Multiagent Reinforcement Learning and Heterogeneity

Bachelor's Thesis of

Christian Burmeister

at the Department of Informatics
Institute for Anthropomatics and Robotics (IAR)
Autonomous Learning Robots (ALR)

Reviewer:	Prof. A
Second reviewer:	Prof. B
Advisor:	M.Sc. C
Second advisor:	M.Sc. D

xx. Month 2021 – xx. Month 2021

I declare that I have developed and written the enclosed thesis completely by myself, and have not used sources or means without declaration in the text.

PLACE, DATE

. . . .

(Christian Burmeister)

Abstract

asdf

Zusammenfassung

Contents

Abstract i

Zusammenfassung ii

1. Motivation 1

1.1. Multiagent Systems 1
1.2. Spacing and indentation 1
1.3. Example: Citation 2
1.4. Example: Figures 2
1.5. Example: Tables 2
1.6. Example: Formula 2

2. Information to sort 3

2.1. MARL - A Comprehensive Survey of Multiagent Reinforcement Learning - 2008 3
2.2. !MAS - An Introduction to Multi-Agent Systems - 2010 4
2.2.1. Classification of MAS 4
2.3. Artificial Intelligence - A modern Approach 8
2.3.1. Agents and Environments 8
2.3.2. Rational Agent 8
2.3.3. Nature of Environments 9
2.3.4. Structure of Agents 10
2.3.5. Multiagent Planning 11
2.3.6. Game Theory 13
2.3.7. Mechanism Design for Multiple Agents 13
2.3.8. Adversarial Search 13
2.3.9. Probabilistic Reasoning over Time 13
2.3.10. Reinforcement Learning 13
2.3.11. Planning Uncertain Movements (Potential Fields) 13
2.4. Ant Colony Optimization 13
2.4.1. Wikipedia Article 13
2.5. Reinforcement Learning 14
2.5.1. Algorithmia Blog 14
2.5.2. Freecodecamp 14
2.5.3. Deep RL Bootcamp 19
2.5.4. RL Lectures from Deepmind 20
2.6. ALR-Wiki 22
2.7. Books 23
2.7.1. Deep Learning Book 23

2.7.2. Reinforcement Learning - An Introduction 23
2.8. UNSORTED 23
2.9. References & Papers 23
2.9.1. Ant Colony Optimization (ACO) 23
2.9.2. Graph Neural Networks 23
2.9.3. Reinforcement Learning 26
2.9.4. Multiagent Systems (MAS) 26
2.9.5. Multi Agent Reinforcement Learning (MARL) 27
2.9.6. GNN for Multi Agent Reinforcement Learning (GNNMARL) 29
2.9.7. Applications 29

3. Fundamentals **30**

4. Problem and Approaches **33**
4.1. Definition of the Problemdomain 33

5. Project **34**

6. Related Works **35**

7. Conclusion **36**

Bibliography **37**

A. Appendix **38**
A.1. First Appendix Section 38

List of Figures

1.1. SDQ logo 2
A.1. A figure 38

List of Tables

1.1. A table 2

1. Motivation

1.1. Multiagent Systems

Use Cases:

- Multiagent systems can be used in game theory and financing
- Reconnaissance robots covering a wide area. Communication not always possible.
- Smart Grid for Electricity, Power allocation, energy management.
- Flow Line Systems
- Stock markets
- Competitive pricing strategies
- Load Balancing.
- Network Systems (IoT).
- Traffic Light Control
- Autonomous Driving, Vehicular networks
- Automating turbulence modelling (aircraft design, weather forecasting, climate prediction).
- Control Systems for industrial processes.
- Intrusion Detection
- resource allocation for UAV Networks
- Large Scale City Traffic (Cityflow).
- Spectrum Management of cognitive radio using MARL.

Aspects:

- Ant-Colony-Optimization, which can be used for learning.
- Emergent Behavior.
- Swarm Intelligence.
- multi-agent reinforcement learning.
- multi-agent learning.
- game theory

This is the SDQ thesis template. For more information on the formatting of theses at SDQ, please refer to `https://sdqweb.ipd.kit.edu/wiki/Ausarbeitungshinweise` or to your advisor.

1.2. Spacing and indentation

To separate parts of text in LaTeX, please use two line breaks. They will then be set with correct indentation. Do *not* use:

- `\\`
- `\parskip`

Figure 1.1.: SDQ logo

abc	def
ghi	jkl
123	456
789	0AB

Table 1.1.: A table

- `\vskip`

or other commands to manually insert spaces, since they break the layout of this template.

1.3. Example: Citation

A citation: [1]

1.4. Example: Figures

A reference: The SDQ logo is displayed in Figure 1.1. (Use `\autoref{}` for easy referencing.)

1.5. Example: Tables

The `booktabs` package offers nicely typeset tables, as in Table 1.1.

1.6. Example: Formula

One of the nice things about the Linux Libertine font is that it comes with a math mode package.

$$f(x) = \Omega(g(x)) \ (x \to \infty) \quad \Leftrightarrow \quad \limsup_{x \to \infty} \left| \frac{f(x)}{g(x)} \right| > 0$$

2. Information to sort

2.1. MARL - A Comprehensive Survey of Multiagent Reinforcement Learning - 2008

MARL - A Comprehensive Survey of Multiagent Reinforcement Learning - 2008
Benefits

- can be parallelized.
- can use experience sharing via communication, or with a teacher-learner relationship.
- Failure of one agent can be covered by other agents.
- insertion of new agents => scaleable.
- MARL Complexity: Exponential in number of agents.
- exploration (new knowledge) - exploitation (current knowledge) - Tradeoff.
- They explore about the environment and other agents.
- need for coordination.

Application Domains

- simulation better than real-life (better scalability and robustness).
- Distributed Control: for controlling processes (for larger industry plants).
  - avenue for future work.
  - used for traffic, power or sensory networks.
  - could also be used for pendulum systems.
- Robotic Teams (Multirobot):
  - simulated 2D space.
  - navigation: Reach a goal with obstacles. Area sweeping (retrival of objects (also cooperative)).
  - pursuit: Capture a prey robot.
- Automated Trading: Exchange goods on electronic markets with negotiation and auctions.
- Resource Management: Cooperatie team to manage resources or as clients. (routing, load balancing).

Practicallity and Future works

- Scalability Problem: Q-functions do not scale well with the size of the state-action space.
  - Approximation needed: for discrete large state-action spaces, for continous states and discrete actions or continious state and action.
  - Heuristic in nature and only work in a narrow set of problems.
  - Could use theoretical results on single-agent approximate RL.
  - also could use discovery and exploitation of the decentralized, modular structure of the multiagent task.
- MARL without prior knowledge is very slow.
  - Need humans to teach the agent.

  - shaping: first simple task then scale them.
  - could use reflex behavior.
  - Knowledge about the task structure.
- Incomplete, uncertain state measurements could be handled with partiall observability techniques (Markov decision process).
- Multiagent Goals needs a stable learning process for the environment and an adaption for the dynamics of other agents.
- using game-theory-based analysis to apply to the dynamics of the environment.

2.2. !MAS - An Introduction to Multi-Agent Systems - 2010

MAS - An Introduction to Multi-Agent Systems - 2010
Benefits of using MAS in large systems

- Increase in the speed and efficiency of the operation due to parallel computation and asynchronous operation.
- Graceful degradation whone one or more of the agent fail, thus increasing realibility and robustness of the system.
- Scalability and flexibility - Agents can be added as and when necessary.
- Cost Reduction: Individual agents cost much less than a centralized architecture
- Reusability: Agents with a modular structure can be easily replaced in other systems or be upgraded more easily than a monolithic sysetm.

Challenges of using MAS in large systems

- environment: An agents action modify its own environment but also that of its neighbours. therefore they need to predict the action of the other agents so that they can reach a goal. This can be an unstable system. Environment dynamic: Is the effect caused by other agents or by the variation in the environment?
- perception: limited sensing range => each agent only has partial observability for the environment. Therefore the decisions reached might be sub-optimal.
- Abstraction: ???
- conflict resolution: lack of global view => conflict. therefore information on constraints, action preferences and goal prioritoes must be shared between agents. When to communicate what to which agent?
- Inference: Single-Agent: State-Action-Space can be mapped with trial and error. Multiagent: each agent may or may not interact with each other. If they are heterogenous, they might even compete and have different goals. You need a fitting inference machanism

2.2.1. Classification of MAS

Internal Architecture

- homogeneous: all agents have the same internal architecture (Local Goals, Sensor Capabilities, Internal states, Inference Mechanism and Possible Actions). In a typical distributed environment, overlap of sensory inputs is rarely present

- Heterogeneous: agents may differ in ability, structure and functionality. Because of the dynamics and location the actions chosen might differ between agents. their local goals may contradict the objective of other agents.

