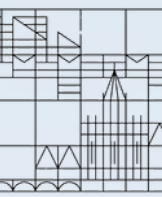


12 | Reinforcement Learning

Giordano De Marzo

<https://giordano-demarzo.github.io/>

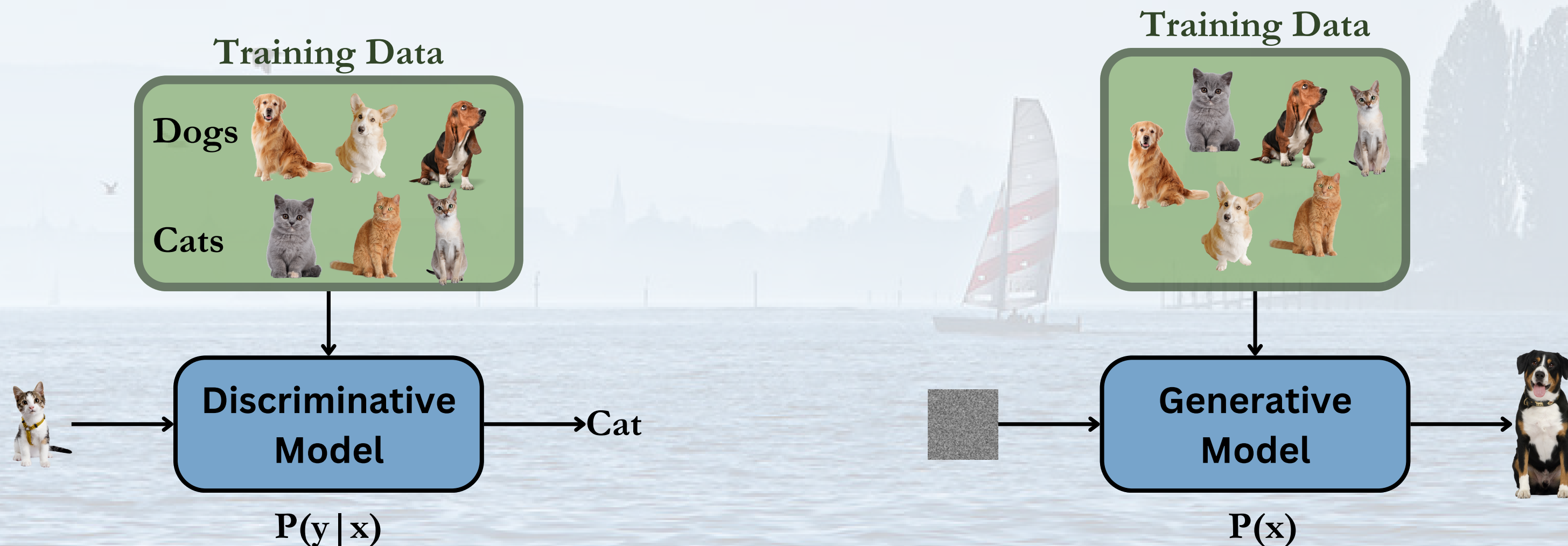
Deep Learning for the Social Sciences

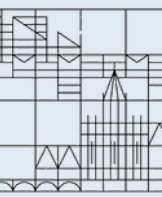


Discriminative vs Generative

Discriminative models assign labels to data: they reconstruct the probability $P(y | x)$ of the label (y) given the data (x)

Generative models create new data: they reconstruct the probability $P(x)$ of the data in our sample

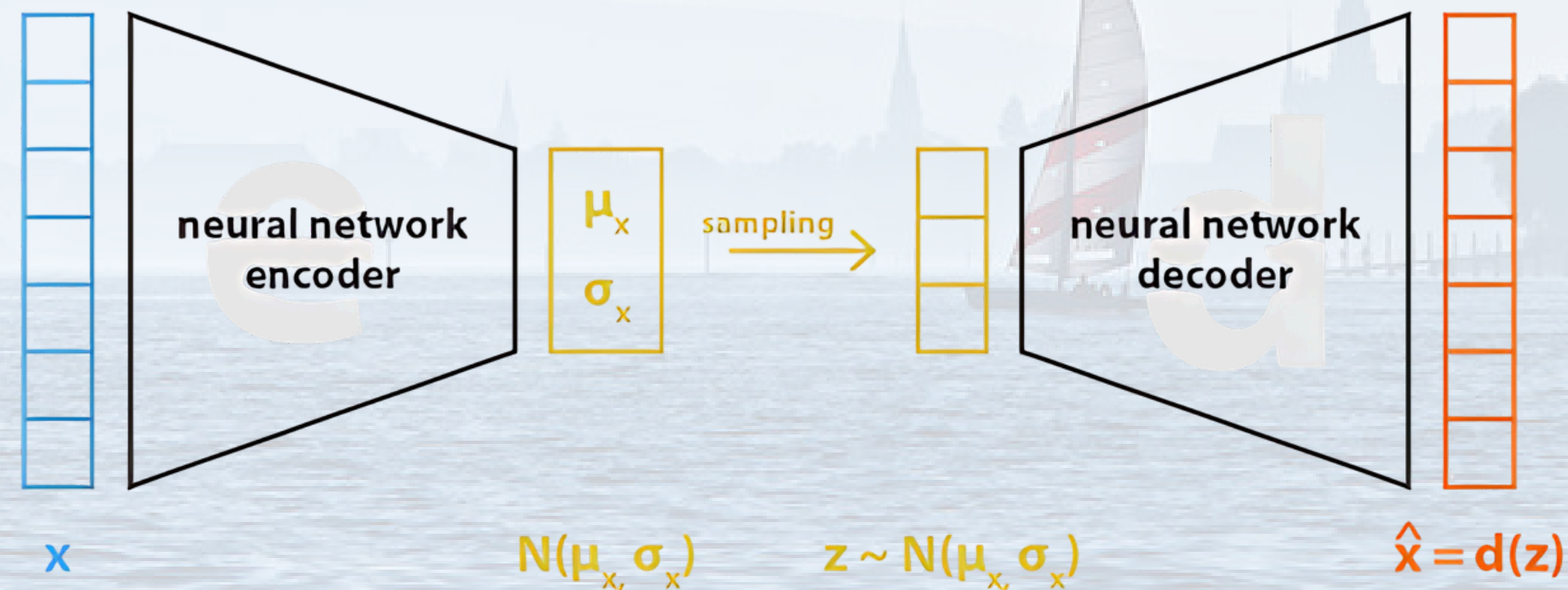


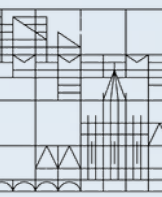


The Variational Autoencoder

The Variational Autoencoder (VAE) adds a twist of stochasticity to the standard AE

- same architecture, but now the encoder output consists of a mean vector and a variance vector
- the value of the latent variable z is computed sampling from a multivariate gaussian whose parameters are defined by the encoder's output
- the latent variable z is feed into the decoder, whose output is trained to be equal to the input

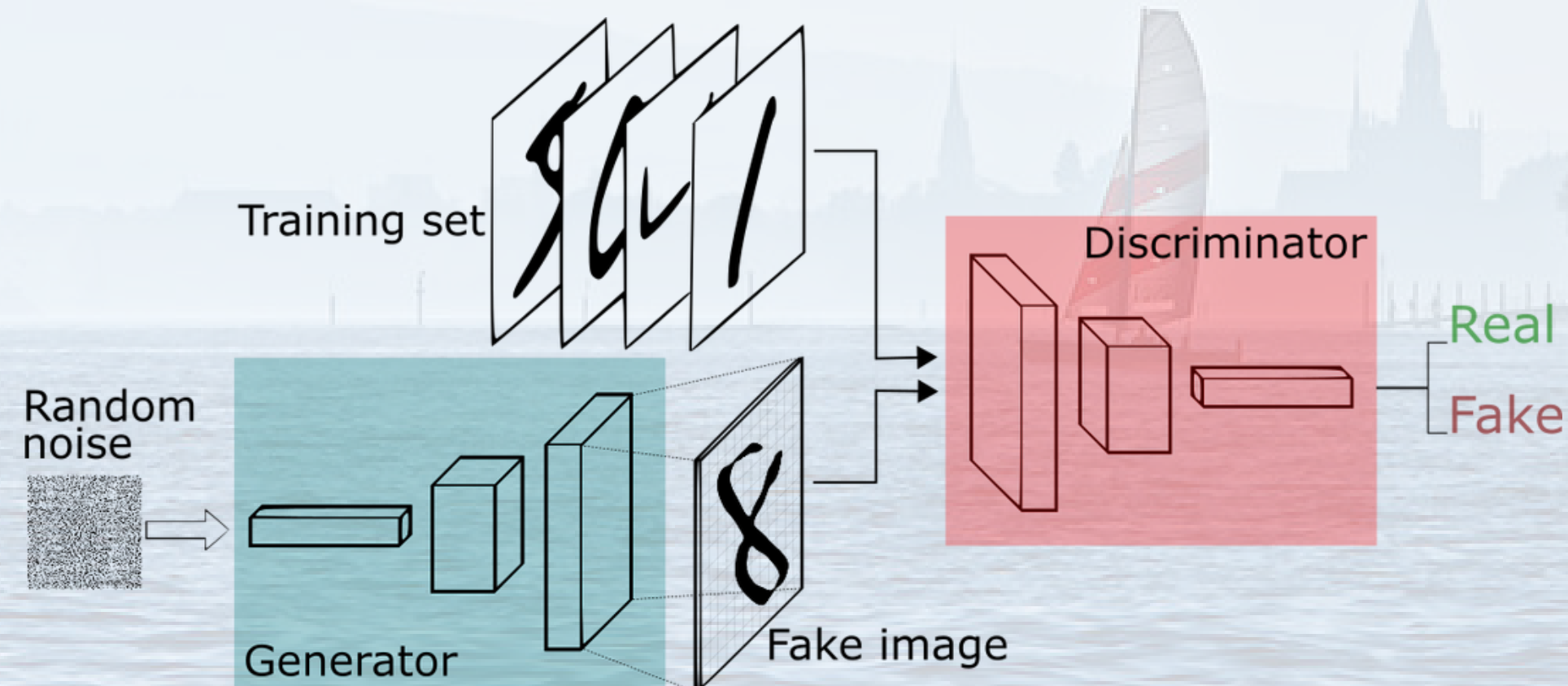


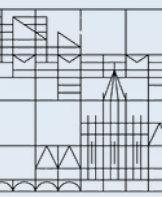


Generative Adversarial Networks

Generative Adversarial Networks (GANs) are much more powerful when it comes to generate realistic images. They are conceptually very similar and are composed of two models

- a generator that is trained to generate artificial data
- a discriminator that is trained to distinguish artificial data from real data
- note that we do not need labelled data (we automatically know what is real and what is fake)

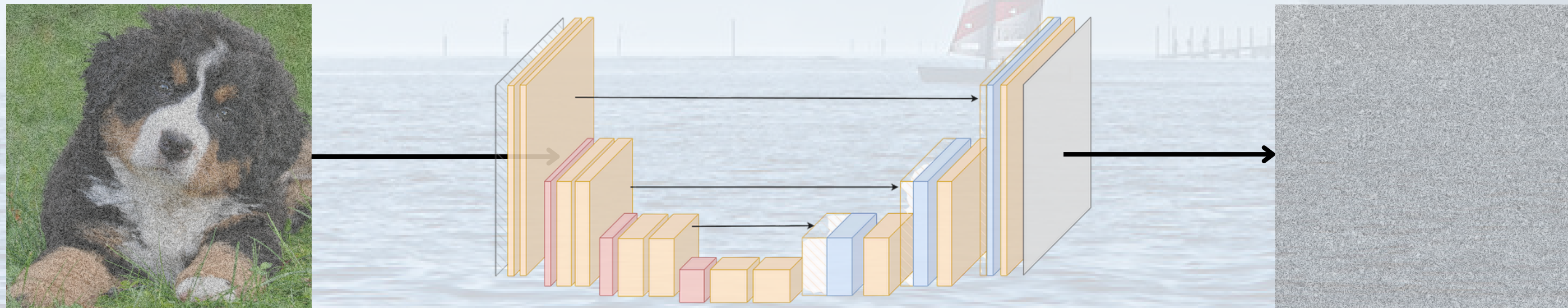


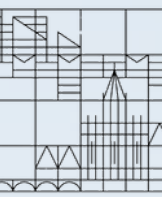


Reverse Process

The reverse process in diffusion models focuses on denoising the image step-by-step, transforming the noise back into a clear image

- The core idea is to train a neural network to predict the noise added at each step in the forward process. By accurately predicting this noise, the network can progressively reduce it.
- A specific CNN architecture called UNet is commonly used for this task. UNet consists of an encoder-decoder structure with skip connections
- The UNet model is applied iteratively, each time reducing a small amount of noise, gradually reconstructing the image from the noisy version.



