Using Concepts to Understand Intelligent Agents

Ole Meyer, Marc Hesenius, and Volker Gruhn
University of Duisburg-Essen
Schtzenbahn 70
45127 Essen
Germany
{*firstname.lastname*@uni-due.de}

Abstract

Intelligent agents using reinforcement learning offer interesting capabilities for optimizing processes, products, and services in various branches of industry. Developing such applications at an economically viable level, however, is still a major challenge: Regardless of the potential offered by reinforcement learning, many solutions suffer from a lack of explainability. Deploying an application that makes unexplainable decisions is a potential risk to a project's success. Despite the ongoing effort, most reinforcement learning solutions remain a black box. In this position paper, we motivate to use concepts derived from human knowledge to unveil the inner workings of an intelligent agent on a more meaningful level without having to limit the algorithms themselves.

Introduction

Reinforcement learning, a method to create *intelligent agents*, is a promising area of Machine Learning (ML) and Artificial Intelligence (AI). In supervised and unsupervised learning methods, algorithms gather knowledge from existing data and search for patterns, which in turn are applied to new data. Thus, algorithms are fed by existing experiences collected in the past. With reinforcement learning, agents typically gather their own experiences by training in simulations. Successful actions are rewarded and the agents aim to maximize their profits by improving their strategies. Several recent examples show the potential: intelligent agents learned to play Chess (Silver et al. 2017a) and Go (Silver et al. 2017b), fly helicopters (Ng et al. 2006; Abbeel et al. 2007), or treat patients with serious illnesses (Parbhoo et al. 2017).

The core of reinforcement learning is an optimization process driven by random exploration and increasing exploitation of knowledge from gathered experiences. The agent more or less blindly stumbles through the various potential actions, probing for success and reaching for higher rewards. Prominent pitfalls are that agents find highly rewarded strategies unintended by developers (Amodei et al. 2016) or encounter a local maximum in the reward function and do not move further to explore possible alternatives due to a temporary loss of reward. Furthermore, applications based on reinforcement learning tend to suffer from a lack of reproducibility, even when trained with the same hyperparameters, and small changes in the parameters themselves quickly lead to very different results (Henderson et al. 2018), making reinforcement learning solutions unreliable. Furthermore, agents cannot explain the reasoning behind their actions, which leads to uncertainties during the development cycle, because such situations cannot be debugged. From a software engineering perspective, high uncertainties, which cannot be controlled by the developer, are a strong threat to the targeted and successful development of software projects.

Explainability of AI algorithms – often referred to as *Exaplainable AI* or *XAI* – is a major research topic, but most approaches have a strong focus on analyzing algorithms and the way data is processed. Unfortunately, we have to argue that the current state of the art still lacks the necessary progress to mitigate the risks resulting from unexplainable erroneous agent behavior. However, in this position paper we argue that techniques well-known in software engineering can be used to achieve a first level of explainability on a more meaningful level.

In this paper, we contribute to the emerging field of software engineering for intelligent agents by describing how concept-based engineering (Meyer and Gruhn 2019) can be used to reason over agent decisions. This approach can be used to derive curricula and hierarchical decompositions to define an agent's training. Applying curricula to machine learning is also known as *Machine Teaching* (Simard et al. 2017) or *Curriculum Learning* (Bengio et al. 2009), which is often used for supervised learning tasks. Machine teachers provide additional knowledge and guide the agent's learning process to avoid wasting learning time with strategies that are known to be inefficient.

We will first review related work with regard to reinforcement learning and XAI. We then introduce our engineering approach and describe thereafter how this can be used to explain an agent's decisions and behavior. Finally, we conclude the paper with an outlook on future work.

Background and Related Work

We will know discuss related work in the areas of machine learning. First, we will focus on the development of intelligent agents and reinforcement learning. We then review recent approaches for Explainable AI.