Agent Organization

- hierarchical: typical: tree-structure. At different heights, different levels of autonomy. data from lower levels flow upwards. Control signal flows from high to low in the hierarchy.
  - simple: the decision making authority is a single agent of highest level. BUT: single point of Failure
  - uniform: authority is distributed among the various agents, for better efficiency, fault tolerance, graceful degradation. Decisions made by agent with appropriate information. (MAS - TrafficControl - Neural Networks for Continuous Online Learning and Control - 2006)
- holonic: fractal structure of several holons. Self-repeating. Used for large organizational behaviours in manufacturing and business.
  - An agent that appears as a single entity might be composed of many sub-agents. They are not predetermined, but form through commitments.
  - Each holon has a head agent that communicates with the environment or with other agents in the environment. It is selected either randomly, through a rotation policy, or selected by resource availability, communicaton capability.
  - Holons can be nested to form Superholons.
  - compare to tree: in Holons cross tree interactions and overlapping of holons is allowed.
  - pro: abstraction good degree of freedom, good agent autonomy.
  - contra: abstraction makes it difficult for other agents to predict the resulting actions of the holon.
- coalitions: group of agents come together for a short time to increase utility or performance of the individual agents in a group. they cease to exist when the performance goal is achieved.
  - coalition may have either a flat or a hierarchical architecture.
  - It may have an leading agent to act as a representative.
  - overlap is allowed. this increased complexity of computation of the negotiation strategy.
  - You can have one coaltion with all agents => maximum performance of system. Impractical due to restraints on communication and resources.
  - minimize amount of colations: because of the cost of creating and dissolving a colation group.
- teams: agents work together to increase the overall performance of the group, rather than working as individual agents.
  - their interactions can be arbitrary and the goals and roals can vary with the performance of the group.
  - large team size is not beneficial under all conditions. some compromises must be made.
  - large teams offer a better visibility of the environment. but is slower computation wise. Learning-Performance Tradeoff.

    - computation cost usually much greater than coalitions.

Communication

- local communication: agents directly communicate similar to message passing. there is no place to store information. creates distributed architecture. used in: (25),(37),(38).
- blackboards: a group of agents share a data repository which is provided for efficient storage.
    - can hold design data and control knowledge, accessable by the agents.
    - control shell: notfies the agent when relevant data is available.
    - single point of failure.
- agent communication language (ACL): common framework for interaction and information sharing. (40).
    - procedural approach: modelled as a sharing of the precedural directives. Shared how an agent does a specific task or the entire working of the agent itself. Script Languages often used. Disadvantage: necessitiy of providing information on the recipient agent, which is in most cases partially known. Also how to merge the scripts into one executable. Not preferred method.
    - declarative approach: sharing of statements for definitions, assumptions assertions, axioms etc. Short declarative statements as length increases probability of information corruption. Example: ARPA knowledge sharing effort.
    - Best known inner languages: Knowledge Interchange Format. Information exchange is implicitly embedded in KIF. But the package size grows with the increase in embedded information. Solution: High-level Languages like KQML (Knowledge QUery and Manipulation Language)

Decision making in Multi-Agent Systems

- undercainty: effects of a specific actions on the environment and dynamics because of the other agents.
- Methodology to try and find a joint action or equilibrium point which maximizes the reward of every agent.
- Typically modelled with game theory method. Strategic games:
    - a set of players (agents)
    - Foreach player, there is a set of actions
    - Foreach player, the prefeernces over a set of actions profiles
    - payoff with the combination of action, a joint-action, that is assumed to be predefined.
    - all actions are observable forall agents.
    - make the assumption that all participating agents are rational.
- Nash equilibrium: for a payoff matrix: An action profile (joint-action), where no player can do better by choosing one of the actions differently, given that the other player chose a specific action.
- there might be multiple nash equilibrium, so that there is no dominant solution. Here the coordination of MAS is needed to find a solution.
- Iterated Elimination Method: Strongly dominated actions are iteratively eliminated. This fails if there are no strictly dominated actions available.

Coordination

- agents work in parallel, therefore they need to be coordinated or synchronize the actions to ensure stability of the system.

- other reasons: prevent chaos, meet global constraints, utilize distributed resources, prevent conflicts, improve efficiency.
- achievable with constraints on the joint actions or by using informatil collated from neighbouring agents. Used to find the equilibrium action.
- payoff matrix necessary might be difficult to determine. It increases expenentially in the number of agents and action choices.
- dividing the game into subgames: roles (permitted actions is reduced, good for distributed coordination or centralized coordination)
- Coordination via Protocol.
  - negotioation to arrive an approdiate solutions.
  - Agents assume the role of manager (divide the problem) and contractor (who deals with the subproblems).
  - The manager and contractor are working in a bidding system.
  - Example: FIPA model
  - disadvantage: assumption of the existence of an cooperative agent. It is very communication intensive
- Coordination via Graphs: Problem is subdivided into easer problems. Assume the payoffs can be linear combinated from the local payoffs of the sub-games. THen just eliminate agents to find the optimal joint.
- Can also use belif models. Internal models of an agent on how he believes the environment works (needs to differentiate between environment and effects of other agents).

Learning

- active learning: analysing the observations to creat a belief or internal model of the corresponding situated agent's environment.
  - can be performed by using a deductive, inductive or probabilistic reasoning approach.
  - deductive: inference to explain an instance or state-action sequence using his knowledge. It is deduced or inferred from the original knowledge it is nothing new. It could form new parts of the knowledge base. uncertainty is usually disregarded (not good for real-time)
  - inductive: learning from observations of state-action pair. Good when environment can be presented in terms of some generalized statements. they use the correlation between observations and the action space.
  - probabilistic: assumption: knowledge base or belief model can be represented as probabilities of occurrence of events. observations of the environment is used to predict the internal state of the agent. Good example: Bayesian learning. Difficult for MAS, as the joint probability scales poorly in the number of agents.
- reactive learning: updating belief without having the actual knowledge of what needs to be learnt.
  - useful when the underlying model of the agent or the environment is not known clearly and are black boxes.
  - can be ssen in agents which utilize connections systems such as NN.
  - can use reactive multi-agent feed forward neural networks.
  - they depend on the application domain and are therefore rarely employed in real world scenarios.
- learning based on consequences:

  - learning methods based on evaluation of the goodness of selected action. like in reinforcement learning.
  - programming the agents using reward and punishment scalar signals without specifying how the task is to be achieved.
  - learnt through trial and error and interaction with the environment.
  - usually used when action space is small and descret. Recent developments allow them to work in continious and large state-action space scenarios.
  - An agent is usally represented as a Markov Decision Process.
  - Expectaation operator optmal policy is the argmax of the Q-value, which uses the bellman equation. Bellman equation is solved iteratively.
  - The solution is referred to as q-learning method.
  - For MAS the reinforcement learning method has the problem of combinatorial explosion in the state-action pairs.
  - The information must be passed between the agents for effective learning.

2.3. Artificial Intelligence - A modern Approach

2.3.1. Agents and Environments

p.34

- **agent**: anything that perceives its **environment** through **sensors** and acting upon that environment using **actuators**.
- **percept**: agent's perceptual inputs at any given instance. Percept sequence is a complete history of perception.
- agents choice of action decided upon the history of perception, but not anything it has not perceived.
- its behavior is described by the **agent function**, which is internally implemented by the **agent program**.

2.3.2. Rational Agent

p.36

- **rational agent**: it does the correct thing. Correctness is determined by a performance measure, which is determined by the changed environment states.
- design **performance measures** according to what one actually wants in the environment, rather than according to how one thinks the agent should behave.
- rational depends on:
  - the performance measure that defines the criterion of success
  - the agent's prior knowledge of the environment.
  - The actions that the agent can perform.
  - The agent's percept sequence of data.
- depending on the measures the agent might be rational or not.

- an **omniscient agent** knows the actual outcome of its actions and can act accordingly, but this is impossible in reality.
- rationality maximizes expected performance, while perfection (omniscient) maximizes actual performance.
- agents can do actions in order to modify future percepts, called **information gathering, or exploration**.
- rational agents learn as much as possible from what it perceives.
- his knowledge can be augmented and modified as it gains experience.
- if the agent relies on the prior knowledge of its designer rather than on its own percepts, we say that the agent lacks **autonomy**.
- it should learn what it can to compensate for partial or incorrect prior knowledge.
- give it some initial knowledge and the ability to learn, so it will become independent of its prior knowledge.

2.3.3. Nature of Environments

p.40

- **task environments**: the "problems" to which rational agents are the "solutions".
- Describe the task environment in the following aspects P(Performance measure), E(Environment), A(Actuators), S(Sensors).
- **fully observable**: the agent's sensors give it access to the complete state of the environment. All aspects that are relevant to the choice of actions
- **partially observable**: otherwise. Because of missing sensors or noise.
- no sensors: unobservable
- single-agent environments and multi-agent environments.
- multi-agent can be either competitive (chess) or cooperative (avoiding collisions maximizes performance).
- **communication** emerges as a rational behavior in multiagent environments.
- randomized behavior is rational because it avoids the pitfalls of predictability.
- **Deterministic**: next state of environment is completely determined by the current state and the action executed by the agent, otherwise it is **stochastic**.
- you can ignore uncertainty that arises purely from the actions of other agents in a multiagent environment.
- If the environment is partial observable, it could appear to be stochastic, which implies quantifiable outcomes in terms of probabilities.
- an environment is **uncertain** if it is not fully observable or not deterministic.
- **episodic**: the agent's experience is divided into atomic episodes. In each the agent receives a percept and performs a single episode. The next episode does not depend on the actions taken in previous episodes, otherwise it is **sequential**.
- When the environment can change while the agent is deliberating, then the environment is **dynamic** for that agent otherwise it is **static**.
- if the environment itself does not change with the passage of time but the agent's performance score does, then we say the environment is **semi dynamic**.
- **discrete/continuous** applies to the state of the environment, to the way time is handled, and to the percepts and actions of the agents.

- **known vs. unknown**: refers to the agent's state of knowledge about the "laws of physics" of the environment. Known environment, the outcomes for all actions are given, otherwise the agent needs to learn how it works. An environment can be known, but partially observable (solitaire: I know the rules but still unable to see the cards that have not yet been turned over)
- hardest case: partially observable, multiagent, stochastic, sequential, dynamic, continuous, and unknown
- **environment class**: multiple environment scenarios to train it for multiple situations.
- you can create an **environment generator**, that selects environments in which to run the agent.