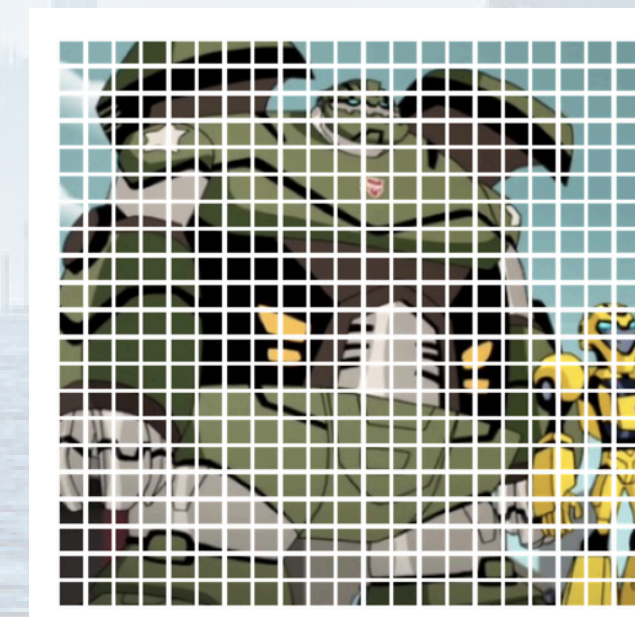
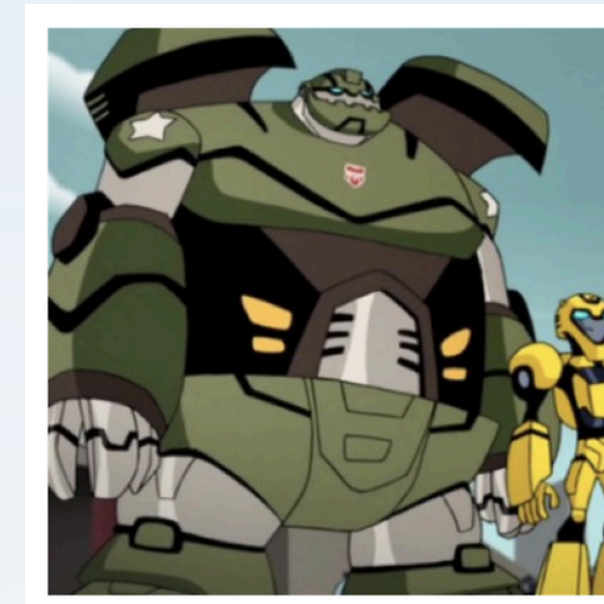


Vision Transformers: Images as Tokens

Before we can combine images and text, we need a way to feed images into a transformer

- An image is split into a grid of small patches
- Each patch is flattened and linearly projected into an embedding vector
- Positional information is added, since patches lose their spatial layout once flattened
- The resulting sequence of patch embeddings is processed by a standard transformer encoder

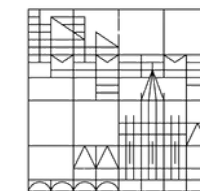
Once an image is a sequence of tokens, every trick we use for text becomes available for images





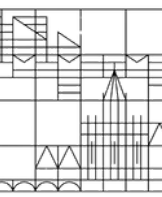
Outline

1. A New Learning Paradigm
2. Reinforcement Learning
3. Q-Learning and Policy Gradient
4. Applications



A New Learning Paradigm





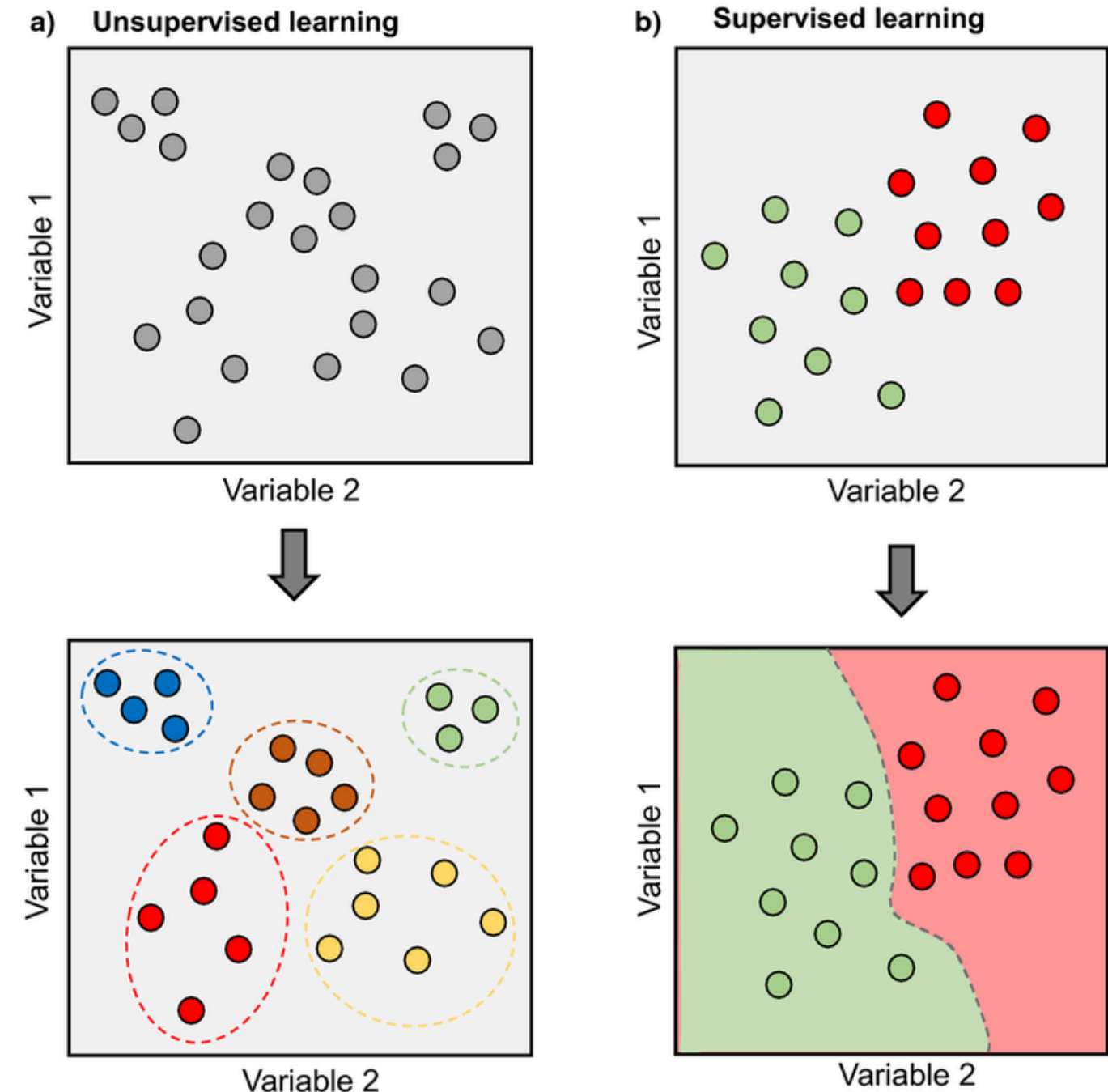
Supervised vs Unsupervised Learning

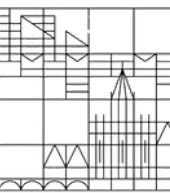
Up to now we mostly consider supervised deep learning models:

- MLP for classification and regression
- CNN for classification
- GNN for classification

We only considered one architecture that performs unsupervised learning, the Autoencoder.

We also saw how LLMs can be trained in a self-supervised way without requiring labelled datasets



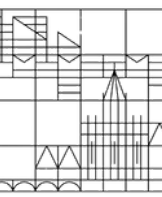


Limits of (Un)Supervised Learning

When we introduced Generative Deep Learning we discussed some of the limits of traditional learning paradigms

- supervised learning need a lot of labelled data
- with unsupervised learning we can't train the model to perform a specific task
- both frameworks are based on offline training, they struggle with new data or scenarios
- supervised learning rely on humans, how can we train a model better than humans?

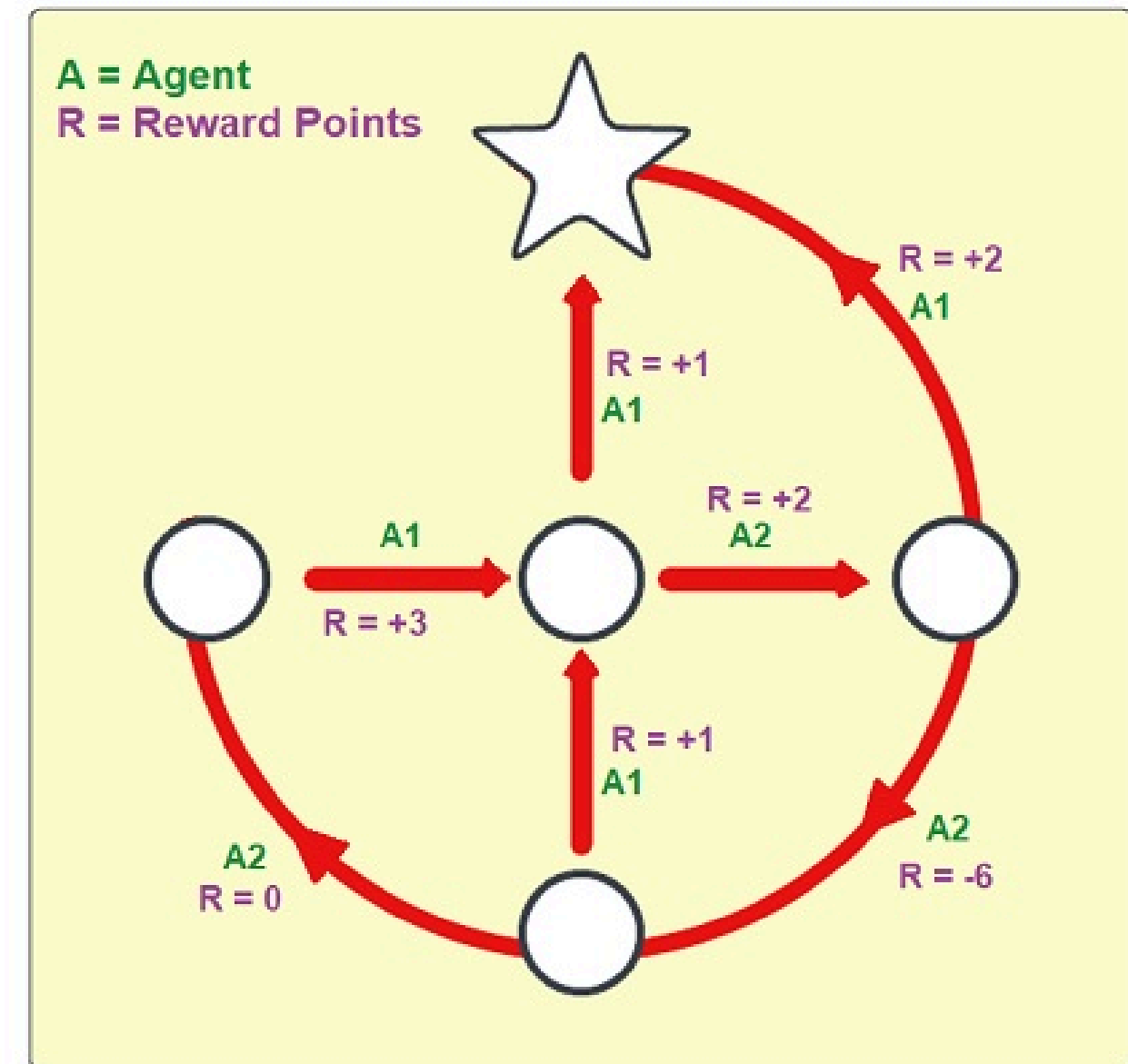
All these problems are solved by Reinforcement Learning (RL). It combines the benefits of the two approaches.



