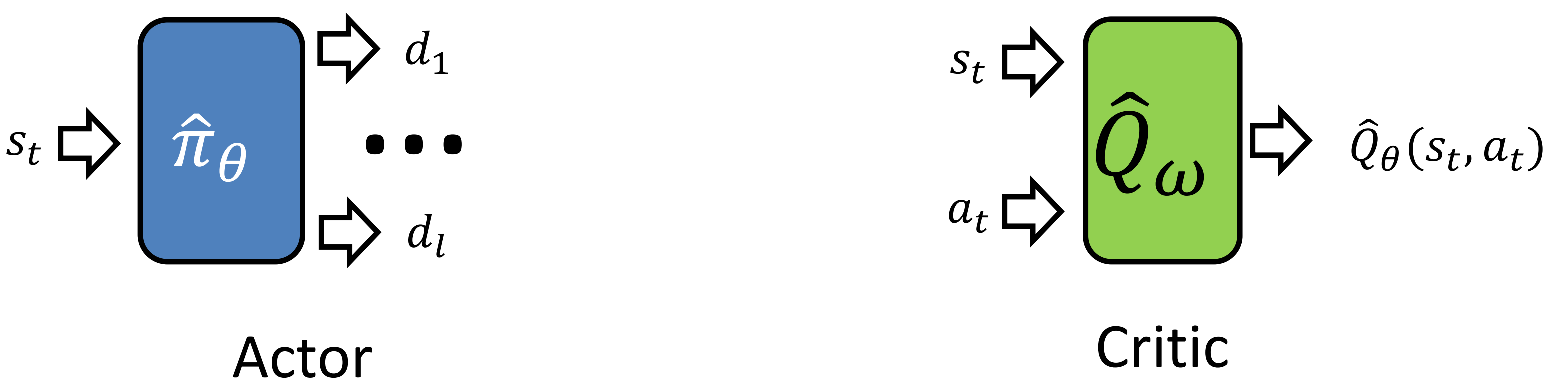
- **Approach:** Add noise from an <u>external</u> origin-centered (normal) distribution $\mathcal{N}_{\mu,\sigma}(s_t)$ to each degree of freedom d_i
  - σ can be adjusted to control the noise level (degree of exploration)
  - Alternatively: add noise to weights θ of approximator $\boldsymbol{\hat{\pi}_\theta}$
  - Note: $\boldsymbol{\hat{\pi}_\theta}$ has to be updated in an **off-policy fashion**!



Off-Policy Exploration in Continuous Action Spaces

- **Example:** Deep Deterministic Policy Gradient (DDPG)
  – Approximates $\hat{\pi}_\theta$ as **actor** and $\hat{Q}_\omega$ as **critic**
  – Given a buffer of experience samples $e_t = \langle s_t, a_t, r_t, s_{t+1} \rangle$
  – Update $\hat{Q}_\omega$ (e.g., using TD-learning)
  – Update $\hat{\pi}_\theta$ with previously computed gradients of $\hat{Q}_\omega$ by minimizing:

$$\mathbb{E}[\nabla_{a_t} \hat{Q}_\omega(s_t, a_t)\ \nabla_\theta\ \hat{\pi}_\theta(s_t)]$$



Policy Approximation Summary

- Direct approximation of π^*
- Advantage function can be approximated in various ways
- Learning of stochastic policies possible
- Applicable to continuous action spaces (requires additional mechanisms for exploration)
- Guarantueed convergence to local optimum

→ **Overview**

Function Approximation Overview

- Value Function Approximation

	Monte Carlo	Temporal Difference
Bias	Low	High
Variance	High	Low
Convergence	Guarantueed	Not guarantueed
On-/Off-Policy	Both	Both

- Policy Approximation

	REINFORCE/AC	DDPG
Bias	Depends on A^{π_θ}	Depends on A^{π_θ}
Variance	Depends on A^{π_θ}	Depends on A^{π_θ}
Convergence	Guarantueed	Guarantueed
On-/Off-Policy	On-Policy	Off-Policy

Thank you!