Reinforcement Learning

Introduction

Reinforcement Learning is a branch of machine learning where an agent learns to make decisions by interacting with an environment.

Key milestones and in RL:

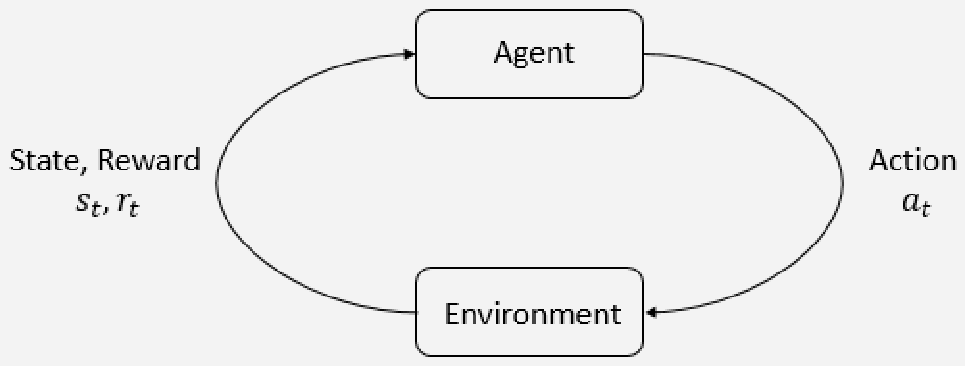
- 1989: Introduction of Q-Learning by Chris Watkins.
- 1992: Temporal Difference (TD) learning methods popularized by Sutton.
- 2013: DeepMind's Deep Q-Network (DQN) demonstrating human-level performance on Atari games.
- 2016: AlphaGo defeats a world champion Go player.
- 2020: MuZero achieves state-of-the-art performance in board games without prior knowledge of the rules.

Comparison with other types of machine learning:

- Supervised Learning: Learning from labeled data to predict outcomes.
- Unsupervised Learning: Finding patterns in data without labeled outcomes.
- Reinforcement Learning: Learning optimal actions through trial and error interactions with an environment.

Terminology

- Agent: The decision-maker and learner that interacts with the environment.
- Environment: Everything outside the agent that it interacts with.
- Episode: agent-environment interactions from initial to final states
- State: A representation of the current situation of the environment.
- State space: discrete, continuous, multi-discrete.
- Action: Decisions or moves the agent can make.
- Action space: discrete, continuous, multi-discrete.
- Reward: Feedback from the environment indicating the success of an action.
- Discounted Reward: Immediate vs. cumulative rewards.
- Policy: The strategy used by the agent to decide actions based on states.
- Value Function: Estimates the expected cumulative reward, if a specific action is taken.



Loop

- Agent observes state
- Agent selects action
- Environment transitions to new state S_t+1 and provides reward R_t
- Agent updates policy or value function based on R_t and S_t+1.

Core idea: learning through interaction from the environment. The agent continuously interacts with the environment, receives feedback in the form of rewards, and updates its policy to improve future actions.

Reinforcement Learning Algorithms

Reinforcement Learning Algorithms allow an agent to learn optimal actions through interactions with an environment to maximize cumulative rewards

Model-Free RL

Model-Free: Learn directly from experiences without an explicit model of the environment

Value Estimation

Value estimation algorithms in reinforcement learning estimate the expected rewards (reward function) of actions in different states, guiding the agent to choose actions that maximize long-term rewards.

Q-Learning: An algorithm that learns the value of actions in each state.

- Significance: Fundamental algorithm for learning the optimal action-value function, foundational to many subsequent RL methods.
- Approach: Uses a 2D table where each cell corresponds to a state-action pair, and the number in each cell is the Q-value. The Q-value represents the expected reward for taking a given action in a specific state, considering the rewards received and the estimated value of the next state.

Deep Q-Networks (DQN): Uses neural networks to approximate Q-values.

- Paper: "Playing Atari with Deep Reinforcement Learning" (2013) by Mnih et al.
- Significance: Demonstrated the application of deep learning to RL, using high-dimensional sensory inputs.
- Approach: Uses a neural network to approximate the Q-value function
- Resources: Youtube, Youtube

Double DQN: An improvement on DQN to reduce overestimation bias in Q-value estimates.