Reinforcement Learning

Reinforcement Learning (RL) is in-between supervised and unsupervised learning:

- The agent begins with little to no knowledge of the environment
- Agents learn to make decisions by interacting with a dynamic environment
- The goal is learning a strategy to perform actions that result in the highest long-term rewards
- Humans only need to set the environment and the rewards, then the agent learns autonomously





Strengths of Reinforcement Learning

Beyond Supervision

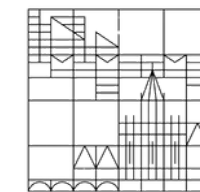
RL learns optimal behaviors through trial and error directly from raw experience without needing pre-labeled data. This capability is critical in environments where prior knowledge is limited or the solution space is complex.

Decision Making

RL excels in scenarios requiring a series of decisions, where each choice impacts future outcomes, making it ideal for strategic planning and operations management.

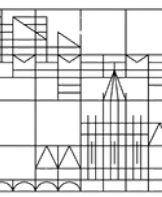
Adaptability

An RL agent can adapt to changes in an unpredictable environment in real-time, continually updating its strategy based on new experiences. This adaptability is essential for applications such as dynamic pricing or autonomous driving.



Reinforcement Learning



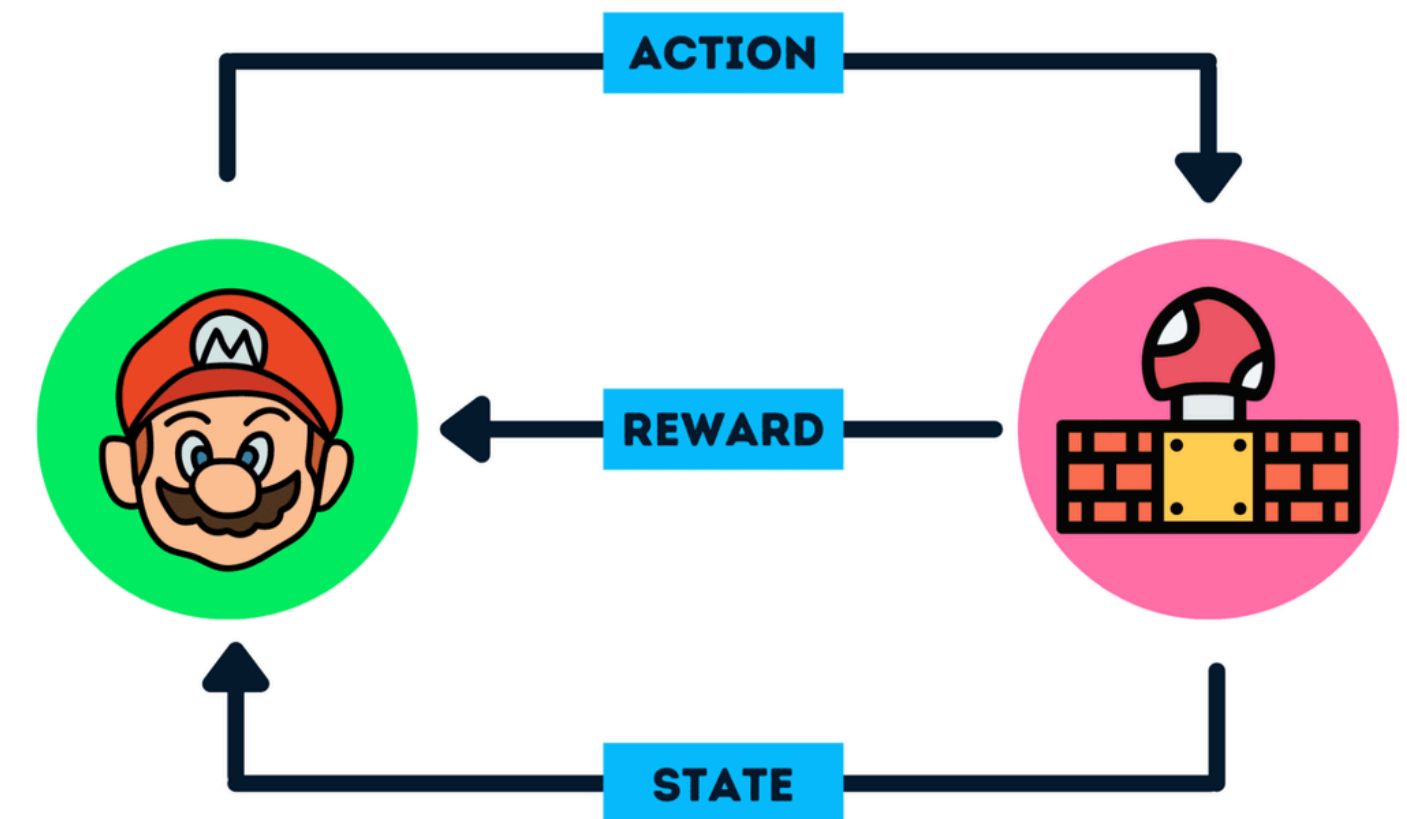


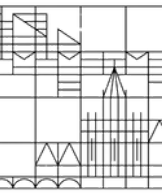
Reinforcement Learning Framework

The RL framework is structured around a set of key elements:

- **State:** The representation of the environment at any given time.
- **Action:** A set of possible moves the agent can make.
- **Reward:** Immediate feedback given to the agent to evaluate its last action.
- **Policy:** A strategy that the agent follows to determine its action at each state.

The goal of RL is to follow a policy or strategy that maximizes the reward





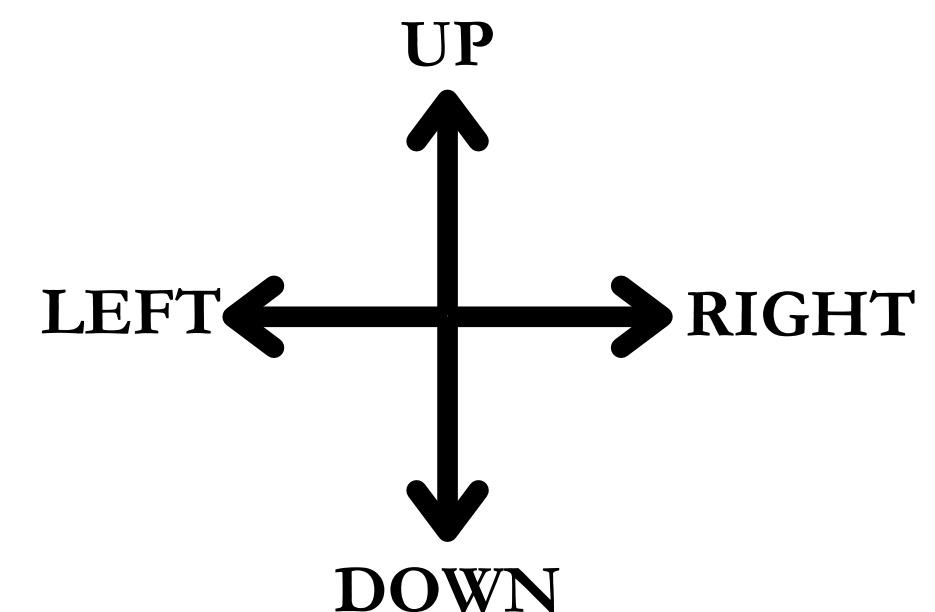
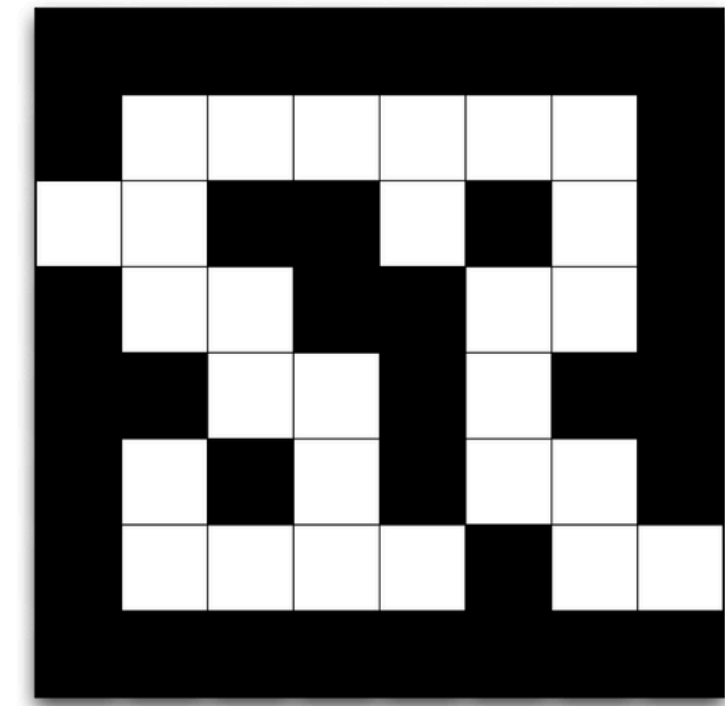
Environment and Actions

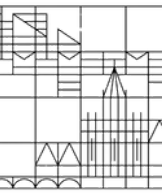
Environment:

- The setting or context within which the agent operates.
- It can be physical (like a robot navigating a room) or virtual (like a video game or simulation).
- Characteristically dynamic, potentially complex, and initially unknown to the agent.

Actions:

- Actions represent the choices available to the agent at each state.
- The set of all possible actions available in a given environment constitutes the action space, which can be discrete or continuous





Reward and Discounted Reward

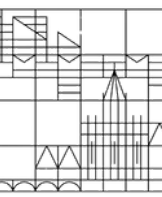
Whenever the agent performs something good, we should give it a reward r_t

- Rewards are signals sent from the environment to the agent
- They indicate the effectiveness of the action relative to achieving the goal.

To prioritize immediate rewards over distant ones and to handle the uncertainty of future rewards, RL uses a concept known as discounting.

- The value of future rewards is multiplied by a discount factor γ . This teaches the agent to prefer receiving rewards sooner rather than later, which is crucial in environments with changing conditions
- The total discounted reward starting from time t is then defined as

$$R_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \gamma^3 r_{t+3} + \dots = \sum_{k=0}^{\infty} \gamma^k r_{t+k}$$



Types of Reinforcement Learning



Policy Based Methods:

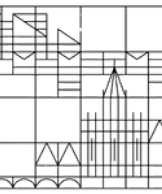
- They find the optimal policy, which defines the agent's way of behaving at a given time. A policy maps states to actions the agent should take, either deterministically or stochastically.