2.3.4. Structure of Agents

p.46

- agent = architecture (computing device) + program (agent program).
- agent programs take the current percept as input and return an action to the actuators.
- agent program takes the current percept, agent function which takes the entire percept history.
- **table driven agent**: Uses a table of actions indexed by percept sequences. This table grows way to fast and is therefore not practical.

Simple reflex agents:

- **simple reflex agents**: Select the actions on the basis of the current percept, ignoring the rest of the history.
- **condition-action-rule**: these agents create actions in a specific condition (if-then). These connections can be seen as reflexes.
- uses an **interpret-input** function as well as a **rule-match** function.
- they need the environment to be fully observable. They could run into infinite loops.
- you can mitigate this by using randomization for the actions. Which is non-rational for single agent environments.

Model-based reflex agents:

- keep track of the part of the world an agent cannot see now. It maintains some sort of **internal state** that depends on the percept history.
- agents needs to know how the world evolves independently of the agent and how the agent's own actions affect the world.
- with this it creates a **model** of the world hence it is called model-based agent.
- it needs to update this state given sensor data.
- this model is a **best guess** and does not determine the entire current state of the environment exactly.

Goal-based agents:

- an agent needs some sort of **goal information** that describes situations that are desirable. This can also be combined with the model.
- Usually agents need to do multiple actions to fulfill a goal which requires **search** and **planning**.
- this also involves consideration of the future.

- the goal-based agent's behavior can be easily changed to go to a different destination by using a goal where a reflex agent needs completely now rules.

Utility-based agents:

- goals provide a crude binary distinction between good and bad states.
- use an internal **utility function** to create a performance measure.
- if the external performance measure and the internal utility function agree, the agent will act rationally.
- if you have conflicting goals the utility function can specify the appropriate **tradeoff**.
- if multiple goals cannot be achieved with certainty, utility provides a way to determine the **likelihood** of success.
- a rational utility-based agent chooses the action that **maximizes the expected utility**.
- any rational agent must behave as if it possesses a utility function whose expected value it tries to maximize.
- a utility-based agent must model and keep track of its environment.

Learning Agents:

- it allows the agent to operate in initially unknown environments and to become more competent than its initial knowledge alone might allow.
- 4 conceptual components: **learning element** (responsible for improvements), **performance element** (select external action), **critic** (gives feedback to change the learning element), **problem generator** (suggesting actions that lead to new and informative experiences).
- critic tells the learning element how well the agent is doing given a performance standard. It tells the agent which percepts are good and which are bad.
- problem generator allows for exploration and suboptimal actions to discover better actions in the long run.
- learning element: simplest case: learning directly from the percept sequence.
- the **performance standard** distinguishes part of the incoming percept as a reward or penalty that provides direct feedback on the quality of the agent's behavior.

How the components of agent programs work:

- **atomic representation**: Each state of the world is indivisible. Algorithms like search and game-playing, Hidden Markov models and Markov decision models work like this.
- **factored representation**: splits up each state of a fixed set of variables or attributes which each can have a value. Used in constraint satisfaction algorithms, propositional logic, planning, Bayesian networks.
- **structured representation**: here the different states have connections to each other. Used in relational databases, first-order logic, first-order probability models, knowledge-based learning and natural language understanding.
- more complex representations are more **expressive** and can capture everything more concise.

2.3.5. Multiagent Planning

p.425

- each agent tries to achieve is own goals with the help or hindrance of others

- wide degree of problems with various degrees of **decomposition of the monolithic agent**.
- multiple concurrent effectors => **multieffector planning** (like type and speaking at the same time).
- effectors are physically decoupled => **multibody planning**.
- if relevant sensor information foreach body can be pooled centrally or in each body like single-agent problem.
- When communication constraint does not allow that: **decentralized planning problem**. planning phase is centralized, but execution phase is at least partially decoupled.
- single entity is doing the planning: one goal, that every body shares.
- When bodies do their own planning, they may share identical goals.
- **multibody**: centralized planning and execution send to each.
- **multiagent**: decentralized local planning, with coordination needed so they do not do the same thing.
- Usage of **incentives** (like salaries) so that goals of the central-planner and the individual align.

Multiple simultaneous actions:

- **correct plan**: if executed by the actors, achieves the goal. Though multiagent might not agree to execute any particular plan.
- **joint action**: An Action for each actor defined => joint planning problem with branching factor b n̂ (b = number of choices).
- if the actors are **loosely coupled** you can describe the system so that the problem complexity only scales linearly.
- standard approach: pretend the problems are completely decoupled and then fix up the interactions.
- **concurrent action list**: which actions must or most not be executed concurrently. (only one at a time)

Multiple agents: cooperation and coordination

- each agent makes its own plan. Assume goals and knowledge base are shared.
- They **might choose different plans** and therefore collectively not achieve the common goal.
- **convention**: A constraint on the selection of joint plans. (cars: do not collide is achieved by “stay on the right side of the road”).
- widespread conventions: social laws.
- absence of convention: use communication to achieve common knowledge of a feasible joint plan.
- The agents can try to **recognize the plan other agents want to execute** and therefore use plan recognition to find the correct plan. This only works if it is unambiguously.
- an **ant** chooses its role according to the local conditions it observes.
- ants have a convention on the importance of roles.
- ants have some learning mechanism: a colony learns to make more successful and prudent actions over the course of its decades-long life, even though individual ants live only about a year.
- Another Example: **Boid**

- If all the boids execute their policies, the flock inhibits the emergent behavior of flying as a pseudorigid body with roughly constant density that does not disperse over time.
- **most difficult multiagent** problems involve both cooperation with members of one's own team and competition against members of opposing teams, all without centralized control.

2.3.6. Game Theory

p.666

2.3.7. Mechanism Design for Multiple Agents

p.679

2.3.8. Adversarial Search

p.182

2.3.9. Probabilistic Reasoning over Time

p.587

2.3.10. Reinforcement Learning

p.830

2.3.11. Planning Uncertain Movements (Potential Fields)

p.993

2.4. Ant Colony Optimization

2.4.1. Wikipedia Article

Ant Colony Optimization Algorithm, Wikipedia

- is used for solving computational problems which can be reduced to finding good paths through graphs.
- artificial ants locate optimal soluions by moving through a parameter space represneting all possible solutions.
- they record their positions and the quality of their solutions for later iterations to find better solutions (pheromones).

2.5. Reinforcement Learning

2.5.1. Algorithmia Blog

Introduction to Reinforcement Learning

- **Policy Learning**: Policy is a function: (state) -> (action). (if you approach an enemy and the enemy is stronger than you, turn backwards).
- Can use Neural Nets to approximate complicated functions
- **Q-Learning / Value Functions**: (state, action) -> (value). It also adds in all of the potential future values that this action might bring you.
- Approximate Q-Learning Functions with Neural Nets: DQN (RL - DQN - Human-level control through deep reinforcement - 2015)
- Newer way to approximate Q-Functions: A3C (Tutorial, RL - A3C - Asynchronous Methods for Deep Reinforcement Learning - 2016)
- **Challenges**:
  - Reinforcement Learning requires a ton of training data, that other algorithms can get to more efficiently.
  - RL is a general algorithm. If the problem has a domain-specific solution that might work better than RL. Tradeoff between scope and intensity.
  - Most pressing Issue: Design of the reward function. it could get stuck in local optima

2.5.2. Freecodecamp

An introduction to Reinforcement Learning

- $State\ S_t, Reward\ R_t, Action\ A_t$
- **Reward Hypothesis**: All goals can be described by the maximization of the expected cumulative reward: $G_t = \sum_{k=0}^{T} R_{t+k+1}$
- But as earlier rewards are more probable to happen you need to increase their perceived value. Therefore you need a factor $0 \le \gamma < 1$.
- Large γ, Agent cares about long-term reward. Small γ, Agent cares more about short term reward.
- **Discounted Accumulative Rewards (return)**: $G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1},\ where\ \gamma \in [0, 1)$
- **Episodic tasks**: starting point and an ending point (terminal state), this creates an episode.
- **Continuous Tasks**: Tasks that continue forever (no terminal state).
- Learning Methods: Collecting the rewards at the end of the episode for the feature (Monte-Carlo), or Estimate the rewards at each step (Temporal Difference Learning)
- **Monte-Carlo**: $V(S_t) \leftarrow V(S_t) + \alpha[G_t - V(S_t)]$. Left-Side: $V(S_t)$ Maximum expected Future, Right-Side: $V(S_t)$ Former estimation of maximum expected future. α: learning rate.
- **TD-Learning**: $V(S_t) \leftarrow V(S_t) + \alpha[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)]$. $R_{t+1} + \gamma V(S_{t+1})$ is the TD-Target. TD-Target is an estimation, by updating it via a one-step target.
- **Exploration/Exploitation Tradeoff**: Exploration (finding more information about the environment), Exploitation (using known information to maximize the reward). The Agent might find better rewards by doing exploration.

- **Value Based RL**: OPtimize the value function V(s), that tells us the maximum expected future reward.
  - The value of each state is the total amount of the reward an agent can expect to accumulate over the future, starting at that state.
  - $v_\pi(s) = \mathbb{E}_\pi[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s]$. The Expected Reward given an State s.
  - The agent takes the state with the biggest expected reward.
- **Policy Based**: optimize the policy function $a = \pi(s)$, without using the value function, a being the action to take, given a state.
  - The policy can either be deterministic, or stochastic $\pi(a|s) = \mathbb{P}[A = a|S = s]$(output is a distribution probebility over actions.)
  - It directly indicates the best action to take for each step.
- **Model Based**: Model the environment. Each environment needs a different model foreach environment.
- Deep Reinforcement Learning: Uses deep neural networks to solve it.