- Paper: "Deep Reinforcement Learning with Double Q-learning" (2015) by Hado van Hasselt et al.
- Significance: Introduced a more accurate method for value estimation, leading to better performance and stability in training.
- Approach: Uses two separate Q-networks to decouple action selection from action evaluation.

Dueling DQN: Enhances the DQN architecture by separating the value and advantage functions.

- Paper: "Dueling Network Architectures for Deep Reinforcement Learning" (2015) by Ziyu Wang et al.
- Significance: Demonstrated improved learning efficiency and policy performance by better capturing the value of states and actions.

- Approach: Introduces two streams in the neural network architecture to separately estimate state values and advantages for actions.

Categorical DQN (C51): An extension of DQN that represents the value distribution of returns using a fixed number of atoms (categories) rather than just the mean.

- Paper: "A Distributional Perspective on Reinforcement Learning" (2017) by Marc G. Bellemare, Will Dabney, and Rémi Munos.
- Significance: Improves the stability and performance of DQN by capturing the full distribution of returns. Provides a richer representation of the uncertainty in the value estimates.
- Approach: Uses a categorical distribution to approximate the return distribution. Implements a projection step to fit the distribution during updates. Updates the value distribution using the Bellman update with a distributional perspective.

Rainbow DQN: Combines several DQN extensions into a single architecture to improve performance and stability.

- Paper: "Rainbow: Combining Improvements in Deep Reinforcement Learning" (2018) by Matteo Hessel, Joseph Modayil, Hado van Hasselt, et al.
- Significance: Offers state-of-the-art performance by integrating multiple improvements into a unified framework. Provides a comprehensive benchmark for value-based deep RL methods.
- Approach: Integrates six key improvements: Double DQN, Prioritized Experience Replay, Dueling Network Architectures, Multi-step Learning, Categorical DQN(C51), and Noisy Nets.

Policy Optimization

Policy optimization algorithms in RL aim to learn the optimal mapping from states to actions directly, without the need to estimate the value function, thus simplifying the learning process.

REINFORCE: Directly optimizes the policy by following the gradient of the expected reward.

Trust Region Policy Optimization (TRPO): Ensures stable and efficient policy updates.

- Paper: "Trust Region Policy Optimization" (2015) by John Schulman et al.

- Significance: Provided a reliable method for policy optimization with theoretical guarantees on performance improvement.
- Approach: Uses a constraint on the size of policy updates to maintain stability during training.

Proximal Policy Optimization (PPO): Balances exploration and exploitation.

- Paper: "Proximal Policy Optimization Algorithms" (2017) by Schulman et al.
- Significance: Simplified the implementation while maintaining the benefits of TRPO (small and safe changes), leading to widespread adoption in various RL applications.
- Approach: Uses clipped probability ratios to constrain policy updates, balancing exploration and exploitation effectively.

Actor-Critic

Actor-Critic methods combine elements of both value-based and policy-based approaches in reinforcement learning, where an actor learns the policy while a critic learns the value function, enabling more stable and efficient learning by leveraging the strengths of both frameworks.

Advantage Actor-Critic (A2C): Combines policy gradients (actor) with value function approximation (critic).

- Paper: "Asynchronous Methods for Deep Reinforcement Learning" (2016) by Volodymyr Mnih et al.
- Significance: Showed improved performance over traditional actor-critic methods by leveraging parallel environments.
- Approach: A2C combines value estimation and policy optimization to learn both functions simultaneously, offering a balance between efficiency and simplicity.
- Approach: Asynchronous Advantage Actor-Critic (A3C) updates the actor and critic networks asynchronously, allowing multiple workers to explore different parts of the environment parallelly.

Deep Deterministic Policy Gradient (DDPG): Environments with continuous action spaces.

- Paper: "Continuous Control with Deep Reinforcement Learning" (2016) by Timothy P. Lillicrap et al.
- Significance: Enabled the application of RL to high-dimensional continuous action spaces, such as robotic control.

- Approach: Uses a deterministic policy with target networks and experience replay to stabilize training and improve performance.

Soft Actor-Critic (SAC): An off-policy actor-critic algorithm designed for maximum entropy reinforcement learning.

- Paper: "Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor" (2018) by Tuomas Haarnoja et al.
- Significance: Demonstrated state-of-the-art performance in continuous control tasks by encouraging exploration through entropy maximization.
- Approach: Uses a stochastic actor and a critic network to optimize a maximum entropy objective, balancing exploration and exploitation.