Value Function Methods:

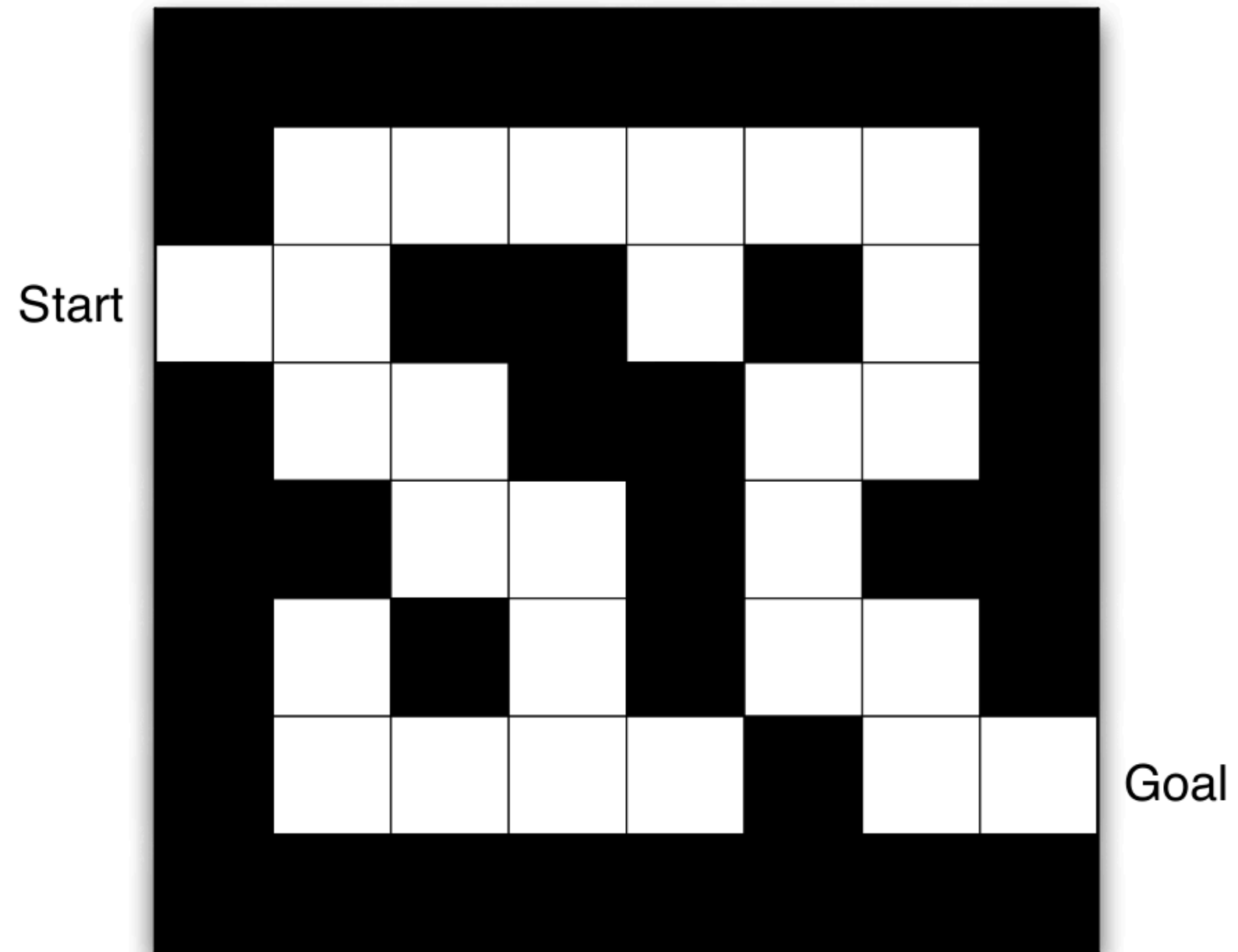
- They derive a function that gives the expected future rewards from each state. It is used to guide policy decisions and helps assess the potential of different states under a particular policy.

Model Based Methods:

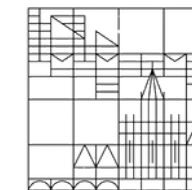
- They allow the agent to get an understanding of the environment. In model-based RL, the agent uses the model to predict how the environment will respond to its actions.



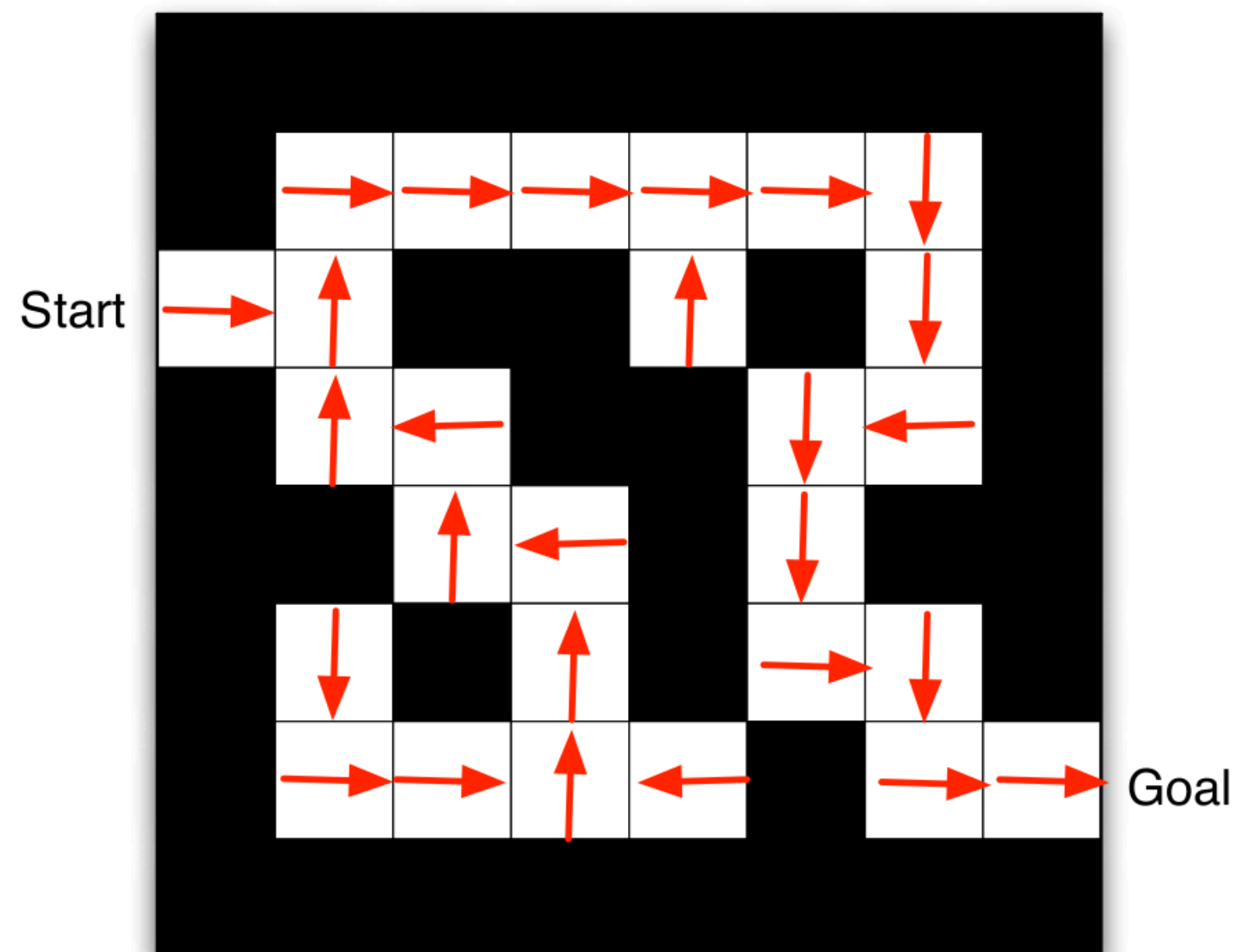
Example: Maze

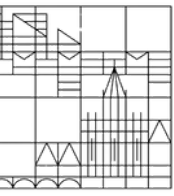


- **Rewards:** -1 per time-step
- **Actions:** up, down, left, right
- **States:** Agent's location in the maze