Diving deeper into Reinforcement Learning with Q-Learning

- **Q-learning** is value-based RL.
- **Q(Quality)-Table** gives you foreach action-state pair a value which moves gives the best maximum expected future reward.
- you don't implement a policy, you improve the Q-table to always choose the best action. The values in the table need to be learned.
- Action-Value Function (Q-Function) takes state and action as input and returns the expeced future reward.
- $Q^\pi(s_t, a_t) = \mathbb{E}\ [\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | s_t, a_t]$
- As we explore the environment, the Q-table will give us a better and better approximation by iteratively updating Q(s,a) using the **Bellman Equation**.
- Algorithm process: 1. Initialize Q-Table -> 2. Choose action a -> 3. perform action -> 4. measure reward -> 5. update Q -> goto 2.
  - 1. Initialize: e.g. initialize everything 0
  - 2-3. choose an action. Use the epsilon greedy strategy. $0 \leq \epsilon \leq 1$ defines the exploration rate. It starts of with 1. We start of doing alot of random guesses what actions to choose (exploration). It is like a chance. We reduce the epsilon progressively to do more exploitation of the knowledge we gained.
  - 4-5. update q: We update Q with the Bellman equation (given a new state s' and a reward r): $newQ(s, a) = Q(s, a) + \alpha[\Delta Q(s, a)]$, $\Delta Q(s, a) = R(s, a) + \gamma \max(Q'(s', a')) - Q(s, a)$
  - $\max(Q'(s', a'))$: Maxium expected future reward given the new s' and all possible actions at that new state. The highest Q-value between possible actions from the new state s'.

An introduction to Deep Q-Learning: let's play Doom

- Instead of using a **Q-table**, use a Neural Network that takes a state and **approximates Q-values** for each action based on that state.
- In a videogame states can be associated with frames. you need multiple state inputs (like 4).
- preprocessing is important to reduce the complexity of the states to reduce the computation time needed for training.

- **temporal limitation**: you need multiple frames to percept motion in the environment.
- using convolutional layers with ELU. Use fully connected layers with ELU and one output layer that produces the Q-value estimataion for each action.
- Making more efficient use of observed experience using experience Replay:
  - **Avoid forgetting previous experiences**: given that we use sequential samples from interactions with our environment, the network tends to forget the previous experiences. You could use previous experiences by learning it multiple times.
  - reducing correlation between experiences: every action affects the next state, the sequence of experiences can be highly correlated. If we train in sequential order we might risk the agent bein influenced by it. Two strategies:
  - stop learning while interacting with the environment. Play a little randomly to explore the state space. Then recall these experiences and learn from then, then play again with the updated value function.
  - This way you have better set of examples. This prevents reinforcing the same action over and over.
- $\Delta w = \alpha[(R + \gamma\ max_a \hat{Q}(s', a, w)) - \hat{Q}(s, a, w)]\ \nabla_w\ \hat{Q}(s, a, w)$
- $\Delta w = \alpha * TD - Error * Gradient\ of\ our\ Prediction$

Improvements in Deep Q Learning: Dueling Double DQN, Prioritized Experience Replay, and fixed Q-targets

- Fixed Q-targets:
  - We calculate **TD-Error** (aka the loss), but we don't have any idea of the real TD-target. Bellman equation states that the TD-Target is the reward of taking that action at that state plus the discounted highest Q-value for the next state.
  - But we use the weights for the target and the Q-value and therefore our Q-value and our target value shifts.
  - **Q-Targets**: Using a seperate network with a fixed parameter (w-tilde) for estimating the TD-Target. At every tau step, we copy the parameters from our DQN network to update the target network:
  - $\Delta w = \alpha[(R + \gamma\ max_a \hat{Q}(s', a, \tilde{w})) - \hat{Q}(s, a, w)]\ \nabla_w\ \hat{Q}(s, a, w),\ At\ every \tau step : \tilde{w} \leftarrow w$
- **Double DQN**: Handles the problem of the overestimation of Q-values.
  - TD-Target = Q-target = reward + discounted max-q.
  - How are we sure the best action for the next state ist the action with the highest Q-value, it depends on what actions we tried and what neighbors we explored.
  - In the beginning of the training the max-q value will obviously b noisy and can lead to false positives. Learning will be complicated.
  - Solution: When computing q-target, use two networks to decouple the action selection from the target Q-value generation
  - Use our DQN network to select what is the best action to take for the next state (the action with the highest Q-value). We use our target network to calculate the target Q-value of taking that action at the next state.
  - $argmax_a Q(s', a) = DQN\ choose\ action\ for\ next\ state,\ Q(s', argmax_a Q(s', a)) = Target\ network\ calculates\ the\ qvalue.$
  - $Q(s, a) = r(s, a) + \gamma Q(s', argmax_a Q(s', a))$
  - this helps us reduce the overestimation of q values and helps us train faster and have more stable learning.

- **Dueling DQN (aka DDQN)**: Seperate the estimator into two parts:
  - Q(s,a) can be decomposed as the sum of: V(s): the value of being at that state. A(s,a): the advantage of taking that action at that state (how much better it is to all other actions).
  - With DDQN, we seperate the estimator using two streams one for V(s) and one for A(s,a) and then combine these two streams through a special aggregation layer to get an estimate of Q(s,a). Two streams in the NN.
  - By decoupling the estimation we can learn which states are valuable without having to learn the effect of each action at each state.
  - Being able to calculate V(s) can be useful for state where their actions do not affect the environment in a relevant way.
  - Aggregation: Simply adding both streams will be problemantic for the back propagation, you can force the advantage function estimator to have 0 advantag at the chosen action. To do that, we subtract the average advantage of all actions possible of the state.
  - $Q(s,a;\theta,\alpha,\beta) = V(s;\theta,\beta) + (A(s,a;\theta,\alpha) - \frac{1}{\mathcal{A}}\sum_a' A(s,a';\theta,\alpha))$
  - $\theta : common\, network\, parameters,\ \alpha : advantage\, stream\, parameters,\ \beta : value\, stream\, paramete$
  - This helps us accelerate the training. This helps us find much more relaible Q-values for each action by decoupling the estimation between two streams.
- **Prioritized Experience Replay**: Some experiences may be more important than others for our training, but might occur less frequently.
  - If we sample the experiences randomly these rich experiences that occur rarely have practilly no chance to be selected.
  - Use a priority. where there is a big difference between our prediction and the TD target, since it means that we have a lot to learn about it.
  - We use the absolute value of the magnitude of our TD-error: $p_t = |\delta_t| + e$, e = const, that assures that no experience has no 0 probability.
  - Put that priority in the experience of each replay buffer to select the experiences.
  - Do not go greedy prioritization: overfitting!. Stochastic prioritization: $P(i) = \frac{p_i^a}{\sum_k p_k^a}$, a reintroduces some randomness, a = 0 pure uniform randomness, a = 1 only select the experiences with the highest priorities.
  - To combat over-fitting by prioritization of high-priority samples use Importance sampling weights (IS): $(\frac{1}{N} * \frac{1}{P(i)})^b$, b = controls how much the w affects learning. Close to 0 at the beginning of learning and annealed up to 1 over the duration of training. Because these weights are more important in the end of learning when our q-values begin to converge.
  - To sort the replays use an unsorted sumtree

An introduction to Policy Gradients with Cartpole and Doom

- in policy-based methods we directly learn the policy function that maps state to action. we directly parameterize π
- Deterministic policies are used in deterministic environments. stochastic policy is used when the environment is uncertain. We call this process a Partially Observable Markov Decision Process (POMDP).
- **Advantage of Policy Gradients**:

  - **convergence**: policy-based methods have better convergence properties. value-based methods might oscillate alot. Policy based methods follow gradients we converge on a local maximum (worst case), or global maximum (best case).
  - Policy gradient are more effective in **high dimensional action spaces**: as Deep Q-learning is that their prediction assign a score for each eaction at each time step, given the current state.
  - Policy gradients **can learn stochastic policies**: value functions can't. In Policy we donÄt need to implement an exploration/explotation trade off.
- **Disadvantages of Policy Gradients**:
  - Alot of the time, they converge on a **local maximum** rather than on the global optimum.
  - **Slower convergence**: Then Deep Q-Learning.
- **Policy Search**: We have our policy π that has a parameter θ. THis pi outputs a probability distribution of actions.
  - $\pi_\theta(a|s) = P[a|s]$
  - Good policy: theta that maximizes the score function: $J(\theta) = E_{\pi\theta}[\sum \gamma r]$
  - **Steps**: 1st: Measure the quality of policy with a policy score function, 2nd: use policy gradient ascent to find best parameter theta that improves our policy.
  - **1st Step**: The Policy Score function J(theta):
    * Episodic environment: Calculate the mean of the return from the first time Step (G1): $J_1(\theta) = E_\pi[G_1 = \sum_{k=0}^{\infty} \gamma^k R_{1+k}] = E_\pi(V(s_1))$. We want a policy that optimizes G1, as this will be the best policy.
    * Continious Environment: We can use the average value, because we can't rely on a specific start state and their values are now weighted by the probability of the occurrence of the respected state: $J_{avgv}(\theta) = E_\pi(V(s)) = \sum d(s)V(s),\ where\ d(s) = \frac{N(s)}{\sum_s' N(s')}$
    * N(s) = Number of occurrences of the state.
    * use the average reward per timestap: $J_{avR}(\theta) = E_\pi(r) = \sum_s d(s) \sum_a \pi_\theta(s,a) R_s^a$. sum over a: Probability that I take this action a from that state under this policy, Rsa: immediate reward that I get.
  - **2nd Step**: Policy gradient ascent.
    * To maximize the score function J(theta), we need to do gradient ascent on policy parameters.
    * We use gradient ascent as the score function is not an error function (there we would use gradient descent.)
    * Goal: $\theta^* = \underset{\theta}{argmax} E_{\pi\theta}[\sum_t R(s_t, a_t)]$, Score function: $J(\theta) = E_\pi[R(\tau)]$
    * Problem: How do we estimate the Gradient with respect to theta, when the gradient depends on the unknown effect of policy changes on the state distribution?
    * Solution: $\nabla_\theta J(\theta) = E_\pi[\nabla_\theta(log\pi(\tau|\theta))R(\tau)], \pi(\tau|\theta)\ :\ policy\ function, R(\tau)\ :\ score\ function$
    * Update Rule: $\Delta\theta = \alpha * \nabla_\theta(log\pi(s,a,\theta))R(\tau)$
    * R(tau): High value: it means thatn on average we took actions that lead to high rewards. If it is low, we want to push down the probabilities of the actions seen.