Developing Intelligent Agents – Technical Background

Reinforcement Learning enables intelligent software agents to optimize their behavior based on past experience. We assume that the agent lives in a world characterized by an underlying Markov Decision Process (MDP) consisting of the tuple (S, A, P, R, γ). The set S contains all possible states and the set A all possible actions that the agent can choose. Each state $s \in S$ and a valid selected action $a \in A$ is followed by a transition to a subsequent state $s' \in S$. While the transition can be deterministic, in reality, uncertainty factors may occur making it more probabilistic. $P(s, a, s')$ thus describes the probability of a transition to the next state. Any transition from a state s to a subsequent state s' by executing an action a may result in an immediate value given by the reward function R, which can be sparse in some cases, meaning that not every new state necessarily has an immediate value. In many real-world problems, different steps need to be taken before a particular outcome can be achieved, and in order to be able to make long-term decisions accordingly, the expected rewards usually have to be considered over several steps. One possibility is to sum up the expected rewards within a k finite horizon, but the result $\mathbb{E}\left[\sum_{t=0}^{k-1} r_t\right]$ is then limited by k, so that potential rewards are not considered in subsequent states. Since there is usually no fixed k, it is more common to use a discount factor $0 \leq \gamma \leq 1$ to represent the expected future reward $\mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r_t\right]$ with an infinite horizon (Kaelbling, Littman, and Cassandra 1998).

The assumption in an MDP is that the agent can directly observe the state s. In practice, the state variables are often only partially or indirectly observable, thus a Partially Observable Markov Decision Process (POMDP) exists and the agent has to estimate the actual state from its observations. The result is a set B of possible estimates called *beliefs*. It is possible to reduce the POMDP to an MDP by considering the set of possible beliefs themselves as a set of possible states in which the agent can find itself. The result is an MDP with a continuous state space described by (B, A, P, R, γ).

Given a complex MDP or POMDP, with reinforcement learning a value function is used that calculates the immediate value of a state s and quantifies its value by taking into account the discounted expected rewards in the future. The value function is defined by $V(s) = \max_a \sum_{s'} P(s, a, s') \left[R(s, a, s') + \gamma V(s')\right]$. Based on this function, the Q-function can be derived, describing the expected future discounted reward, given a state s and additionally an already selected action a. The Q-function is defined by $Q(s, a) = \sum_{s'} P(s, a, s') \left[R(s, a, s') + \gamma V(s')\right]$.

Q-Learning is a method of reinforcement learning that learns to approximate the Q-function. This is a kind of mapping problem of the state to the corresponding Q-value, which can be learned with supervised learning. In theory, any function approximator can be used, such as decision trees (Pyeatt and Howe 2001). In practice, however, one often finds neural networks for more complex problems. If the learning process is sufficiently accurate, the information resulting from the approximated Q-function can be used to select the policy for the next action. One possibility, for example, is to always use the action with the maximum Q-value.

Q-Learning is potentially capable of dealing with very large and continuous state spaces. Especially if neural networks are used, all the possibilities of this technique can be used. For example, the processing of image data by applying convolutional layers or temporal data using LSTM cells or similar methods is possible. However, the action space must be discrete and is limited in size (Lillicrap et al. 2015). In many areas, however, continuous action spaces can be found. This requires learning a direct mapping function from the state space to the action space using policy gradient methods.

However, these methods often suffer from large variance in policy gradients (Grondman et al. 2012), making them potentially unstable (Lillicrap et al. 2015). Many successful reinforcement learning algorithms are therefore based on *actor-critic architectures* that can combine the advantages of both, policy gradients and value function approximation. One model, the actor, maps the observation space into the action space and another model, the critic, learns the underlying value function. Having a good approximation of the value function may help to stabilize the estimates of the policy gradient. Examples of techniques based on the actor-critic architecture are Deep Deterministic Policy Gradients (DDPG) (Lillicrap et al. 2015), Trust Region Policy Optimization (TRPO) (Schulman et al. 2015), and Proximal Policy Optimization (PPO) (Schulman et al. 2017).

Developing Intelligent Agents – Challenges

Reinforcement learning strategies require the exploration of new ways to gain experience from which the best potential actions can be approximated. Without prior knowledge, only random exploration can take place. With increasing knowledge about the environment, the agent might exploit it further to find the best possible ways to reach the desired goal. Exploration and exploitation are therefore important elements of intelligent agents. The challenge is to find the right trade-off between exploration and exploitation.