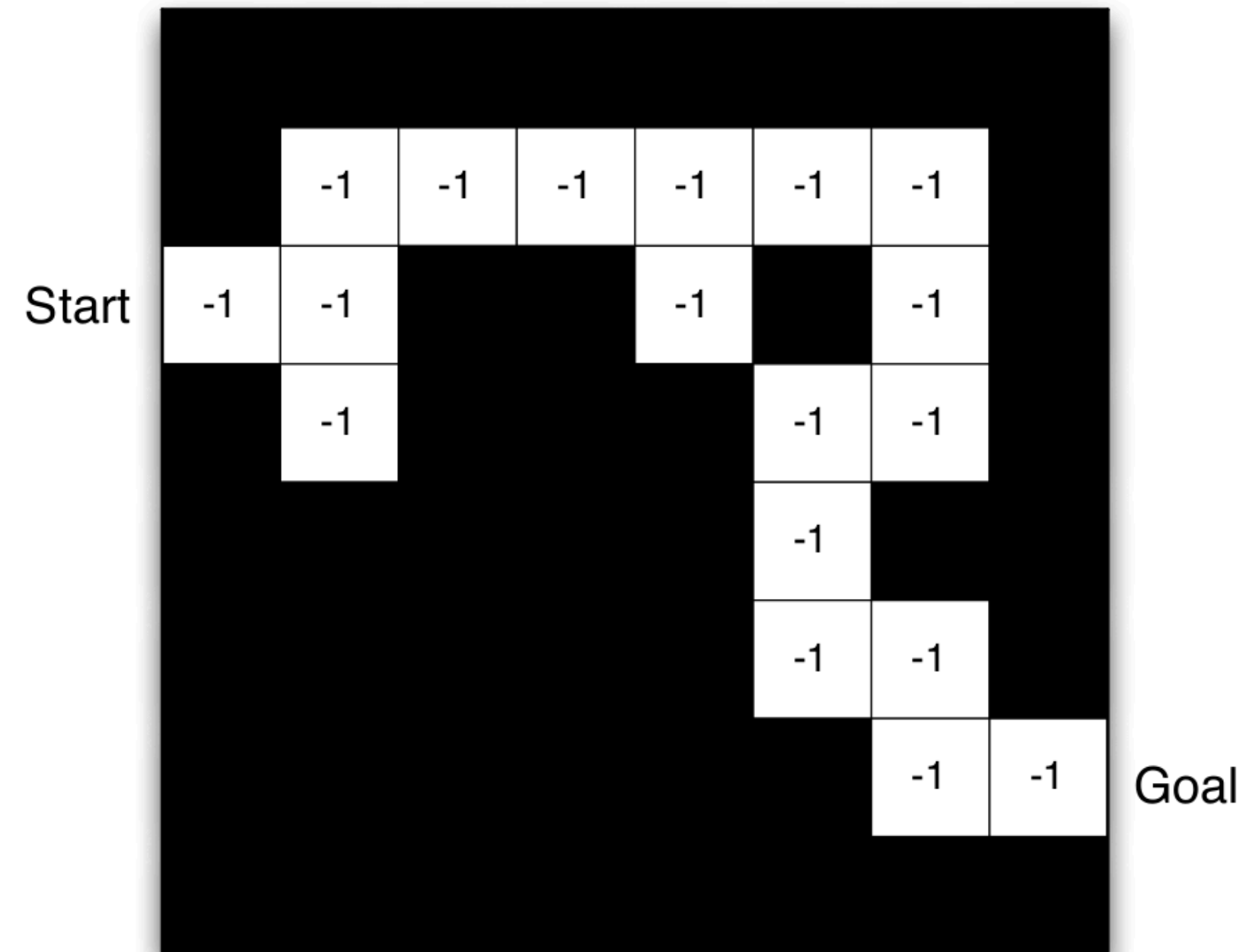


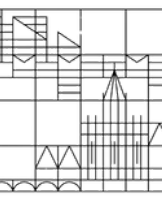
Example: Policy





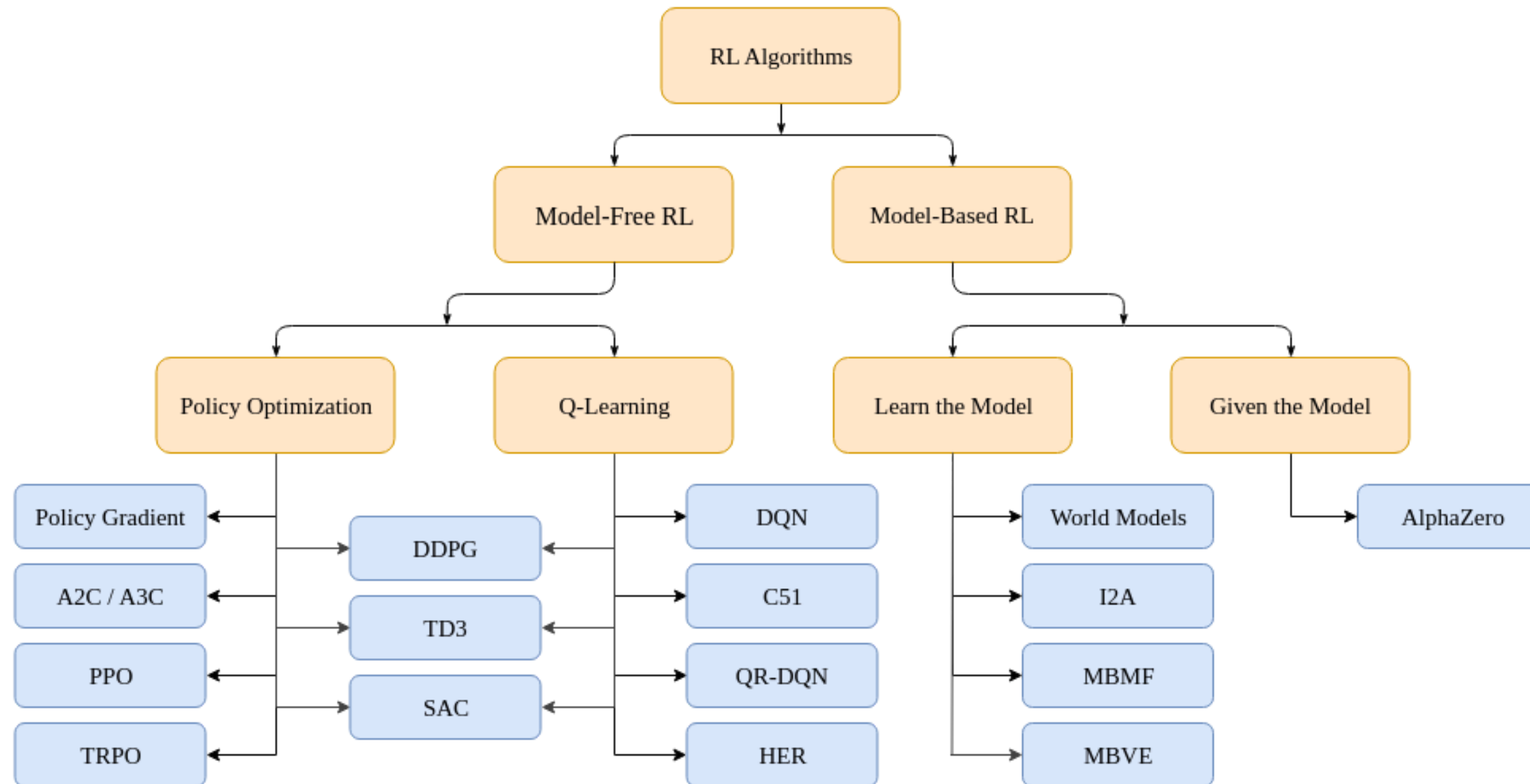
Example: Model

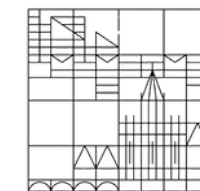




Taxonomy of Reinforcement Learning

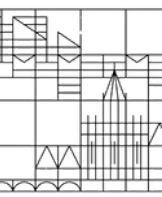
We will focus on Deep Q-Learning Networks (DQNs) and on Policy Gradient.





Deep Q-Learning and Policy Gradient

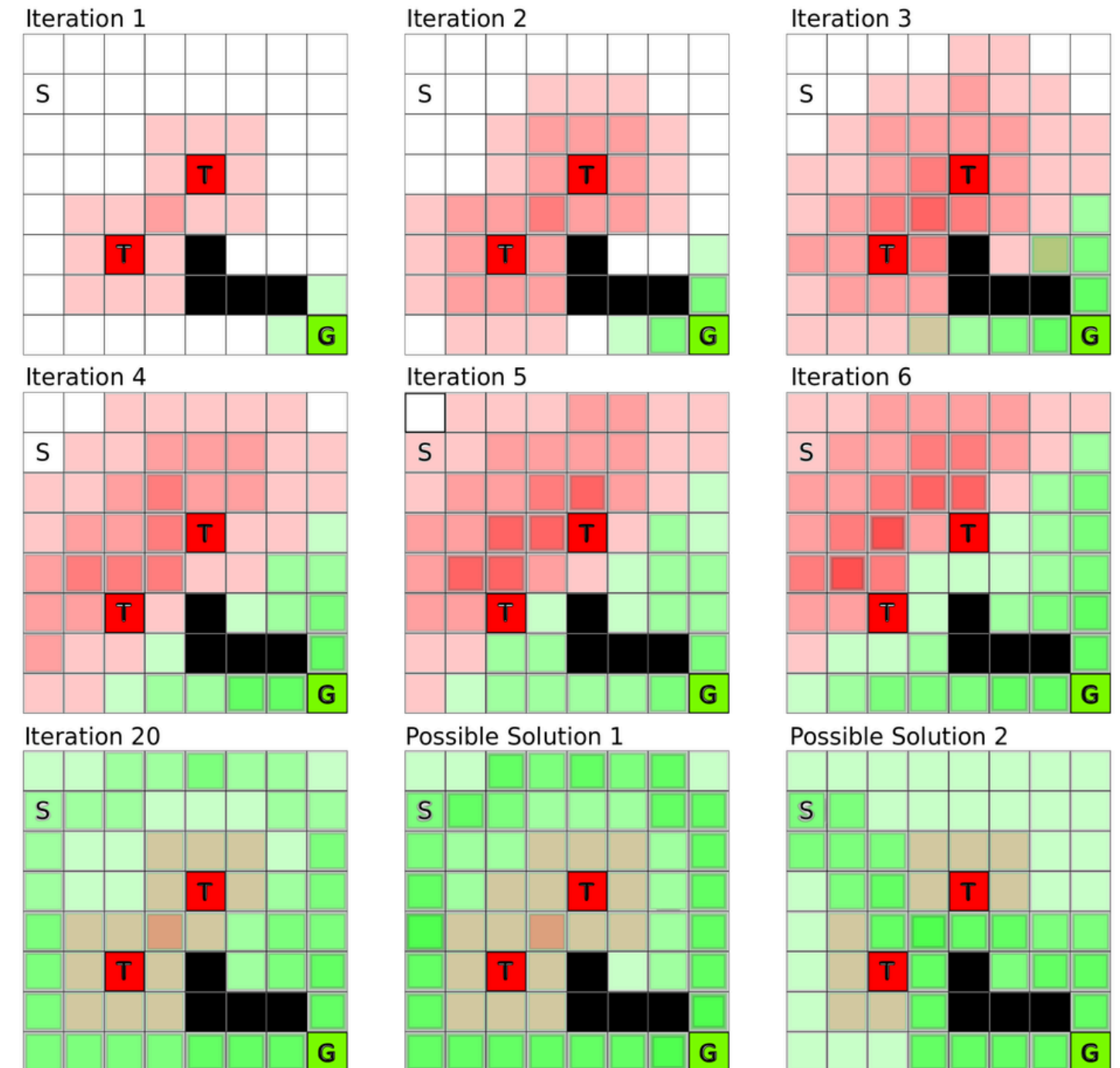


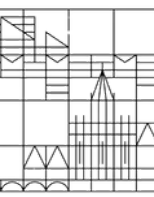


Q-Learning

Q-Learning is an off-policy reinforcement learning algorithm that seeks to find the best action to take given the current state.

- The algorithm uses a Q-function to evaluate the quality of a particular action taken from a particular state.
- The goal is to learn the Q-function and then to use it to get the maximal possible discounted reward.
- The agent observes the outcomes of actions taken, and iteratively updates the Q-values based on the rewards





Bellman's Formula

The Bellman equation allows to define the optimal Q-value function $Q^*(s, a)$ as:

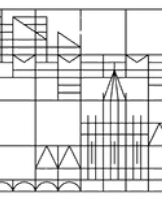
$$Q^*(s, a) = r + \gamma \max_{a'} Q^*(s', a')$$

In practice $Q^*(s, a)$ is approximated using the Q-learning algorithm, which iteratively updates the Q-values based on experience:

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

Here's how it works:

1. **Initialization:** Q-values are typically set to zero for all state-action pairs.
2. **Experience:** For each action taken, observe the immediate reward r and the next state s' .
3. **Update:** Adjust the Q-value of the state-action pair based on Bellman equation
4. **Repeat:** Iterate until the Q-values converge.



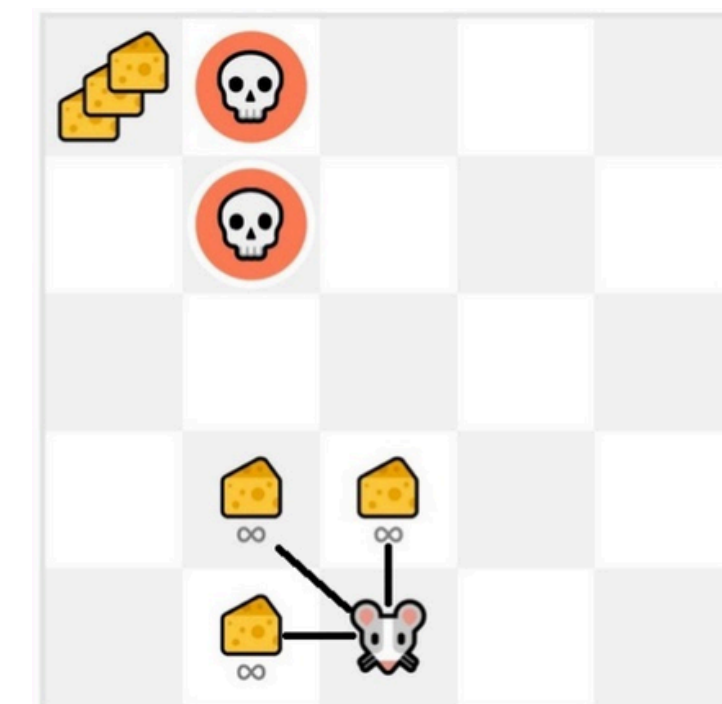
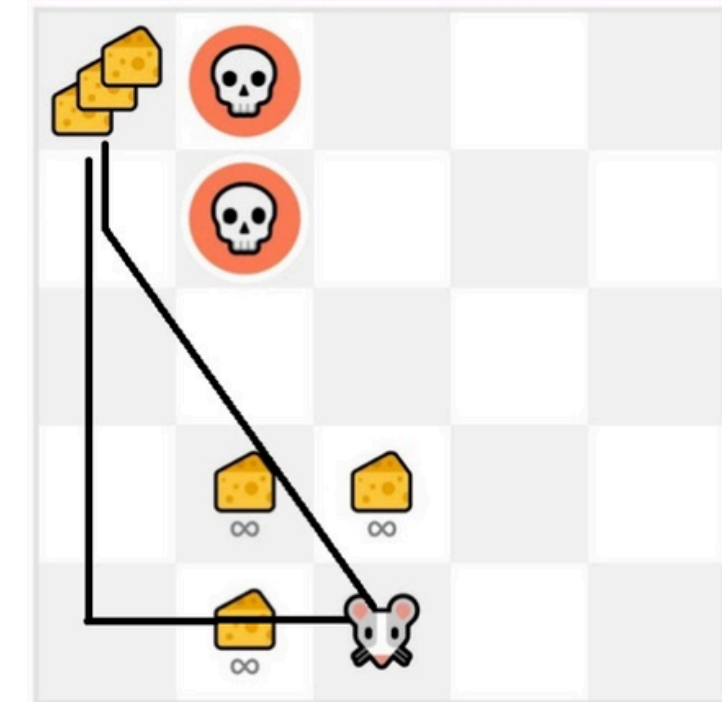
Exploration vs Exploitation

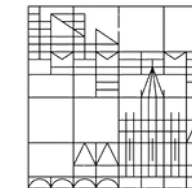
Balancing exploration and exploitation is a central point in RL

- Exploration is crucial in early learning stages or if conditions change over time. It prevents the algorithm from getting stuck in local optima.
- Exploitation is important for utilizing the learned behaviors to achieve the best results.

A common approach to this problem is the ϵ -greedy algorithm

- ϵ set the likelihood of taking a random action (exploration) versus the best-known action (exploitation).
- ϵ decreases gradually as learning progresses, allowing more exploration initially and more exploitation later.