  - Policy gradient can be improved with Proximal Policy Gradients (ensure that the deviations from the previous policy stays relatively small) and Actor Critic (a hybrid between value-based algorithms and policy-based algorithms).

An intro to Advantage Actor Critic methods: let's play Sonic the Hedgehog!

- **Actor Critic**: Hybrid method. Use two neural networks: A Critic that measures how good the action takesn is (value-based) and an Actor that controls how our agent behaves (policy-based).
- State of the art: **Proximal Policy Optimization (PPO)**, is based on Advantage Actor Critic.
- **Policy Gradient Problem**: Reward is done for-each episode, so small bad decisions will be averaged out. And we won't find an optimal policy.
- Use TD-Learning: $\Delta\theta = \alpha * \nabla_\theta * (log\pi(S_t, A_t, \theta)) * Q(S_t, A_t)$. We do update each step sou we don't use the total rewards R(t). The Critic model approximates the value function.
- The critic will help to find the policy and update their own way to provide better feedback.
- Actor: $\pi(s, a, \theta)$ Critic: $\hat{q}(s, a, w)$
- Weights: Policy: $\Delta\theta = \alpha\nabla_\theta(log\pi_\theta(s, a)) * \hat{q}_w(s, a)$, Value: $\Delta w = \beta(R(s, a) + \gamma\hat{q}_w(s_{t+1}, a_{t+1}) - \hat{q}_w((s_t, a_t))\ \ \nabla_w\ \hat{q}_w(s_t, a_t)$
- **Process**: At each time-step: current State St into Actor and Critic. Policy outputs Action At and receives a new State and a reward.
- The Critic computes the value of taking that action at that state and the actor updates is policy parameters (weights) using this q-value.
- To reduce the Variability: Use Advantage function: $A(s, a) = Q(s, a) - V(s)$ Q(s,a): q-value for action a in state s, V(s): average value of that state.
- This function calculates the extra reward I get if I take this action. A(s,a) > 0: our gradient is pushed in that direction, A(s,a) < 0: our gradient is pushed in the opposite direction.
- Use the TD-Error as an good estimator: $A(s, a) = r + \gamma V(s') - V(s)$
- Strategies: Synchronous: **A2C** (Advantage Actor Critic), Asynchronous: **A3C** (Asynchronous Advantage Actor Critic).
- A3C uses different agents in parallel on multiple instances of the environment. Each worker will update the global network asynchronously.
- Problem of A3C: Link. Because of asynchronous nature of A3C, some workers will be playing with older version of the parameters, thus the aggregating update will not be optimal. In A2C it waits for each actor to finish before updating the global parameters. Therefore the training will be more cohesive and faster.
- Each worker in A2C will ahve the same set of weights since, contrary to A3C, A2C updates all their workers at the same time. YOu can create multiple versions of environments and then execute them in parallel.

Proximal Policy Optimization (PPO) with Sonic the Hedgehog 2 and 3

2.5.3. Deep RL Bootcamp

Deep RL Bootcamp

2.5.4. RL Lectures from Deepmind

RL Course by DeepMind
RL Course by DeepMind - Part 1

- Actions may have long term consequences and rewards may be delayed. May need to sacrifice immediate reward to gain more long-term reward.
- $Observation\ O_t, Reward\ R_t, Action\ A_t, History\ H_t\ (sequence\ of\ O_t,\ A_t,\ R_t)$
- $State\ S_t$ (simpler information to determine what happens next, usually function of history: $S_t = f(H_t)$
- State Definitions:
  - enviroment state S_t^e is the enviroments private representation. Environment state not visible to the agent.
  - agent state S_t^a is the agents internal representation. Used to pick next action. $S_t^a = f(H_t)$
  - markov (property) state $A\ state\ S_t\ is\ Markov\ iff\ :\ \mathbb{P}[S_{t+1}|S_t] = \mathbb{P}[S_{t+1}|S_1, ..., S_t]$. You only need the current state to infer the next state or the future. A helicopter state needs velocity. Otherwise you need the complete history to calculate velocity if it only stored position.
  - environment state S_t^e and the history H_t is Markov.
- Environments:
  - fully observability: agent directly observes environment state $O_t = S_t^a = S_t^e$. This is a Markov decision process (MDP).
  - partial observability: $S_t^a \neq S_t^e$. This is a partially observable Markov decision process (POMDP). Agent constructs it's own S_t^a.
  - partial observability state: complete history $S_t^a = H_t$, beliefs: $S_t^a = (\mathbf{P}[S_t^e = s^1], ..., \mathbf{P}[S_t^e = s^n])$, recurrent NN: $S_t^a = \sigma(S_{t-1}^a W_s + O_t W_o)$ (linear transformation)
- Inside an RL Agent
  - policy (agent's behavior), value function (how good is state-action pair), model (agents representation of the environment).
  - model: predicts what the environment will do next. you don't need to do models.
  - Transitions: $\mathcal{P}$ predicts next state (dynamics). Rewards $\mathcal{R}$ predicts next immediate reward
  - e.g.: $\mathcal{P}_{ss'}^a = \mathbf{P}[S = s'|S = s, A = a]$, $\mathcal{R}_s^a = \mathbf{E}[R|S = s, A = a]$
  - model-free agent: Policy and/or Value Function and no model.
  - model-based agent: Policy and/or Value Function and a model. first build the dynamics of the environmant es with the model
- Problems with RL
  - RL-Problem: Environment initially unknown and the agent learns by interaction.
  - Planning-Problem: Environment-model is known from the start.
  - Prediction: evaluate the future (given a policy) vs. Control: optimise the future (find the pest policy)

RL Course by DeepMind - Part 2

- Markov Processes:
  - Markov Decision Processes Describe the environment for RL and is fully observable.
  - State Transition: $\mathcal{P}_{ss'} = \mathbb{P}[S_{t+1} = s'|S_t = s]$.