One of the core things that makes reinforcement learning work is the Bellmann Equation (Lillicrap et al. 2015). It ensures that the agent's policy converges to the optimum, assuming infinite visits to each state of the environment and infinite learning steps. Even if this cannot be achieved, a finite number of steps is usually sufficient to find an acceptable approximation. However, state spaces quickly become large, due to the curse of dimensionality. The number of necessary steps usually increases with the size of the state and action space. The problem is obviously even bigger in continuous spaces and can only be solved with generalization, which means trying to learn the underlying function with the help of a function approximator like neural networks. This principle is very well known from supervised learning.

Having a good function approximation allows to deduce from data that have not been seen before. In contrast to supervised learning, however, reinforcement learning does not know the real target function. Instead of the real values having to be specified and labeled, the target function is tried to be inferred from observations of the real environment. This makes reinforcement learning more open to new solutions that are less influenced by human biases, but has two shortcomings: dynamic training data and the lack of a deterministic transition function.

The set of training data is not static, it is dynamically influenced by the learning itself. Each update may lead to different explorations and thus to a possibly different distribution of the data. Learning from dynamically changing observations is usually much more unstable than learning from a fixed set. Furthermore, especially in realistic scenarios, no deterministic transition function between the states of the environment can be assumed. This is due to various possibilities of uncertainties, such as unobservable variables or external factors. Beyond self-contained toy examples, it must therefore be assumed that the same action in the same observed state does not necessarily lead to the same result. Large updates due to fewer observations therefore also potentially lead to instability. An advantage of reinforcement learning here is that if the right hyperparameters are found, potentially good policies can also be learned in stochastic and noisy environments. However, finding the hyperparameters that are ideal for exploration is another challenge and the success of a reinforcement learning solution is potentially unstable depending on the parameters used.

Another challenge is the assignment of rewards. To be able to learn successfully, the agent needs negative or positive feedback after an action and reinforcement learning works best with continuous feedback (Kulkarni et al. 2016). Sometimes, however, the desired result depends on a long sequence of actions without immediate reward and with each action required, the number of possible paths in the solution space increases. Thus, at some point, the agent needs some luck to discover the reward. However, local minima may also be discovered first and if more and more existing knowledge is exploited, the agent may begin to develop more in this direction and the targeted goal may not be achieved. In addition, the landscape of the value function may look very different than assumed by the reward function. The algorithm optimizes the future cumulative reward and not necessarily the immediate reward. However, the reward function defines the immediate value. The difference between the immediate landscape of the reward function and the landscape of the value function can quickly become unintuitive, leading to unexpected behavior. In the worst case, reward hacking can occur, which means that the agent learns to do something completely different from what was intended when defining the reward function (Amodei et al. 2016). Current procedures show the possible influence of these challenges on the unpredictability of a result. Especially the choice of hyperparameters has a strong influence, at the same time the reason is often not always obvious and explainable. This has serious influences on the repeatability and controllability of experiments but also on the design of applications (Henderson et al. 2018). Especially when we try to use reinforcement learning as a method in the real world, where critical decisions, processes and infrastructures are affected, this leads to a serious flaw. Especially since the lack of explanation is not only given for the internal algorithms (usually deep neural networks), but often also for the configuration itself.

Developing Intelligent Agents – Solution Approaches

The development of intelligent agents through machine learning techniques such as reinforcement learning offers great potential for future applications. On the other hand, especially on the engineering side of entire applications, a major challenge is to control the potential variance of the algorithms and to ensure that the final result meets the requirements. The lack of controllability is given by a set of intrinsically rooted properties of the underlying algorithms discussed in the last section.

Given the possible future significance of the procedures, however, there are already a number of possible solutions within the community which have to be mentioned. It has been shown that one promising way to address these problems is to decompose the core problem into smaller problems. This can be done in the context of reinforcement learning in two different ways.

The first possibility is a more architectural approach: Hierarchical Reinforcement Learning. The core idea here is to divide complex tasks into subtasks, which can be learned more easily and reassembled later in further steps. From a software engineering point of view, Hierarchical Reinforcement Learning is a way of *Divide and Conquer* using reinforcement learning, which is often very intuitive. Especially in the area of robotics but also in other areas this methodology has shown success and helped to successfully implement more complex tasks (Gudimella et al. 2017; Meyer and Gruhn 2019; Kulkarni et al. 2016; Frans et al. 2017). The result of Hierarchical Reinforcement Learning is a set of individual models that are linked by a defined hierarchical structure.