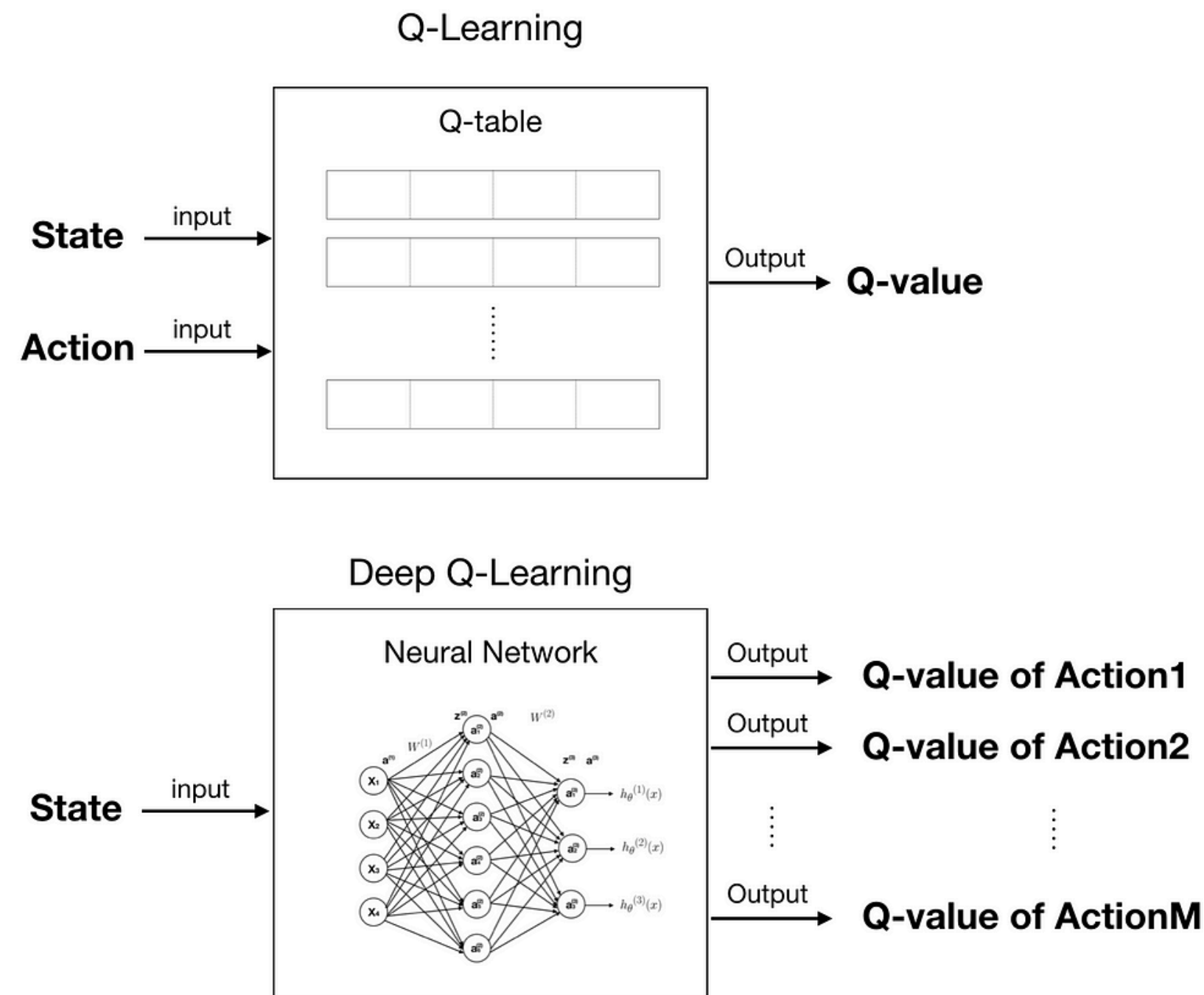
Deep Q-Learning Networks

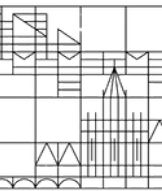
Traditional Q-Learning has many limitations

- It relies on a Q-table, which becomes impractical in environments with vast numbers of state-action pairs.
- In high-dimensional state spaces it requires exponential data to adequately cover the state-action space.

Deep Q-Networks (DQN) utilize a neural network to approximate the Q-value function

$$Q(s, a; \theta) \approx r + \gamma \max_{a'} Q(s', a'; \theta)$$





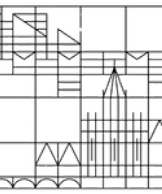
DQN Loss Function and Training

The loss function for DQN is typically framed as the Mean Squared Error (MSE) between the predicted Q-values and the target Q-values computed using the Bellman equation:

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

Training data in DQN is generated through the agent's interactions with the environment

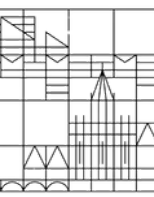
- At each timestep, the agent observes the current state s , selects and executes an action a , and observes the outcome, which includes the next state s' and reward r .
- All this information is stored in memory and they are later used to update the weights of the neural network with the standard optimization procedure.
- The updated neural network is the used to gather new data and the process is iterated.



DQNs Training Tricks

Training DQNs is not trivial and several tricks have been devised to make the process more stable

- **Experience Replay:**
 - instead of learning directly from consecutive experiences, the algorithm randomly samples a small batch of experiences from the buffer.
 - this random sampling helps to break the correlation and allows the network to learn from a more diverse set of experiences, thus stabilizing the update steps.
- **Target Network:**
 - the Target Network is a less frequently updated neural network that is added to the algorithm
 - the Target Network is used to estimate the target Q-values
 - this addresses the "moving target" problem, where continuous updates can make the training process unstable and inefficient.



Policy Gradient

Policy Gradient methods directly optimize the policy function π . They work by adjusting the policy parameters θ directly in response to an estimator of the long-term reward.

- It is effective in high-dimensional or continuous action spaces.
- It can learn stochastic policies, allowing for more exploration.

The policy function is approximated by a neural network and the objective is to maximize the expected reward $J(\theta)$

$$J(\theta) = \mathbb{E}_{\pi_{\theta}}[R_t]$$

Policy gradient updates parameters via a standard gradient ascent:

$$\theta \leftarrow \theta + \alpha \nabla_{\theta} J(\theta)$$



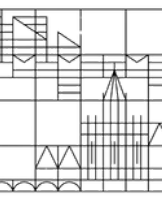
REINFORCE Algorithm

REINFORCE, is a simple policy gradient method. Policy parameters are updated as

$$\theta \leftarrow \theta + \alpha \sum_{t=0}^T R_t \nabla_{\theta} \log \pi_{\theta}(a_t | s_t)$$

Training is done as detailed below

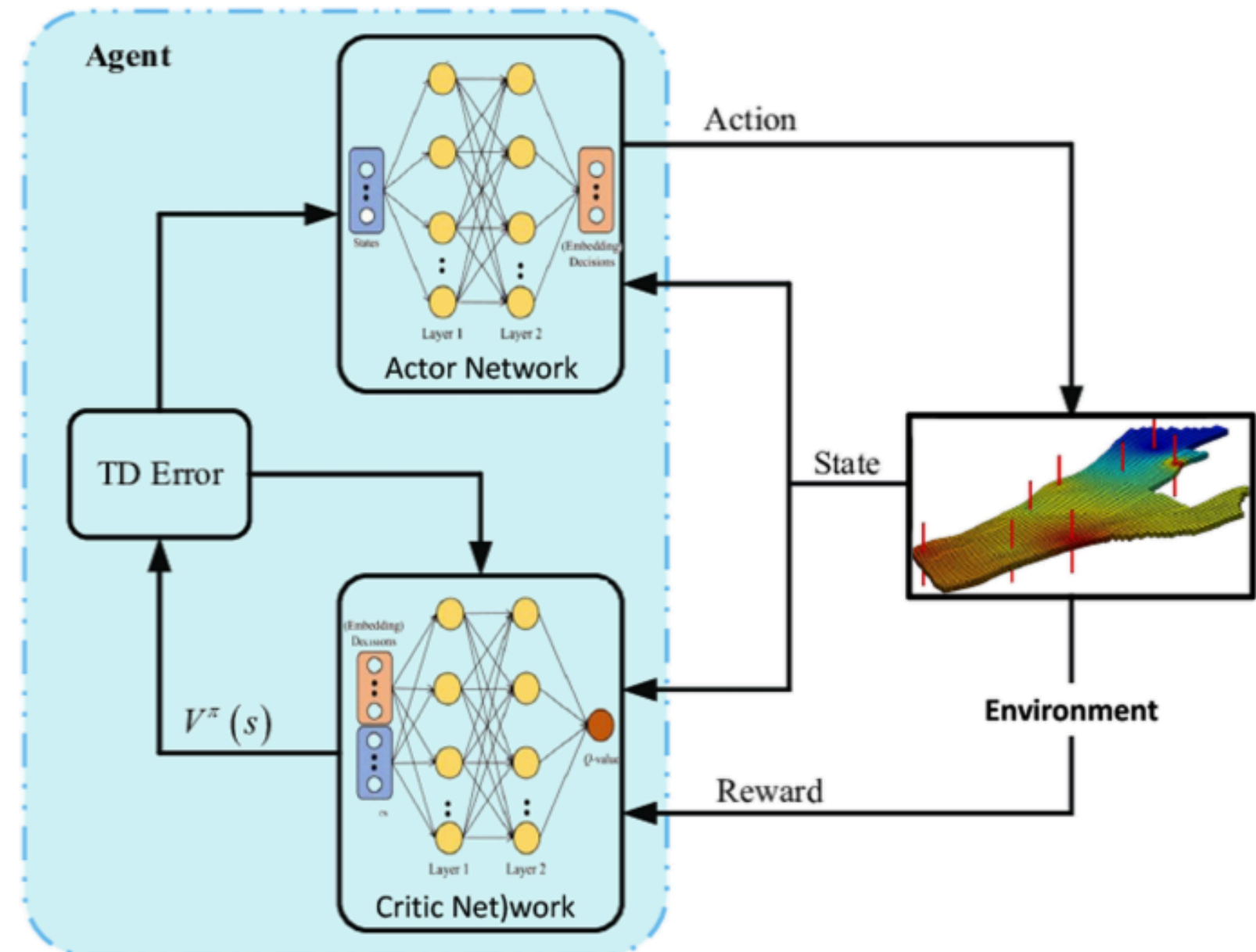
- The agent observes the current state s , selects an action a based on its current policy, and observes the outcome, which includes the reward r and the next state s' .
- This sequence of states, actions, and rewards is collected until the episode ends. The returns R_t for each timestep are calculated post-episode.
- These returns are used to update the policy parameters θ via gradient ascent
- The updated policy is then employed to gather new data in subsequent episodes, repeating the process.

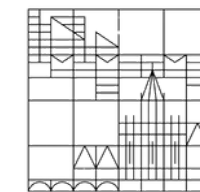


The Actor-Critic Framework

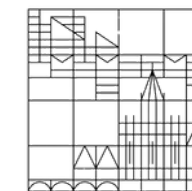
The Actor-Critic framework combines the benefits of both policy-based and value-based approaches in reinforcement learning.

- This method uses two models: the Actor, which proposes actions given the current state, and the Critic, which evaluates these actions by estimating the value function.
- By providing constant feedback after every action, the Critic helps the Actor adjust its policy in a more timely and informed manner.