  - This allows a Matrix to be defined: $\begin{pmatrix} \mathcal{P}_{11} & \dots & \mathcal{P}_{1n} \\ \vdots & & \\ \mathcal{P}_{n1} & \dots & \mathcal{P}_{nn} \end{pmatrix}$. Each Row sums up to 1
  - Markov Process: tuple $\langle \mathcal{S}, \mathcal{P} \rangle$. S is a (finite) set os states. and P is a state transition probability matrix.
- Markov Reward Processes:
  - A MRP is a Markov Processes with the additions: tuple $\langle \mathcal{S}, \mathcal{P}, \mathcal{R}, \gamma \rangle$.
  - R is a reward function $\mathcal{R}_s = \mathbb{E}[R_{t+1} | S_t = s]$ and $\gamma \in [0, 1]$ is a discount factor.
  - G_t is the total doscounted reward form time-step t. Value function v(s) (see above).
  - Bellam Equation: $v(s) = \mathbb{E}[G_t | S_t = s] = \mathbb{E}[R_{t+1} + \gamma G_{t+1} | S_t = s] = \mathbb{E}[R_{t+1} + \gamma v(S_{t+1}) | S_t = s]$
  - This allows: $v(s) = \mathcal{R}_s + \gamma \sum_{s' \in S} \mathcal{P}_{ss'} v(s')$
  - In Matrix form: $\begin{bmatrix} v(1) \\ \vdots \\ v(n) \end{bmatrix} = \begin{bmatrix} \mathcal{R}_1 \\ \vdots \\ \mathcal{R}_n \end{bmatrix} + \gamma \begin{bmatrix} \mathcal{P}_{11} & \dots & \mathcal{P}_{1n} \\ \vdots & & \\ \mathcal{P}_{n1} & \dots & \mathcal{P}_{nn} \end{bmatrix} \cdot \begin{bmatrix} v(1) \\ \vdots \\ v(n) \end{bmatrix}$.
- Markov Decisions Processes:
  - A MDP Is a Markov reward Process with a finite set of actions. The State Transition and reward function now also depend on the action chosen.
  - stochastic policy: $\pi(a|s) = \mathbb{P}[A_t = a | S_t = s]$. They depend only on the current state. Policies are stationary (time-independent).
  - The state sequence given by any policy is itself a markov process (chain) $\langle \mathcal{S}, \mathcal{P}^{\pi} \rangle$. If we add the rewards we got through this policy induced sequence we get a MRP $\langle \mathcal{S}, \mathcal{P}^{\pi}, \mathcal{R}^{\pi}, \gamma \rangle$.
  - So: $\mathcal{P}^{\pi}_{s,s'} = \sum_{a \in \mathcal{A}} \pi(a|s) \mathcal{P}^{a}_{s,s'}$ and $\mathcal{R}^{\pi}_{s} = \sum_{a \in \mathcal{A}} \pi(a|s) \mathcal{R}^{a}_{s}$
  - So the transition dynamics and rewards are averaged over what our policy gives us.
  - state-value function $v_{\pi}(s) = \mathbb{E}_{\pi}[G_t | S_t = s]$
  - action-value function $q_{\pi}(s, a) = \mathbb{E}_{\pi}[G_t | S_t = s, A_t = a]$.
  - bellman equation for state-value functions: $v_{\pi}(s) = \mathbb{E}_{\pi}[R_{t+1} + \gamma v_{\pi}(S_{t+1}) | S_t = s]$
  - bellman equation for action-value functions: $q_{\pi}(s, a) = \mathbb{E}_{\pi}[R_{t+1} + \gamma q_{\pi}(S_{t+1}, A_{t+1}) | S_t = s, A_t = a]$
  - V-Step: $v_{\pi}(s) = \sum_{a \in \mathcal{A}} \pi(a|s) q_{\pi}(s, a)$. For a given state we average the actions we can take
  - Q-Step: $q_{\pi}(s, a) = \mathcal{R}^{a}_{s} + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}^{a}_{ss'} v_{\pi}(s)$. For a given state how good is it to do a given action we averate the situations we could go to.
  - Equation for v_{π}: $v_{\pi}(s) = \sum_{a \in \mathcal{A}} (\mathcal{R}^{a}_{s} + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}^{a}_{ss'} v_{\pi}(s))$. state-value relates to the state-value of the next step.
  - Equation for q_{π}: $p_{\pi}(s, a) = \mathcal{R}^{a}_{s} + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}^{a}_{ss'} \sum_{a \in \mathcal{A}} \pi(a|s) q_{\pi}(s, a)$. q-value relates to the q-value of the next step.
  - Bellman Expectation Equation in Matrix form: $v_{\pi} = \mathcal{R}^{\pi} + \gamma \mathcal{P}^{\pi} v_{\pi}$. Direct Solution $v_{\pi} = (I - \gamma \mathcal{P}^{\pi})^{-1} \mathcal{R}^{\pi}$. $\mathcal{P}^{\pi}$ and $\mathcal{R}^{\pi}$ are averages.
  - Optimality
    * optimal state-value function $v_{*}(s) = \underset{\pi}{max}\, v_{\pi}(s)$. optimal action-value function $q_{*}(s, a) = \underset{\pi}{max}\, q_{\pi}(s, a)$

    * discounted reward does not have the problem of infinite loops of positive rewards.
    * un-discounted rewards need to fulfill either an average reward (average reward RL) or certain technical conditions must be met so un-discounted MDP are guaranteed to terminate. (well defined).
    * partial ordering over policies: $\pi \geq \pi' \ if \ v_\pi(s) \geq v'_\pi(s), \forall s$
    * Optimal Policy: For any MDP there exists at least one an optimal policy thate is better or erqual to all other policies: $\pi_* \geq \pi, \forall \pi$
    * All optimal policies achieve the optimal value and optimal action-value function.
    * optimal policy through optimal q-function $\pi_*(a|s) = \begin{cases} 1 & if \ a = \underset{a \in \mathcal{A}}{argmax} \ q_*(s,a) \\ 0 & otherwise \end{cases}$.
    * There is always a deterministic optimal policy for any MDP.
    * Bellman Optimality Equation for v*. $v_*(s) = \max_a \mathcal{R}_s^a + \gamma \sum_{s' \in S} \mathcal{P}_{ss'}^a v_*(s')$. This is a 1-step look ahead.
    * Bellman Optimality Equation for q*. $q_*(s,a) = \mathcal{R}_s^a + \gamma \sum_{s' \in S} \mathcal{P}_{ss'}^a \max_{a'} q_*(s',a')$.
    * Bellman Optimality Equation is non-linear. No closed form solution (in general). Needs iterative solution: value-iteration, policy-iteration, Q-learning, Sarsa.
- Infinite and continuous MDPs
  - countably infinite state and/or action spaces: straightforward
  - continuous state and/or action spaces: Closed form for linear quadratic model (LQR)
  - continuous time: Requires partial differntial equations (Hamilton-Jacobi-Bellman).
- Partially observable MDPs
  - A POMDP is a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{O}, \mathcal{P}, \mathcal{R}, \mathcal{Z}, \gamma \rangle$ with hidden states. Hidden Markov Model with actions.
  - $\mathcal{O}$ is a finite set of observations.
  - $\mathcal{Z}$ is an observation function, $\mathcal{Z}_{s'o}^a = \mathbb{P}[O_{t+1} = 0 | S_{t+1} = s', A_t = a]$
  - History changes $H_t = A_0, O_1, R_1, ..., A_{t-1}, O_t, R_t$
  - Beilief state is a probability over states conditioned on the history $b(h) = (\mathbb{P}[S_t = s^1 | H_t = h], ..., \mathbb{P}[S_t = s^n | H_t = h])$
  - A POMDP can be reduced to an infinite history tree or an infinite belief state tree.
- Undiscounted, average reward MDPs
  - Ergodic Markov Process: Is recurrent (each state is visited an infinite number of times) and Aperiodic (each state is visited without any systematic period)
  - An ergodic Markov process has a limiting stationary distribution $d^\pi(s) = \sum_{s' \in S} d^\pi(s') \mathcal{P}_{s's}$
  - An MDP is ergodic if the Markov chain induced by any policy is ergodic.
  - Average reward per time-step $\rho^\pi = \lim_{T \to \infty} \frac{1}{T} \mathbb{E}[\sum_{t=1}^T R_t]$
  - extra reward due to starting from satte s: $\tilde{v}_\pi(s) = \mathbb{E}_\pi[\sum_{k=1}^\infty (R_{t+k} - \rho^\pi) | S_t = s]$

2.6. ALR-Wiki

ALR-Wiki Link
ALR-Wiki Books Link

Implementation Guidelines
Latex Templates
Writing Checklist
How to do Research

2.7. Books

2.7.1. Deep Learning Book

Deep Learning Book

2.7.2. Reinforcement Learning - An Introduction

Reinforcement Learning
a

2.8. UNSORTED

Gordon 2000: Ants at Work.
Gordon 2007: Control without hierarchy. Nature.
Links:

Ant Simulation Video 1
Ant Simulation Video 2
Boids Video
Distributed Artificial Intelligence, Wikipedia
Multi-agent learning, Wikipedia
Bees algorithm, Wikipedia
Swarm Intelligence, Wikipedia

2.9. References & Papers

2.9.1. Ant Colony Optimization (ACO)

ACO - Ant Colony Optimization for learning Bayesian network - 2002

2.9.2. Graph Neural Networks

(Base) GNN - Sur - Theoretical Foundations of Graph Neural Networks - 2021

- Goal: Exact same results for two **isomorphic graphs** (graphs that are the same the nodes are just arranged differently).

- Nodes: $x_i \in \mathbb{R}^k$ (features of node), feature matrix $\mathbf{X} = (x_1, ...x_n)^T \in \mathbb{R}^{k \times n}$
- By stacking the nodes in the matrix you have already ordered them (result should not depend on this).
- Operations that change the nod order: permutation matrices. They ahve exactly one 1 in every row and column, and zeroes everywhere examples. Left Multiplied: permute the rows. $P_{(2,4,1,3)}$: The numbers indicate where the 1 in the row is.
- **Permutation Invariance**: f is permutation invariant iff: $\forall P \in Permutation : f(PX) = f(X)$. Example: Deep Sets Model $f(X) = \phi(\sum_{i \in V} \psi(x_i))$. This is true for the entire data-set.
- **Permutation equivariance**: for identification on the node level. Seek functions that don't change the node order. f is permutation equivariant iff: $\forall P \in Permutation : f(PX) = Pf(X)$.
- **equivariance**: each node's row is unchanged by f. So foreach node we could define: $\forall i : h_i = \psi(x_i)$., the latent features h-i. Stacking h yields: $H = f(x)$. The functions are applied indipendently to each node.
- Stacking equivarent functions with an invariant tail: $f(X) = \phi(\bigoplus_{i \in V} \psi(x_i))$. $\bigoplus$ $is permutationinvar$
- **Learning on Graphs**:
  - Represent Edges with adjacency matrix A: $a_{ij} = \begin{cases} 1 & (i,j) \in E \\ 0 & otherwise \end{cases}$. Edge features could be added aswell. permutation x-variance still holds.
  - xvariance on graphs: To not change edges: permutate rows and columns. Permutate with PAP^T.
  - **Invariance**: $f(PX, PAP^T) = f(X, A)$ $(A = Edges,\ X = Nodes)$
  - **Exvariance**: $f(PX, PAP^T) = Pf(X, A)$ $(A = Edges,\ X = Nodes)$
  - Neighbourhoods: Node i, it's 1-hop neighbors are defined as: $\mathcal{N}_i = \{j : (i,j) \in E \vee (j,i) \in E\}$. (Non-directed edges, node i is in it's own neighbourhood).
  - Multiset of features in the neighbourhood: $X_{\mathcal{N}_i} = \{\{x_j : j \in \mathcal{N}_i\}\}$. With a local function g as operating over this multiset: $g(x_i, X_{\mathcal{N}_i})$
  - Construct perm-equi function f(X,A) by applying g over all neighbourhoods: $f(\mathbf{X,A}) = \begin{pmatrix} g(x_1, X_{\mathcal{N}_1}) \\ g(x_2, X_{\mathcal{N}_1}) \\ \vdots \\ g(x_n, X_{\mathcal{N}_n}) \end{pmatrix}$. g should not depend on the order of the neighbourhood, it should be permu-invari.
  - Once you have the latent-Graph via the GNN you can use them in a Node-classification, Graph-classification, or Link-prediction task.
- Message Passing in Graphs.
  - GNN Layer: Construct f(X,A) via the local function g (known as diffusion, propagation or message passing). F is refered ta as a GNN layer.
  - How to define g? Active research!
  - Classification thre flavoulrs of CNN:
  - Convolutional GNN:
    * constants c_{ij}. How much does Node i value the features of nodes j. They are coefficients for weighted combinations. The weights usually depend on A.