Model-Based RL

Model-Based: Use a model of the environment to plan and make decisions

Planning

Planning techniques assume a given model of the environment and use it to predict future states and optimize actions.

Monte Carlo Tree Search (MCTS): Uses a tree structure to explore possible future states through simulations, optimizing decisions based on the evaluated outcomes.

- Paper: "A Survey of Monte Carlo Tree Search Methods" (2012) by Cameron B. Browne et al.
- Significance: Revolutionized game-playing AI by providing a powerful and flexible search method for large state spaces.
- Approach: Iteratively builds a search tree by simulating many random plays (rollouts), refining the tree based on the outcomes to guide decision-making. The search process involves four main steps:
    - Selection:Traverse the tree from the root to a leaf node using a policy that balances exploration and exploitation, often using Upper Confidence Bound (UCB).
    - Expansion: Add one or more child nodes to the selected leaf node if it is not a terminal state.
    - Simulation: Perform a random simulation from the newly added node to estimate the outcome.

    - Backpropagation: Update the value estimates of the nodes along the path from the new node to the root based on the simulation result.
- Note: MCTS requires a model of the environment to simulate future states. It can be combined with model-free approaches when the simulation model is derived from synthetic data.
- Resources: Youtube, Youtube

A* Search: Uses heuristics to find the most promising path to the goal node.

- Paper: "A Formal Basis for the Heuristic Determination of Minimum Cost Paths" (1968) by Peter E. Hart, Nils J. Nilsson, and Bertram Raphael.
- Significance: A* search is one of the most widely used pathfinding algorithms because of its efficiency and optimality when a heuristic is available and consistent.
- Approach: Iteratively finds the optimal path from a start node to a goal node by using a priority queue (min-heap) to evaluate nodes based on cost and heuristic estimates. The process involves six main steps:
    - Initialization: Start from the initial state and place it in a priority queue with its initial cost (just the heuristic estimate to the goal).
    - Selection: Remove the node with the lowest cost value (sum of the cost to reach the node and the heuristic estimate to the goal) from the priority queue (min-heap). If this node is the goal state, proceed to path construction.
    - Expansion: Add neighbors of the selected node. For each neighbor, calculate the g value (cost from the start to the current node), h value (heuristic estimate to the goal), and f value ($f = g + h$).
    - Evaluation: If a neighbor is not in the priority queue or has a lower f value than previously found, add it to the priority queue or update its value and parent pointer.
    - Marking: Place the selected node in a closed list to mark it as evaluated.
    - Repeat the process from selection until the goal state is reached.
    - Path Construction: Once the goal state is reached, reconstruct the optimal path by tracing back from the goal to the start node using parent pointers or using the removed list from the min-heap.
- Note: A* search is a deterministic pathfinding algorithm, used in navigation and route planning.

Dynamic Programming (DP) for MDPs: Solves Markov Decision Processes (MDPs) by breaking down the problem into simpler subproblems.

- Key Concepts:
    - MDPs: Provide a mathematical framework for decision-making in environments with stochastic outcomes.
    - Components of MDPs: States (S), Actions (A), Transition Function (T), Reward Function (R), Discount Factor (γ).
- Value Iteration Algorithm:
    - Objective: Find the optimal value function V*.
    - Approach: Iteratively update V(s) using the Bellman equation until convergence.
- Policy Iteration Algorithm:
    - Objective: Find the optimal policy π*.
    - Approach: Alternates between policy evaluation and policy improvement until the policy converges.
- Paper: "Dynamic Programming and Markov Processes" (1960) by Ronald A. Howard.
- Significance: DP provides foundational algorithms for solving MDPs
- Note: DP methods assume a perfect model of the environment, making them suitable for problems with known transition dynamics.

Model Predictive Control (MPC): An optimization-based control strategy that uses a model of the system dynamics to optimize control actions.

- Paper: "Model Predictive Control: Theory and Practice—A Survey" (2000) by E.F. Camacho and C. Bordons.
- Significance: MPC is widely used in industrial applications for its ability to handle multi-variable control problems with constraints.
- Approach:
    - Model Formulation: Develop a predictive model of the system dynamics.
    - Optimization: At each time step, solve an optimization problem to find the control actions that minimize a cost function over a future horizon.
    - Receding Horizon: Apply the first control action from the optimized sequence and re-solve the optimization problem at the next time step with updated state information.
- Note: MPC is a control strategy used for dynamic systems requiring precise and adaptive control actions, applied in process control, robotics, and autonomous vehicles.