The second option is Curriculum Learning. This is not only limited to reinforcement learning but has its origin in the supervised learning methods (Bengio et al. 2009). The core idea here is a decomposition of the learning process itself. A model is first trained in a simpler environment and then in increasingly complex environments. Like Hierarchical Reinforcement Learning, Curriculum Learning has been successfully applied and proven in many cases (Hacohen and Weinshall 2019; Florensa et al. 2017). The main difference is that here the learning process is subdivided and not the model itself.

Explainable AI

Machine learning processes are gaining importance in the development of software applications and are thus moving more and more from research to practice and industrially used applications. One of the big challenges today, which results especially from the real world, is the comprehensibility of machine learning models and the resulting decisions.

Many methods, especially in the field of deep learning, are often referred to as *black-box*. Although they are transparent at a technical level, their enormous internal complexity makes them virtually incomprehensible. The comprehensibility and explainability, however, is sometimes necessary for reasons of acceptance or possibly also for legal reasons. This leads to an increasingly important field of research in the field of *Explainable AI*, which is also called *XAI*. The XAI problem for models that would be considered as *black-boxes* can be divided into three subproblems: Model explainability, outcome explainability, and model inspection (Guidotti et al. 2019).

The problem of model explainability is about training another model that imitates the original model and is itself explainable. Decision trees are one of the essential models, which are considered to have a certain explainability because the learned rules are directly readable. However, this must be limited to simple trees and does not necessarily apply anymore to random forests. But even simple decision trees cannot be interpreted if they are sufficiently complex. Due to the principally explainable nature of the model, solving the problem by approximating the actual model with a simple decision tree is an essential approach in the literature. Here there are successful examples in current (van der Waa et al. 2018; Boz 2002; Johansson and Niklasson 2009) and even earlier literature (Craven and Shavlik 1996). The second frequently found approach is the extraction of rules from the model which cannot be explained itself. Rules are usually understandable for humans, although here it is also the case that this applies only to a limited set of rules. Current approaches show that rules can potentially be extracted from neural networks (Zilke, Mencía, and Janssen 2016) and even from networks with complex recurrent structures, such as LSTM cells (Murdoch and Szlam 2017).

The problem of outcome explainability deals with the explainability of a specific output for a given input. The overall explainability of the model is of secondary importance and does not necessarily have to be given. The most common approach is to calculate a mask over the actual input that identifies the subset that is mainly responsible for the calculated result. Most of the approaches work on images. These have the advantage that humans are particularly well able to visually interpret and deduce more complex data in this way. One way to determine the masking is to do this via backpropagation, which is called *Layer-wise Relevance Propagation* (Bach et al. 2015). Another possibility is to create *Attention-Maps* over the input (Xu et al. 2015; Fong and Vedaldi 2017). Since these approaches are strongly based on the model (in most cases neural networks), there are also further experiments of model agnostic methods. One of the successful is the method *Local Interpretable Model-agnostic Explanations (LIME)* (Ribeiro, Singh, and Guestrin 2016). The core idea is to generate examples from the neighborhood of the output, which can then be interpreted by humans. Similar ideas can be found for example in (Turner 2016). The problem of model inspection deals with the explainability and identification of certain characteristics of a model, such as the identification of certain neurons in a neural network that are responsible for certain tasks. Technically, similar approaches can be found as for the explanation of the outcome, but the goal here is to get a general overview of the model and why some inputs might work better or worse. There are currently three main approaches. The first is sensitivity analysis. Here it is examined to what extent uncertainty in the input influences the output (Saltelli 2002). Another possibility is to plot the partial dependency between features and the output. Due to the limited comprehension of humans, however, this must be limited to a small set (typically the most important one or two features). This approach gives information about the form of the influence on the task, e.g. whether it is linear, quadratic or in another form (Friedman 2001). The third frequently found way is *Activation Maximization (AM)*, which is very similar to the approaches that can be found in explaining the outcome and determining an importance mask over the input. Here, however, the intermediate layers are also visualized (typically as an image) (Yosinski et al. 2015).