    - $h_i = \phi(x_i, \bigoplus_{j \in \mathcal{N}_i} c_{i,j}\psi(x_j))$.
    - Examples: ChebyNet, GCN (Graph Convolutional Network), SGC (Simplified Graph Convolutional Networks)
    - useful for homophilous graphs (similar edges) and scales well.
  - Attentional GNN:
    - neighbour features aggregated with implicit weights (via attention a). This weights are learnable.
    - $h_i = \phi(x_i, \bigoplus_{j \in \mathcal{N}_i} a(x_i, x_j)\psi(x_j))$.
    - Examples: MoNet, GAT (Graph Attention Network), GaAN (Gated Attention Network).
    - useful as a middle ground with respect to capacity and scale. Edges are not strict homophily, but you compute sclarar value in each edge.
  - Message Passing GNN:
    - sender and receiver work together to compute arbitrary vectors ("messages") to be sent across edges.
    - $h_i = \phi(x_i, \bigoplus_{j \in \mathcal{N}_i} \psi(x_i, x_j)).\psi(x_i, x_j)) = m_{ij}$.
    - Examples: Interaction Networks, MPNN (Message Passing Neural Networks), GraphNets
    - most generic GNN. May have scalability or learnability issues. Ideal for reasoning.
- Node embedding techniques:
  - embedding: Finding an Encoding, so that x-i are now the latent features of h-i.
  - a good representation should preserve the graph structure. This leads to the unsupervised objective: $optimise\ h_i\ and\ h_j\ to\ be\ nearby\ iff : (i, j) \in E$. They predict if there is an edge between the nodes.
  - Can use binary cross-entropy loss: $\sum_{(i,j) \in E} log\sigma(h_i^T h_j) + \sum_{(i,j) \notin E} log(1 - \sigma(h_i^T h_j))$
- local objective emulate Convolutional GNNs. Neighbouring nodes tend to highly overlap in n-step neighborhoods. A conv-GNN enforces similar features for neighbouring nodes by design.
- GNN and Natural Language Processing (NLP) correspond alot (nodes similar to words).
- Common assumption if you have no information about how the graph should look like: Assume a complete graph and then let the network infer the actual relations.
- Transformers: are fully connected attentional GNNs.
- Spectral GNNs:
  - Time Sequences can be imageind as a cyclical grid graph with a convolution over it. A node is a time-step and the convolution looks at the time step and it's immediate neighbors.
  - You don't need to know the convolutional operation if you know the eigenvalues with respect to the fourier basis (36:13)
  - convolutional GNN: $c_{ij} = (p_k(L))_{ij}$. Use a polynomial function p-k for the Laplacian matrix L = D - A. D being the Degree matrix. p-k is necessary to make the eigenvalue decomposition easier.
  - This means there is no spectral GNN and spatial GNN as they can be transformed into each other.
- Probabilistic Graphical Models:

  - Nodes are random variables and edges are dependencies between their distributions. THis is an Probabilistic graphical Model (PGMs). This helps you decompose a joint probability distribution.
  - Can use Markov Random Fields (MRF) to decompose the joint into a product of edge potentials.
  - Mean-field inference.
  - PGM corresponds to a message passing GNN.
- how powerful are GNNs?
  - untrained GNNs work well, as they are similar to random hashes. (Weisfeiler-Lemann Test). Also called 1-WL test.
  - Though some instances the isomorphism test fails.
  - GNNs can only be as powerful as the 1-WL test.
  - Can make the strongers by analysing failer cases.
  - Continous Features: Sums may not destinguish parts of the graph (2+2 = 4+0).
- curr

2.9.3. Reinforcement Learning

Multiple blog post for RL (even has A3C)
RL - DQN - Human-level control through deep reinforcement - 2015
RL - A3C - Asynchronous Methods for Deep Reinforcement Learning - 2016
(Sur)veys/Reviews
RL - Sur - State-of-the-art Reinforcement LEarning Algorithms - 2020

2.9.4. Multiagent Systems (MAS)

(App)lications

Actor-Critic (AC)

(Base)
MAS - Base - The Multiagent Planning Problem - 2016

(Com)munication

(Con)ference
MAS - Con - Distributed Cooperative Control and Communication for Multi-agent Systems - 2021
MAS - Con - PRIMA 2020 Principles and Practice of Multi-Agent Systems - 2021
MAS - Con - Swarm Intelligence - 2010
MAS - Con - Swarm Intelligence - 2012
MAS - Con - Swarm Intelligence - 2014
MAS - Con - Swarm Intelligence - 2016
MAS - Con - Swarm Intelligence - 2018

MAS - Con - Swarm Intelligence - 2020

(Evo)lutionary
MAS - Evo - Co-evolutionary Multi-agent System with Predator-Prey Mechanism for Multi-objective Optimization - 2007

(Het)erogeneous
MAS - Het - Multiagent Systems A Survey from a Machine Learning Perspective - 2000

(Hie)rarchy
MAS - Hie - Hierarchical Control in a Multiagent System - 2007
MAS - Hie - Holonic - A Taxonomy of Autonomy in Multiagent Organisation - 2003

Multi-Objective (MO)

(Role)-Based

(Sca)ling

(Sur)veys/Reviews
MAS - Multi-Agent Systems - A Survey - 2018
MAS - Sur - A survey of multi-agent organizational paradigms - 2004

(Tra)nsferlearning
MAS - Tra - Transfer Learning for Multi-agent Coordination - 2011
MAS - Tra - Transfer learning in multi-agent systems through paralllel transfer - 2013

2.9.5. Multi Agent Reinforcement Learning (MARL)

(App)lications

Actor-Critic (AC)
MARL - AC - Networked Multi-Agent Reinforcement Learning in Continuous Spaces - 2018
MARL - AC - Multi-Agent Actor-Critic for Mixed Cooperative-Competitive Environment - 2017
MARL - AC - Actor-Attention-Critic for Multi-Agent Reinforcement Learning - 2019

(Base) MARL - Base - Multiagent Reinforcement Learning - Theoretical Framework and an Algorithm - 1998
MARL - Base - Deep Reinforcement Learning for Robot Swarms - 2019 - KIT
MARL - Base - PettingZoo - Gym for Multi-Agent Reinforcement Learning - 2020
MARL - Base - Multi-agent reinforcement learning weighting and partitioning - 1999

(Com)munication
MARL - Com - Learning to Communicate with Deep Multi-Agent Reinforcement Learning -

2016
MARL - Com - Coordinating multi-agent reinforcement learning with limited communication - 2013

(Con)ference

(Evo)lutionary

(Het)erogeneous
MARL - Het - LIIR - Learning Individual Intrinsic Reward inMulti-Agent Reinforcement Learning - 2019
MARL - Het - An approach to the pursuit problem on a heterogeneous multiagent system using reinforcement learning - 2002

(Hie)rarchy
MARL - Hie - Hierarchical multi-agent reinforcement learning - 2006

Multi-Objective (MO)
MARL - MO - Reward shaping for knowledge-based multi-objective multi-agent reinforcement learning - 2017

(Role)-Based
MARL - Role - ROMA Multi-Agent Reinforcement Learning with Emergent Roles - 2020

(Sca)ling
MARL - Sca - GAMA - Graph Attention Multi-agent reinforcement learning algorithm for cooperation - 2020
MARL - Sca - Plan-based reward shaping for multi-agent reinforcement learning - 2016
MARL - Sca - Multi-Agent Reinforcement Learning Using Linear Fuzzy Model Applied to Cooperative Mobile Robots - 2018
MARL - Sca - Stabilising Experience Replay for Deep Multi-Agent Reinforcement Learning - 2017
MARL - Sca - Mean Field Multi-Agent Reinforcement Learning - 2018
MARL - Sca - A modular approach to multi-agent reinforcement learning - 2005

(Sur)veys/Reviews
MARL - Sur - Multi-Agent Reinforcement Learning - a critical survey - 2003
MARL - Sur - Multi-Agent Reinforcement Learning A Selective Overview of Theories and Algorithms - 2021
MARL - Sur - Multi-Agent Reinforcement Learning A Report on Challenges and Approaches - 2018
MARL - Sur - A Review of Cooperative Multi-Agent Deep Reinforcement Learning - 2019
MARL - Sur - A Survey on Transfer Learning for Multiagent Reinforcement Learning Systems - 2019

(Tra)nsferlearning
MARL - Tra - Transfer Learning in Multi-agent ReinforcementLearning Domains - 2011
MARL - Tra - Parallel Transfer Learning in Multi-Agent Systems What, when and how to transfer - 2019
MARL - Tra - Transfer among Agents An Efficient Multiagent Transfer Learning Framework - 2020
MARL - Tra - Agents teaching agents a survey on inter-agent transfer learning - 2019