Model Learning

Model learning involves learning a model of the environment's dynamics from data and then using planning algorithms on this learned model.

Planning to Explore via Self-Supervised World Models: A method that leverages self-supervised learning to build world models, which are then used to plan exploratory actions in reinforcement learning environments.

- Paper: "Planning to Explore via Self-Supervised World Models" (2020) by Danijar Hafner, Alexander Pritzel, Mohammad Norouzi, and Jimmy Ba.
- Significance: Improves exploration efficiency by using learned models of the environment to simulate and evaluate potential actions.
- Approach: Uses self-supervised learning to train a world model that predicts future states and rewards. Employs planning algorithms, such as Model Predictive Control (MPC), to generate exploratory action sequences.
- Resources: Youtube, Website, Github

PlaNet (Deep Planning Network): Uses a latent dynamics model to predict future states and optimize actions by planning in the latent space.

- Paper: "Learning Latent Dynamics for Planning from Pixels" (2019) by Danijar Hafner et al.
- Significance: PlaNet enables planning directly from raw pixel observations, making it applicable to a wide range of visual tasks. It demonstrates high sample efficiency and performance in continuous control tasks.
- Approach: Learns a latent space model of the environment's dynamics and performs planning in this latent space using model-predictive control techniques.

World Models: Builds a generative model of the environment, combining visual and temporal models to simulate future trajectories for policy training.

- Paper: "World Models" (2018) by David Ha and Jürgen Schmidhuber
- Significance: World Models significantly reduce the need for extensive interactions with the real environment by leveraging learned models for simulation. This approach has shown success in solving complex tasks with fewer real-world trials.
- Approach: Uses a Variational Autoencoder (VAE) to model visual input and a Recurrent Neural Network (RNN) to model temporal dynamics, enabling simulation of future states for policy training.

- Resources: Paper walkthrough, Youtube

DreamerV1: Utilizes a learned dynamics model to simulate future states and optimize actions through imagined trajectories in a latent space.

- Paper: "Dream to Control: Learning Behaviors by Latent Imagination" (2020) by Danijar Hafner et al.
- Significance: Dreamer achieves high performance in continuous control tasks with high sample efficiency, demonstrating the power of learning and planning in latent spaces.
- Approach: Trains a dynamics model in a latent space and uses this model to simulate future trajectories, enabling policy optimization through imagined experiences.

SimPLe (Simulated Policy Learning): Generates synthetic experiences using a learned model and trains a policy with these experiences to improve performance in the real environment.

- Paper: "Model-Based Reinforcement Learning for Atari" (2020) by Lukasz Kaiser et al.
- Significance: SimPLe shows significant improvements in sample efficiency by using a learned model to generate additional training data. It effectively bridges the gap between model-free and model-based learning.
- Approach: Learns a model of the environment's dynamics and generates synthetic experiences, which are then used to train a policy in a model-free manner.

Sequence Modeling

Sequence modeling predicts future elements in a sequence based on previous ones, using models like recurrent neural networks (RNNs), Long short-term memories (LSTMs), and transformers to capture dependencies and patterns.

Decision Transformers: A model that uses transformer architectures to handle decision-making problems by treating them as sequence modeling tasks.

- Paper: "Decision Transformer: Reinforcement Learning via Sequence Modeling" (2021) by Lili Chen, Kevin Lu, Aravind Srinivas, et al.
- Significance: Converts reinforcement learning problems into sequence prediction problems and leverages the power of transformer models and large-scale pre training for policy learning.
- Approach: Uses transformer models to predict actions based on past experience, treating states, actions, and rewards as elements of a

sequence. Uses attention mechanisms to capture long-term dependencies and correlations in the trajectories.
- Resources: Youtube, Site, Github

Hybrid Methods

Hybrid algorithms integrate model-based planning with model-free learning, using a combination of learned models and policy/value networks to optimize decisions.

Value Prediction Network (VPN)

- Paper: "Value Prediction Network" (2017) by Junhyuk Oh et al.
- Significance: Integrates model-free and model-based RL methods into a single neural network, demonstrating advantages in complex and stochastic environments.
- Approach: Learns a dynamics model to make option-conditional predictions of future values rather than future observations, combining value prediction with planning in a unified framework.

DreamerV2: A method that combines discrete latent space models with reinforcement learning to achieve high performance in Atari games.