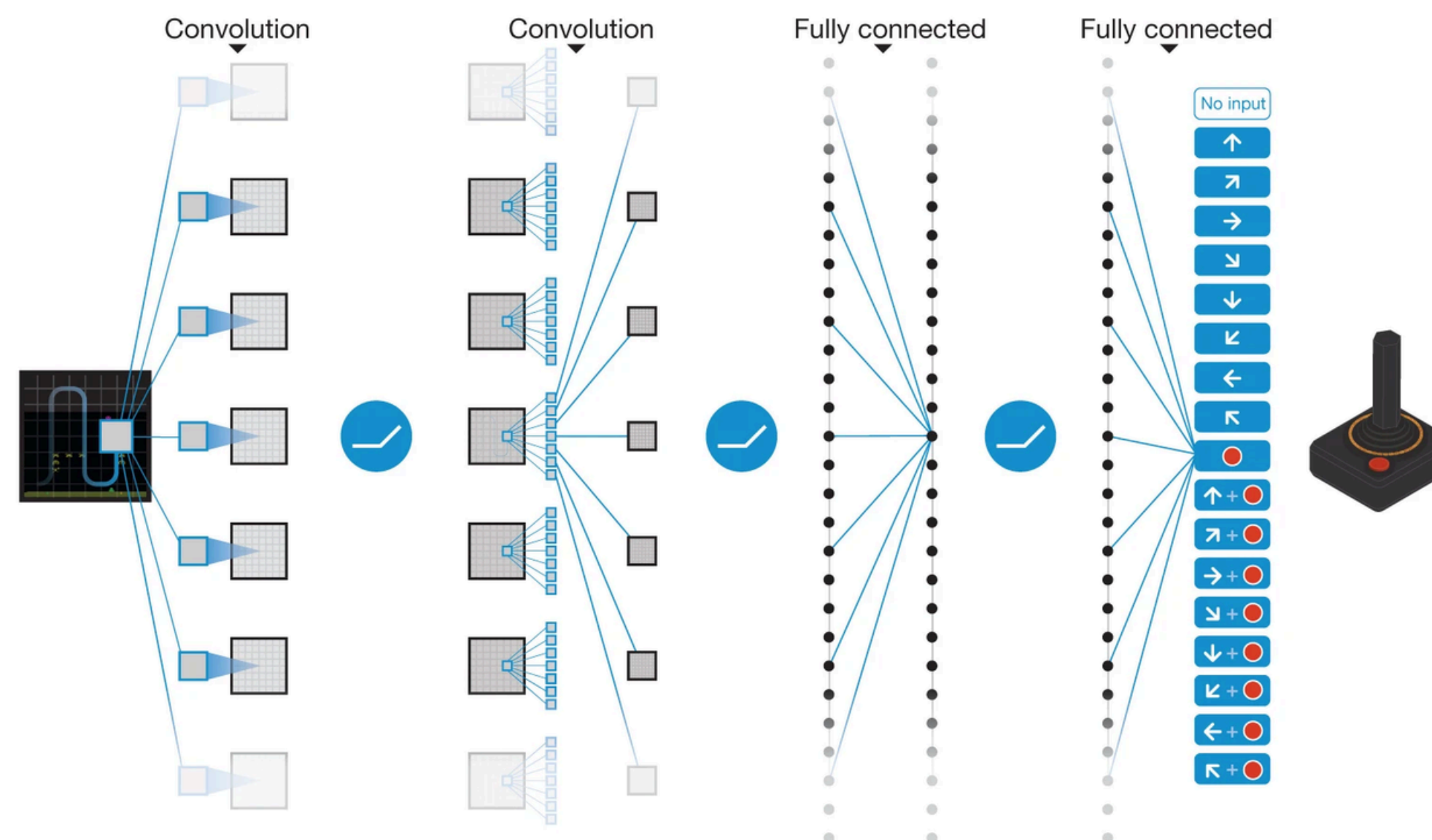


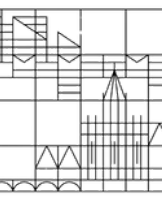
Applications of Reinforcement Learning



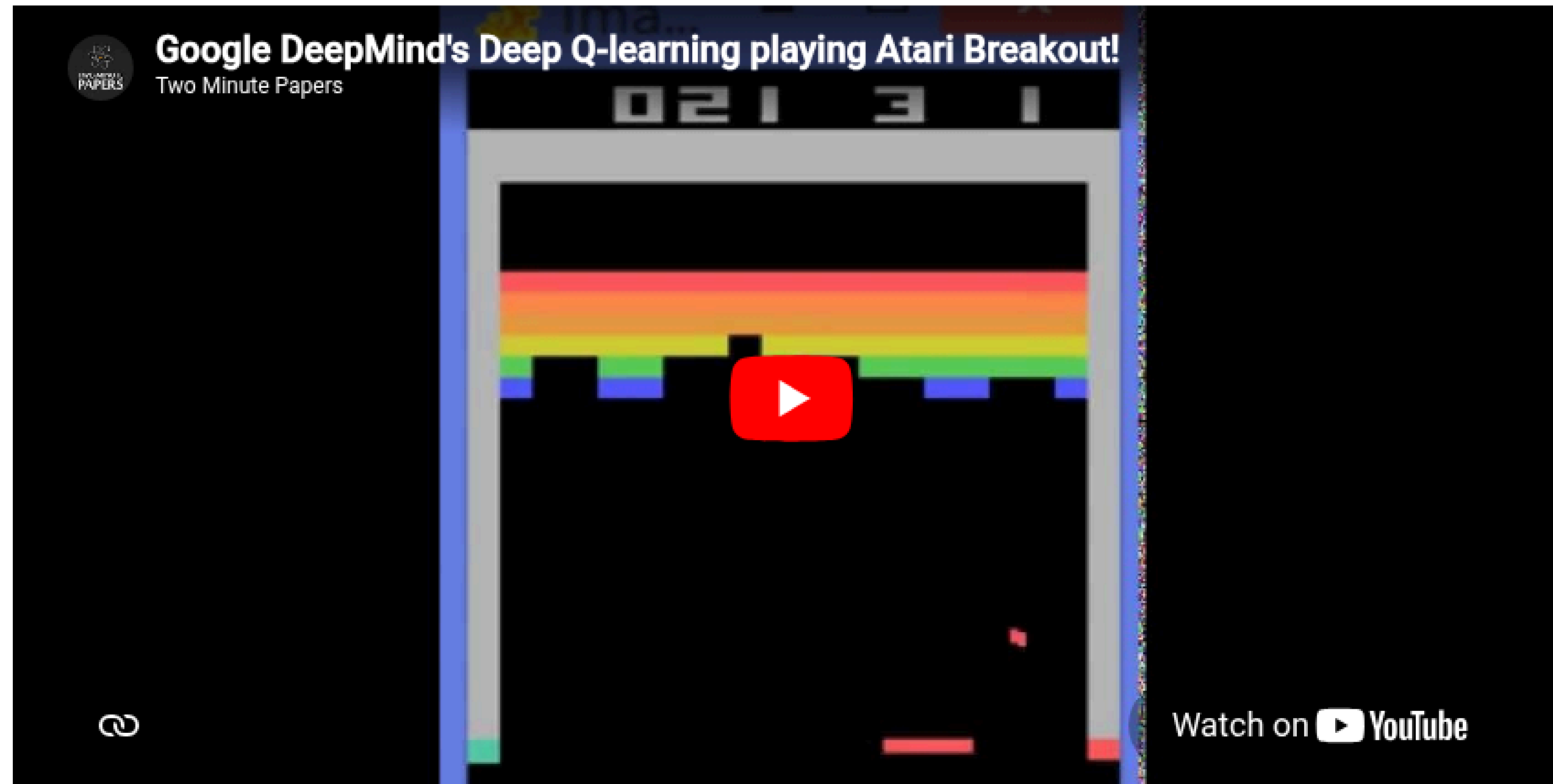
DQNs Beating Atari Games

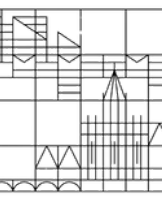
The first big breakthrough in RL has been a famous [paper](#) by Google Deep Mind using DQNs and CNNs to play Atari games with RL (2013)





DQN Playing Atari





Other Games

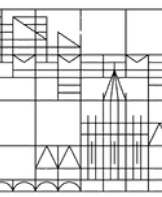
 **Reinforcement Learning Agent playing various Atari games using novelty driven ex**
Suraj Narayanan Sasikumar



The screenshot shows the Atari game 'Endorphin'. The player character is a small figure on a checkered floor, standing next to a ladder. A green arrow points down from the top of the screen, indicating the player's current action. The game environment includes a purple bar at the top and a ladder in the center.