2.9.6. GNN for Multi Agent Reinforcement Learning (GNNMARL)

(App)lications GNN - App - Optimizing Large-Scale Fleet Management on a Road Network using Multi-Agent Deep Reinforcement Learning with Graph Neural Network - 2020

UNSORTED!!!!! GNN - Deep Multi-Agent Reinforcement Learning with Relevance Graphs - 2018
GNN - Deep Implicit Coordination Graphs for Multi-agent Reinforcement Learning- 2020
GNN - Multi-Agent Game Abstraction via Graph Attention Neural Network - 2020
GNN - Scaling Up Multiagent Reinforcement Learning for Robotic Systems Learn an Adaptive Sparse Communication Graph - 2020
GNN - Graphcomm A Graph Neural Network Based Method for Multi-Agent Reinforcement Learning - 2021
GNN - Towards Heterogeneous Multi-Agent Reinforcement Learning with Graph Neural Networks - 2020
GNN - The Emergence of Adversarial Communication in Multi-Agent Reinforcement Learning - 2020
GNN - Multi-Agent Deep Reinforcement Learning using Attentive Graph Neural Architectures for Real-Time Strategy Games - 2021
GNN - Global-Localized Agent Graph Convolution for Multi-Agent Reinforcement Learning - 2021
GNN - Specializing Inter-Agent Communication in Heterogeneous Multi-Agent Reinforcement Learning using Agent Class Information - 2020
GNN - Collaborative Multiagent Reinforcement Learning by Payoff Propagation - 2006
Write up for GCNs - 2016
GNN - Sur - A Comprehensive Survey on Graph Neural Networks - 2019

2.9.7. Applications

MAS - TrafficControl - Neural Networks for Continuous Online Learning and Control - 2006

3. Fundamentals

Topics:

- multiagent/multibody Systems (MAS).
  - MAS Reinforcement Learning
    * They use stochastic games (Markov Games) as generalization of Markov Decision Processes.
  - Hierarchical MAS, Hierarchical Reinforcement Learning for MAS.
  - MAS with Cooperation and Competition.
  - Particle Swarms (nicht so meins).
  - Problems:
    * MAS Movement Problems (Potential Fields). Mean fields?
    * ? using MAS for Moving a Multi-Legged Robot (Spider-like) with a navigation problem design as a hierarchical MAS?
    * Path Planning Navigation with Heterogeneous Agents?
    * MAS Task Problems: Rendezvous, Pursuit Evasion (Single and one Evader) (Boid?), (MAS - Deep Reinforcement Learning for Robot Swarms - 2019 - KIT)
    * Multi-Agent Path Finding (MAPF). Scalability for this: For fixed space they get into each others way.
    * Collective Foraging: (Ants-kinda). Problem when communication only happens in an area, use local information exchange groups. Information Transfer. (MAS - Swarm Intelligence - 2020), Preferential Foraging (MAS - Swarm Intelligence - 2018 - p.289)
    * Coverage: Multi-robot Information Gathering / Scouting. (MAS - Swarm Intelligence - 2020), Pattern Formation (MAS - Swarm Intelligence - 2016 - p.14)
    * Coalition: Heterogeneous Group of Agents. They have different skills / attrributes that affect the environment. Like an Ant Caste System? Some Agents have better sensors? Only some agents have some sensors? What if the specialization is taken to the extreme? (MAS - Swarm Intelligence - 2020). Limited Visibility Sensors (MAS - Swarm Intelligence - 2018 - p.56). Going from a homogeneous group of agents, to randomized specialities, to extrem specializations. How does it change? Mixed with an Hierarchical Approach? They need to find groups to work together? Some are fast (but cannot see much, there is an insect that cannot see while running), Some have good sensors. Communication range? Genetic Diversity, Task-Allocation and Task-Switching (MAS - Swarm Intelligence - 2016 - p.109)
    * Collective Gradient Perception: Using Abilities of other Agents to take advantage of the whole group. (Flocking) (MAS - Swarm Intelligence - 2020)
    * Indirect Communication through changing states in the environment (birds transport something via cable). Also like using Pheromone Trails (Quality-

Sensitive Foraging through virtual pheromone trails). (MAS - Swarm Intelligence - 2018 - p.15 - p.147)
    * Control Architecture: Behavior Trees, FSM. (MAS - Swarm Intelligence - 2018 - p.42)
    * Maze-Like Environment with Ant Algorithms (MAS - Swarm Intelligence - 2018 - p.162)
    * Search and Rescue? (Kinda like Foraging?)
    * Disruption: Disrupting Aspects of the Swarm and how they react to it, Swarm Attack: (MAS - Swarm Intelligence - 2018 - p.225), Coherence of Collective Decision Making (MAS - Swarm Intelligence - 2018 - p.264)
    * Evolutionary Systems: NEAT (MAS - Swarm Intelligence - 2012 - p.98)
- Graph-Based Visualisation for MAS. (MAS - Swarm Intelligence - 2010). How do you visualize them?
- Transfer Learning for MARL/MAS
    * Some approaches for parallel transfor of different problems even for MARL Problems.
    * So you can transfer even in parallel.
    * But they only transfer between similar problems. Which would held if you can create a simpler version of your problem and make it more and more complex.
    * Are there transfer learning approaches for MAS/MARL, so that Learning can be transfered between agents? So that if you add agents the complexity isn't as steep?
- Adaptive Learning for MAS?
- Control System: Either fully self-organizing or completely centralized. Hybrid Control of Swarms (MAS - Swarm Intelligence - 2018 - p.69)
- Simulation of MAS: ARGoS
- Best-of-n Problem: Swarm selects best option out of n alternatives. (MAS - Swarm Intelligence - 2018 - p.251)
- Sensory Errors for Foraging, Dynamic Task Partiotioning (MAS - Swarm Intelligence - 2016 - p.124), Task Partitioning Problem (MAS - Swarm Intelligence - 2012 - p.122)
- Task Hierarchy, Multi-Objective
- Random Walks as a search strategy (MAS - Swarm Intelligence - 2016 - p.196)
- Critic: Centralized Critic or Learning individual intrinsic reward (LIIR)
- Standardizing Testing Scenarios (PettingZoo).
- Role concept to for MAS. Agents with similar role share similar behavior.
- Modular Approach to MARL to remedy the poor scalability in the state-space in the number of partner agents.
- Weighting and Partitioning to decrease complexity.
- Hierarchical Groups of MAS where each epoch each Group (5 Agents) exchange Data for learning. And every 10 Epoch the groups exchange data for learning? Randomize these groups? (every few epoch?). Groups of 5 that get reshuffled every 10 Epoch or so.
- holonic agent structure: fractals structure of MAS. holonic and heterogenous? Does this form naturally for extreme heterogenous?
- holonic coalitions?

- GNN
  - How many Hops? Use 2-hop neighborhoods: https://youtu.be/H6oOhElB3yE?t=842 (Realworld networks have a small diameter => cannot afford 3-hops).
  - Which GNN Type for MARL? Convolutional, Attentional, Message-passing GNN.
  - Where do you apply GNNs? As the communication between the agents (preprocess, communication relevance or state representation), or the actual function to learn with RL? (The Graph needs to emcompass the q-function. The reward.), or a relation graph.
  - Don't learn the graph structure, learn the weights. MAS: Nodes are Agents, Edges are their communications.
  - Complete Graph: Weights dynamic threshhold. What an agent can see defined over the weights.
- RL
  - un-discounted rewards need to fulfill either an average reward (average reward RL) or certain technical conditions must be met so un-discounted MDP are guaranteed to terminate. (well defined). How does this work?
  - Uncertainty in MDP: Use bayesian to not solve one MDP but a distribution of MDP. Use the uncertainty in your representation of the MDP itself non explicit uncertainty. States could have different discount factors to model your uncertainty of informations you have of the environment.
  - transformations for a risk-sensitive MDP into a risk-unaware MDP.

MARL - Sur - Multi-Agent Reinforcement Learning - a critical survey - 2003

4. Problem and Approaches

4.1. Definition of the Problemdomain

5. Project

Idee:

- Based on the KIT Paper "Deep Reinforcement Learning for Robot Swarms".
- Look at Approach from a handful of papers and start adjusting what KIT did by adding to it.
- Add GNN instead of mean of all the agents observations use a GNN here. (Mesasge Passing)
- Find Problems that the KIT-Approach cannot handle well or is just able to and show how my additions lets it tackle harder tasks.
- Improve Scalability on the KIT-Approach (to better see the Heterogeneity?)
- Aspects: Direct/Indirect Communication, Control-Architecture (centralized, hybrid, self-organization), Cooperative/Competetive Environments, Global/Decentralized Communication.
- Given the problem a completly self-organized approach would be super interesting (no centralized critic).
- Tasks: Start with one simple task. Can add more later. Predator-Prey, Foraging
- Heterogeneity: Would be cool to have 3 cases: fully-homogeneous, (reelle zahlen (0,1))-skill-heterogeneous, binary-skill-heterogeneous (communication + one skill)

6. Related Works

Is this part of the motivation???

7. Conclusion

Bibliography

[1] Steffen Becker, Heiko Koziolek, and Ralf Reussner. "The Palladio Component Model for Model-driven Performance Prediction". In: *Journal of Systems and Software* 82 (2009), pp. 3–22. DOI: 10.1016/j.jss.2008.03.066. URL: http://dx.doi.org/10.1016/j.jss.2008.03.066.

A. Appendix

A.1. First Appendix Section

Figure A.1.: A figure

...