- Paper: "Mastering Atari with Discrete World Models" (2021) by Danijar Hafner, Timothy Lillicrap, Mohammad Norouzi, and Jimmy Ba.
- Significance: Demonstrates the effectiveness of discrete latent spaces for modeling complex environments. Achieves competitive performance on Atari games, showcasing the potential of discrete world models in reinforcement learning.
- Approach: Trains a discrete latent world model that compresses high-dimensional observations into a lower-dimensional latent space. Uses this latent space to predict future states and rewards, facilitating planning and policy learning. Uses model free algorithms such as A2C to optimize policies based on the predictions made by the learnt model.
- Resources: Youtube, Github, Author Blog, Google Blog

Dyna-Q: Combines model-free Q-learning with simulated experiences generated from a learned model of the environment to update Q-values.

- Paper: "Reinforcement Learning: An Introduction" (1998) by Richard S. Sutton and Andrew G. Barto

- Significance: Dyna-Q introduces a novel way to integrate planning with direct reinforcement learning, improving sample efficiency by using simulated experiences.
- Approach: Learns a model of the environment and uses it to generate additional experiences to update Q-values alongside real experiences.

MBPO (Model-Based Policy Optimization): Combines model-based rollouts with model-free policy optimization, enhancing sample efficiency by generating additional training data.

- Paper: "When to Trust Your Model: Model-Based Policy Optimization" (2020) by Michael Janner et al.
- Significance: MBPO bridges the gap between model-based and model-free methods by using short model-based rollouts to improve the performance and efficiency of policy optimization.
- Approach: Performs short rollouts using a learned dynamics model to generate additional training data, which is then used to optimize the policy with model-free methods.

MuZero: Combines model-based and model-free RL, learning a model of the environment and using it for planning.

- Achievement: Builds its own understanding of the game. It can imagine for itself what the game will look like if one performs such or such actions without prior knowledge of the rules.
- Paper: "Mastering Atari, Go, Chess and Shogi by Planning with a Learned Model" (2020) by Julian Schrittwieser et al.
- Significance: Achieved state-of-the-art performance in complex games, showing the power of combining learned models with planning.
- Method: Learns a model of environment (model-based) + neural networks for policy optimization and value estimation (model-free) + MCTS for planning (model-based)
- Resources: Read more + source, Blog, Youtube

Hierarchical

Hierarchical reinforcement learning involves breaking down tasks into a hierarchy of sub-tasks to improve learning efficiency and performance.

Strategic Attentive Writer for Learning Macro-Actions (STRAW): Introduces a method to learn and execute macro-actions for efficient RL

- Paper: "Strategic Attentive Writer for Learning Macro-Actions" (2016) by Vezhnevets et al.
- Significance: Enhances efficiency in complex environments by learning temporally extended actions.
- Method: Uses a hierarchical structure where a high-level controller outputs a sequence of macro-actions, each consisting of multiple primitive actions.

FeUdal Networks (Feudal Networks): Implements a hierarchical approach by decomposing tasks into multiple levels.

- Paper: "FeUdal Networks for Hierarchical Reinforcement Learning" (2017) by Vezhnevets et al.
- Significance: Allows more efficient learning and exploration by leveraging hierarchical structures.
- Method: Employs a manager network to set goals in an abstract space and a worker network to achieve these goals, effectively breaking down complex tasks.

Data-Efficient Hierarchical Reinforcement Learning (HIRO): Demonstrates improved data efficiency and performance using hierarchical structures.

- Paper: "Data-Efficient Hierarchical Reinforcement Learning" (2018) by Nachum et al.
- Significance: Facilitates learning in sparse reward environments by leveraging hierarchical decomposition.
- Method: Utilizes a high-level policy that sets sub-goals for a low-level policy to achieve, enhancing data efficiency and performance.

Meta-RL

Meta reinforcement learning enables agents to learn new tasks quickly by leveraging experience from previous tasks.

RL^2: Fast Reinforcement Learning via Slow Reinforcement Learning (RL^2)

- Paper: "RL^2: Fast Reinforcement Learning via Slow Reinforcement Learning" (2016) by Duan et al.
- Overview: Proposes a meta-learning approach to enable rapid adaptation to new tasks.

- Significance: Enhances the ability to quickly adapt to new environments by leveraging past experiences.
- Method: Uses a recurrent neural network to encode the agent's past experiences, enabling rapid adaptation to new environments.