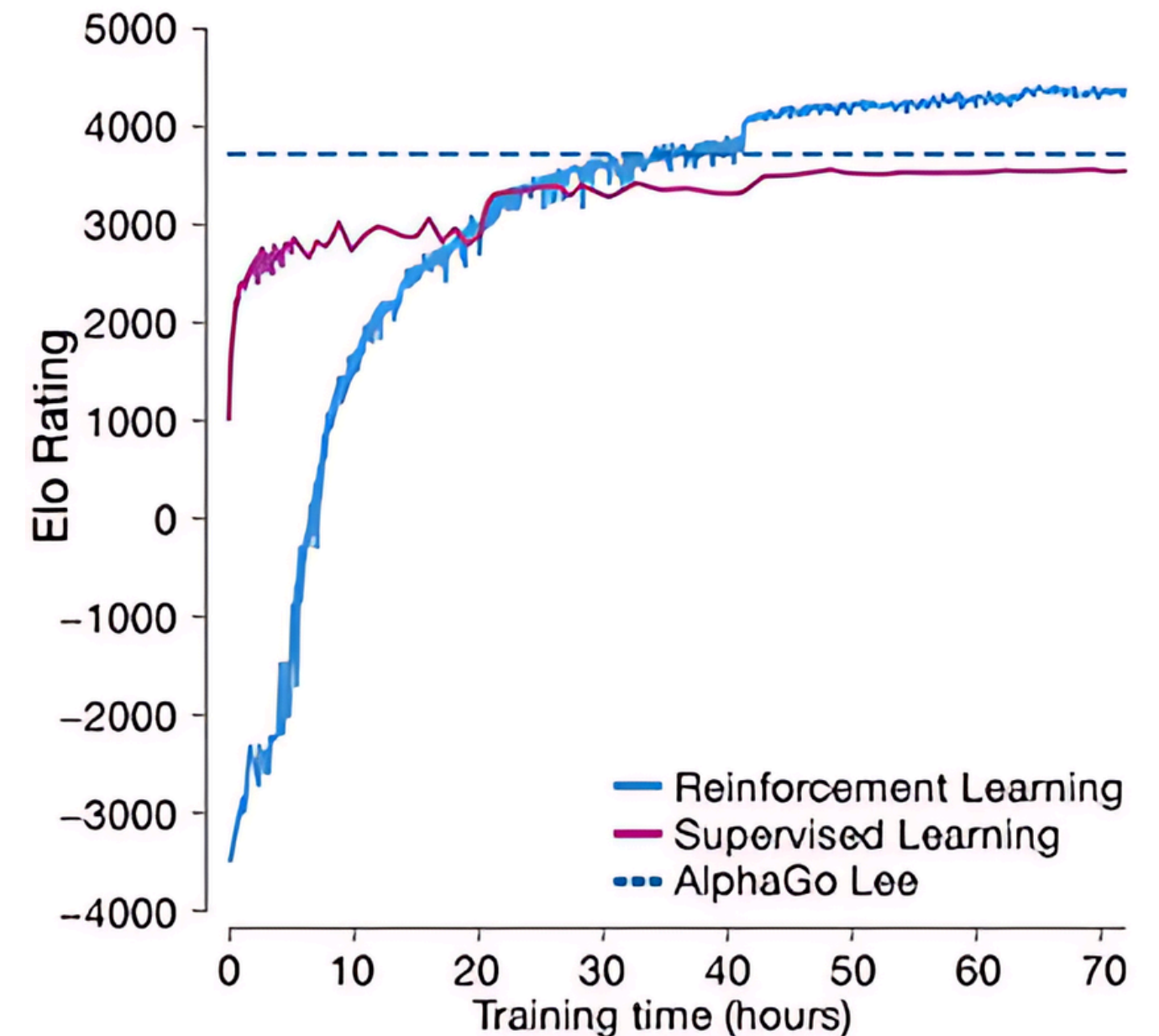
 Watch on  YouTube

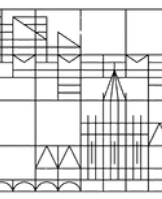


AlphaGo and AlphaZero

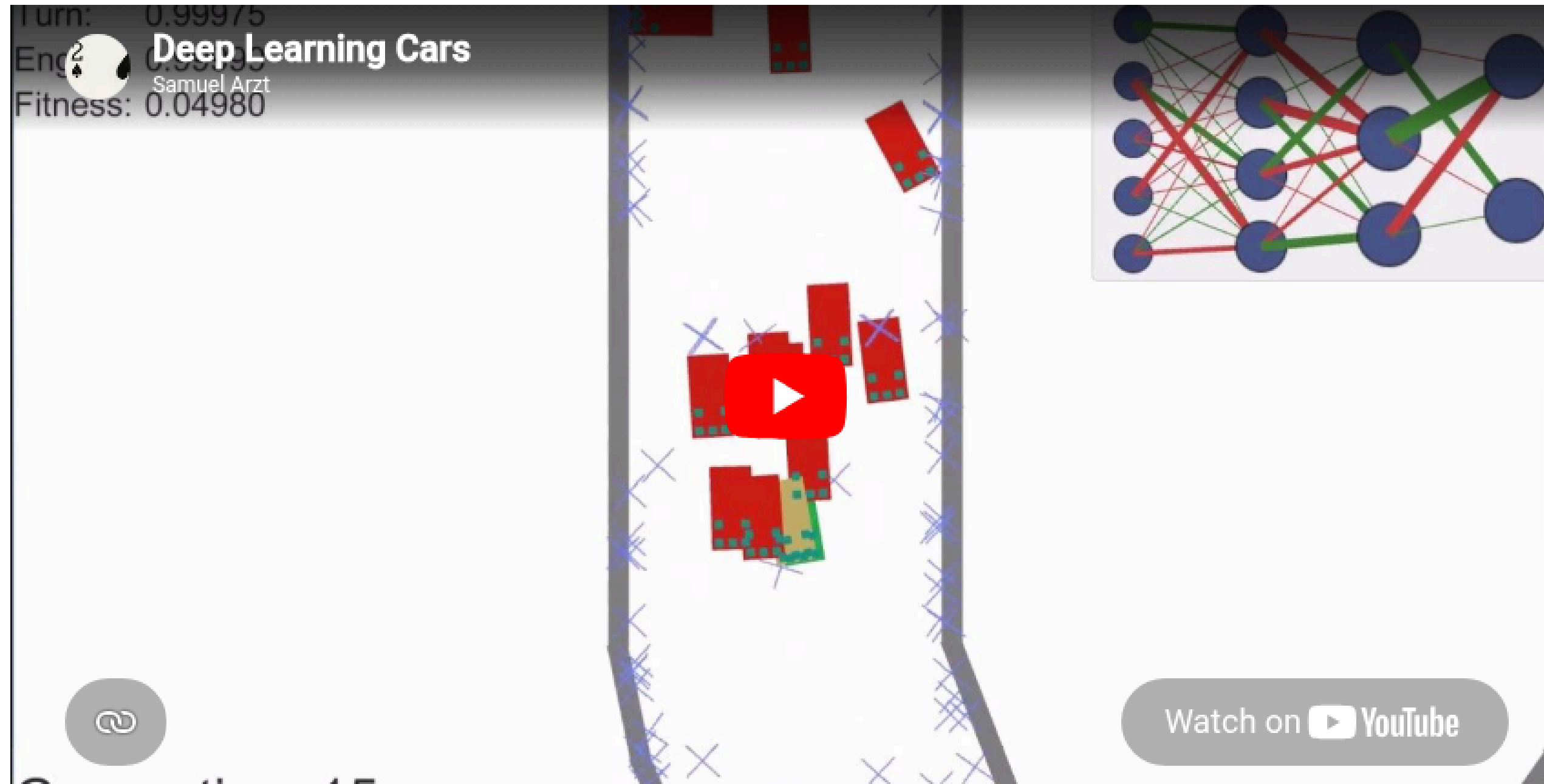
In 2016 Deep Mind makes another breakthrough with AlphaGo, beating the go world champion

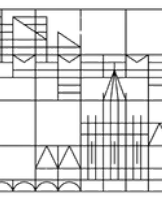
- AlphaGo was initially trained on professional human games to learn winning patterns and then improved through self-play, refining its policy and value networks.
- AlphaZero enhances this approach by using purely self-play reinforcement learning without any human data, starting from random play. It learns optimal strategies in games like Go, chess, and Shogi by playing millions of games against itself.





Self-Driving Cars?



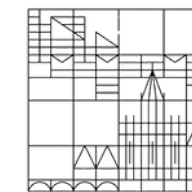


Reinforcement Learning with Human Feedback

While traditional language models excel at predicting text, they often fail to produce appropriate responses

- **Traditional training:** Models learn to predict next tokens from massive internet text corpora
- **Core limitation:** High perplexity \neq helpful responses to human queries
- **Alignment gap:** Models generate fluent but often irrelevant, verbose, or harmful content
- **RLHF solution:** Uses reinforcement learning with human feedback to optimize for human-preferred responses

This approach transforms capable but misaligned language models into helpful and aligned AI assistants.



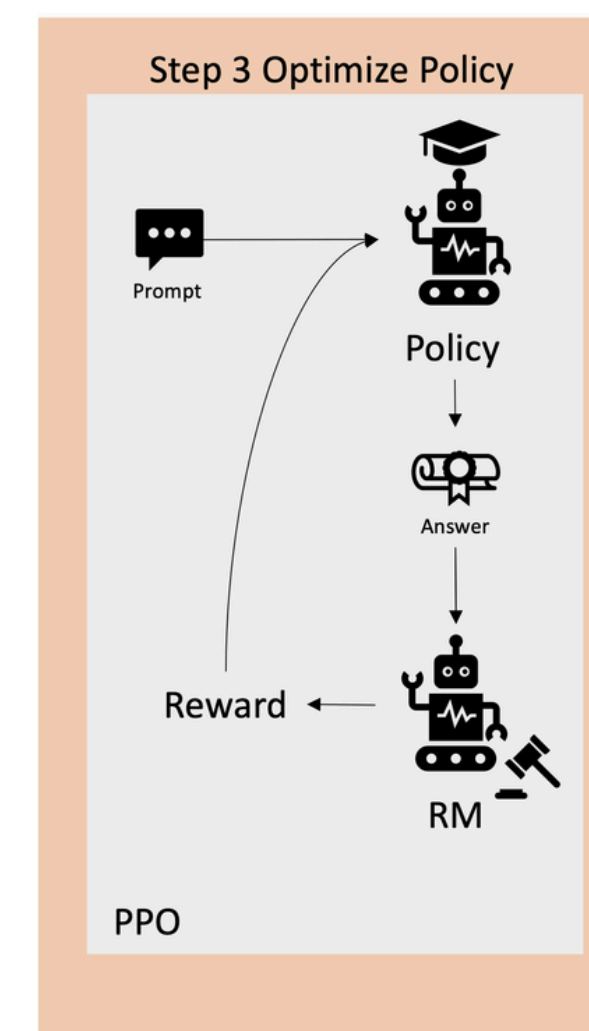
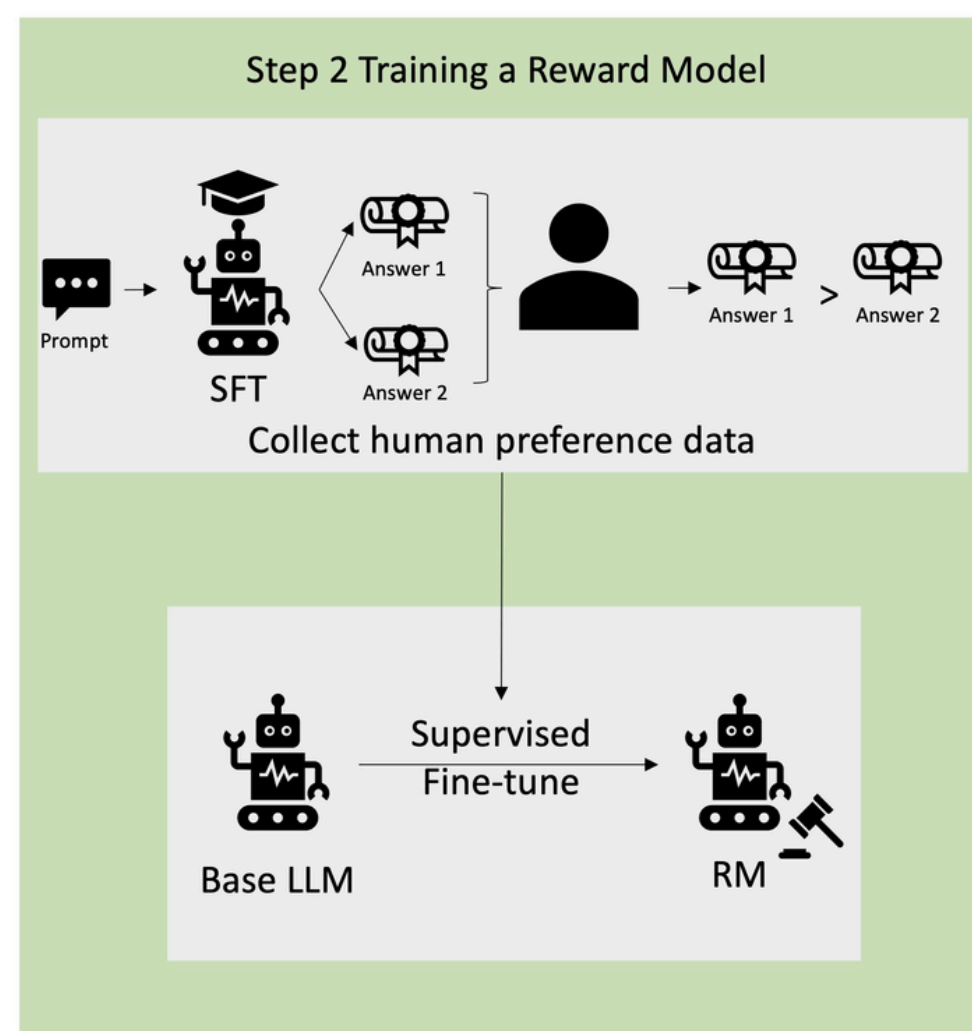
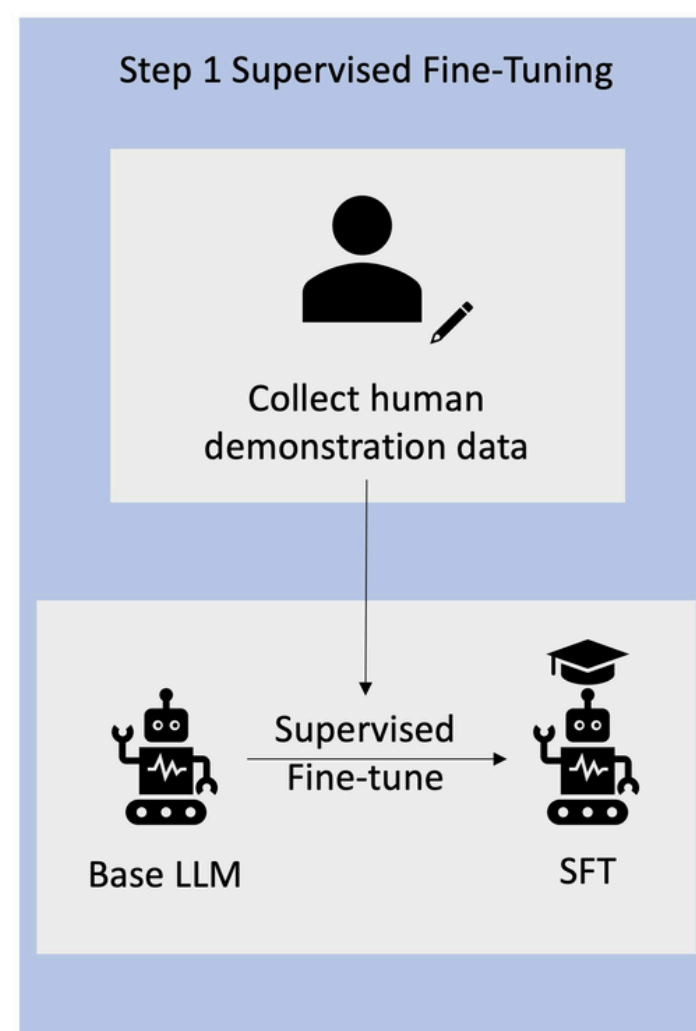
The RLHF Process

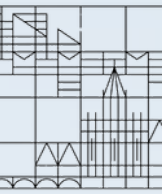
RLHF consists of a three-stage pipeline that progressively aligns model behavior

Fine-tune base LLM on high-quality human demonstrations to create initial helpful behavior

Collect human preference comparisons (A vs B) and train a neural network to predict human preferences

Optimize the language model against the reward model





Summary

A New Learning Paradigm

Reinforcement learning solves the problems of supervised and unsupervised learning, combining their benefits.

Reinforcement Learning

In reinforcement learning agents learn by interacting with an environment and performing actions. They learn optimal strategies by maximizing the reward they receive.

Deep Q-Learning and Policy Gradient

DQNs are neural networks that learn the Q-value function, providing the agent an optimal policy. Policy gradient instead directly optimize the policy.

Applications of Reinforcement Learning

Reinforcement learning has found applications especially in games such as chess or go. At present days the most relevant application is probably in LLMs.