Learning to Reinforcement Learn: Focuses on training a meta-learner to generalize across different RL tasks.

- Paper: "Learning to Reinforcement Learn" (2016) by Wang et al.
- Significance: Improves learning efficiency and performance by transferring knowledge across tasks.
- Method: Employs a meta-learner that optimizes the learning process itself, providing a framework for transferring knowledge across tasks.

Model-Agnostic Meta-Learning (MAML): Offers a general approach to meta-learning for quick adaptation to new tasks.

- Paper: "Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks" (2017) by Finn et al.
- Significance: Facilitates rapid learning across various tasks by optimizing for initial parameters that can be quickly adapted.
- Method: Optimizes for a set of initial parameters that can be rapidly adapted to new tasks with minimal updates.

A Simple Neural Attentive Meta-Learner (SNAIL): Introduces a neural network architecture designed for meta-learning.

- Paper: "A Simple Neural Attentive Meta-Learner" (2018) by Mishra et al.
- Significance: Leverages attention mechanisms to improve learning speed and accuracy.
- Method: Combines temporal convolutions with attention mechanisms to effectively process and learn from past experiences.

Memory

Memory-based methods in reinforcement learning use past experiences to make decisions, constructing a working memory.

Model-Free Episodic Control (MFEC): Proposes a memory-based approach to enable faster learning by using episodic memories.

- Paper: "Model-Free Episodic Control" (2016) by Blundell et al.
- Significance: Bypasses trial-and-error learning by leveraging past experiences.
- Method: Stores and retrieves past experiences to guide decision-making, facilitating efficient learning.

Neural Episodic Control (NEC): Enhances model-free episodic control by integrating it with deep neural networks.

- Paper: "Neural Episodic Control" (2017) by Pritzel et al.
- Significance: Improves scalability and performance by using a neural dictionary to store and retrieve episodic memories.
- Method: Uses a differentiable neural dictionary to store and retrieve episodic memories, enabling efficient learning.

Neural Map: Structured Memory for Deep Reinforcement Learning (Neural Map): Introduces a structured memory architecture to navigate and learn in complex environments.

- Paper: "Neural Map: Structured Memory for Deep Reinforcement Learning" (2017) by Parisotto and Salakhutdinov.
- Significance: Enhances the agent's ability to encode and retrieve information about the environment.
- Method: Combines a neural network with a spatially organized memory to encode and retrieve information about the environment.

Unsupervised Predictive Memory in a Goal-Directed Agent (MERLIN): Proposes a method for unsupervised learning of predictive memory representations.

- Paper: "Unsupervised Predictive Memory in a Goal-Directed Agent" (2018) by Wayne et al.
- Significance: Improves goal-directed behavior by developing memory representations that predict future states.
- Method: Utilizes a combination of unsupervised learning and reinforcement learning to develop memory representations that predict future states.

Relational Recurrent Neural Networks (RMC): Introduces a neural network architecture that captures relational information between elements in the environment.

- Paper: "Relational Recurrent Neural Networks" (2018) by Santoro et al.

- Significance: Enhances learning and generalization by modeling relational dependencies.
- Method: Combines recurrent neural networks with attention mechanisms to model relational dependencies, improving decision-making.

Theoretical Foundations

Markov Decision Processes (MDP)

Bellman Equation

Exploration vs. Exploitation

Applications

Hybrid

AlphaGo: Combines supervised learning from human games with reinforcement learning, using MCTS for move evaluation and learning policy and value networks.

- Achievement: Defeated Go champion Lee Sedol in 2015 using Monte Carlo Tree Search (MCTS) combined with neural networks trained on human games.
- Paper: "Mastering the game of Go with deep neural networks and tree search" (2016) by David Silver et al.
- Significance: AlphaGo achieved a historic milestone by defeating human champions in Go, showcasing the power of combining deep learning and tree search.
- Method: Uses a trained neural network on human games to guide MCTS
- Resources: Blog

AlphaZero: Combines deep neural networks with MCTS for learning to play board games from scratch.

- Achievement: Learned solely through self-play without human game histories.
- Paper: "Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm" (2017) by Silver et al.

- Significance: Demonstrated superhuman performance in Chess, Shogi, and Go by learning solely through self-play.
- Method: Uses neural networks for policy optimization and value estimation which guides MCTS, model-based planning (MCTS). The neural networks guide the MCTS, and the MCTS, in turn, helps refine the neural network predictions.
- Resources: Alpha Zero General (AZG), Lc0 (Leela Chess Zero), ELF OpenGo, Blog

MuZero: Combines model-based and model-free RL, learning a model of the environment and using it for planning.

- Achievement: Builds its own understanding of the game. It can imagine for itself what the game will look like if one performs such or such actions without prior knowledge of the rules.
- Paper: "Mastering Atari, Go, Chess and Shogi by Planning with a Learned Model" (2020) by Julian Schrittwieser et al.
- Significance: Achieved state-of-the-art performance in complex games, showing the power of combining learned models with planning.
- Method: Learns a model of environment (model-based) + neural networks for policy optimization and value estimation (model-free) + MCTS for planning (model-based)
- Resources: Read more + source, Blog, Youtube

OpenAI: Competitive Self-Play: Studies the development of advanced strategies and skills through self-play.

- Achievement: Demonstrated that AI agents can develop complex strategies without human intervention.
- Paper: "Competitive Self-Play and the Emergence of Complex Behaviors" (2019) by OpenAI et al.
- Significance: Highlighted the effectiveness of self-play in training AI agents to achieve superhuman performance in various tasks.
- Method: Self-play reinforcement learning with evolving challenges and adaptive learning.
- Resources: Blog

Hybrid + Transfer Learning

AlphaDev: Optimizes computer algorithms, focusing on assembly language for enhanced performance.

- Achievement: Enhanced the efficiency of fundamental computing tasks like sorting.
- Paper: “Faster sorting algorithms discovered using deep reinforcement learning” (2022) by Daniel J. Mankowitz et al.
- Significance: Transferred learning from AlphaZero, from games to scientific challenges, simulations to real world applications.
- Method: Explores and optimizes assembly instructions, uncovering new algorithms that reduce computational steps and execution time.
- Resources: Blog

Multi-Agent

AlphaStar: Applies deep reinforcement learning to real-time strategy games like StarCraft II.

- Achievement: reached Grandmaster level in StarCraft II.
- Paper: "Grandmaster level in StarCraft II using multi-agent reinforcement learning" (2019) by Oriol Vinyals et al..
- Significance: Demonstrated the application of RL in real-time, complex, and strategic environments.
- Method: Combines neural networks with RL algorithms to learn strategies and make decisions in real-time.
- Resources: Blog, Github, Youtube
-

OpenAI: OpenAI Five: A team of AI agents trained to play Dota 2 at a competitive level.

- Achievement: Defeated professional human teams in Dota 2.
- Paper: "Dota 2 with Large Scale Deep Reinforcement Learning" (2019) by OpenAI et al.
- Significance: Showcased the potential of RL in highly complex, multi-agent environments and highlighted the scalability of RL algorithms.
- Method: Large-scale deep reinforcement learning with self-play and multi-agent coordination.
- Resources: Blog

OpenAI: Emergent Tool Use: Explores the ability of agents to develop and use tools in simulated environments.

- Achievement: Demonstrated that AI agents can develop sophisticated tool-use strategies in virtual environments.
- Paper: "Emergent Tool Use from Multi-Agent Autocurricula" (2020) by Bowen Baker et al.
- Significance: Showed the potential for AI to autonomously develop complex behaviors and strategies, highlighting the adaptability of RL in novel tasks.
- Method: Multi-agent reinforcement learning with evolving goals and self-play.
- Resources: Blog

OpenAI: Learning Dexterity: Focuses on training robotic hands to manipulate objects with high precision.

- Achievement: Achieved significant progress in robotic hand dexterity using RL.
- Paper: "Solving Rubik's Cube with a Robot Hand" (2019) by OpenAI et al.
- Significance: Demonstrated the application of RL in robotic manipulation tasks, paving the way for advancements in autonomous robotics.
- Method: Deep reinforcement learning combined with domain randomization to train robotic hands in simulation and transfer skills to real-world tasks.
- Resources: Blog

Read more

Papers in Deep RL

Simulation and Environments

OpenAI Gym: A toolkit for developing environments and comparing RL algorithms.

Stable Baseline 3 Zoo: A toolkit for fine-tuning and comparing RL algorithms.

Things to do:

- Recreating algorithms
- Recreating Implementations
- Learning Libraries
- Custom additions
- Projects