Concept-based Engineering and Decomposition

Idea and Context

Explainable AI currently works on the algorithm level, but there are other levels that are helpful. If you try to understand the actions of another human being, you don't usually do this on the technical level of neuron connections, but try to understand how certain concepts and goals led to a decision. Using the possibilities of Hierarchical Reinforcement Learning and Curriculum Learning, this level of understanding can also be made possible for intelligent agents without having to severely limit the complexity and capacity of the underlying learning algorithms. What we present here is an idea of how we can better combine knowledge engineering and state-of-the-art technologies of machine learning, especially reinforcement learning, to achieve comprehensibility not only at a more technical level but also at a more meaningful human level. To better define this idea, we first define the context in which machine learning applications typically come into play and are developed in the real world.

Machine learning is always a solution in software development projects if a certain aspect is not economical or good enough to implement based on rules (Hesenius et al. 2019). This is typically the case when the rules are not easy to define. In this case, the problems are usually problems that require tacit knowledge from the experts. Explicit knowledge can usually simply be converted into rules, tacit knowledge not.

If machine learning is used to solve these problems, expert knowledge is often no longer part of the solution. This makes sense because knowledge in these cases is usually not explicit, but what is discarded is that tacit knowledge can also be externalized. The result is not knowledge that can be described at the rule level, but more abstract knowledge that is also called conceptual knowledge. Conceptual knowledge is often too abstract to be transformed into rules, however, it is a powerful tool that can be used as a basis for decomposition, but is often simply underestimated and not used.

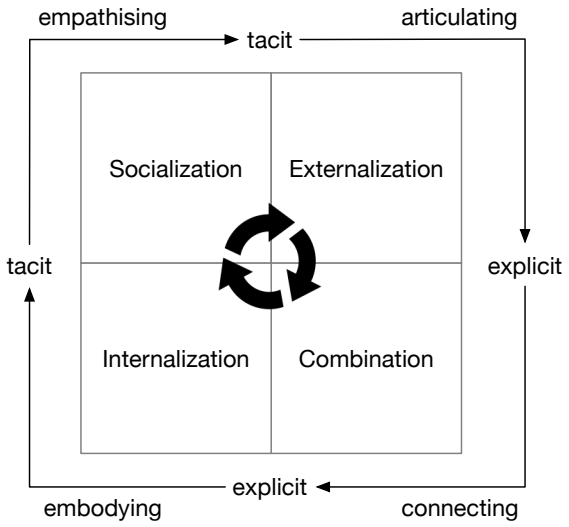

Figure 1: The SECI-Model of Knowledge Dimensions and Creation (Nonaka and Takeuchi 1995)

This is especially true when talking about ideas of complex long-term strategies. There are some successful application examples in the field of ontologies.

In knowledge management, a well-known model for the description of different knowledge generations and conversions is the SECI-model (figure 1) presented in (Nonaka and Takeuchi 1995).

Knowledge can be available in tacit and explicit form. In our experience, the boundary is fluid; stakeholders often have different intermediate states of knowledge. Since methods of machine learning and artificial intelligence in software development projects make sense when knowledge is not available in a sufficiently explicit form to convert it into acceptable rules at economically affordable costs, we assume at this point that knowledge in machine learning projects is, to a large extend, available to the stakeholders as tacit knowledge. Implicit or tacit knowledge is gained through the process of socialization, i.e. through observations, imitations, practice and participation in formal and informal communities, such as business environments (Yeh, Huang, and Yeh 2011). The conversion of tacit into explicit knowledge takes place through the process of externalization, whose main tool is articulation (Nonaka and Takeuchi 1995). Knowledge is not transformed directly (because it is implicit knowledge), but more in the form of abstract ideas. When knowledge carriers try to articulate tacit knowledge, this happens in the form of metaphors and analogies that sketch the knowledge. The resulting artifacts are called concepts. These are knowledge components that can be learned, which help to solve a given problem and over which we can form rules and conclusions that can be understood by humans. In a way, the rough concepts that are formulated describe a sketchy idea of what the possible solution might look like. For example, if you consider an area where most of us humans are experts, such as cycling, it is relatively difficult to describe the exact solution in the form of programmable rules, but if you speak on a conceptual level, it becomes increasingly easier to identify important elements and concepts. We know, for example, that it is important to drive at a certain speed in order not to tip over. At the same time, we know that it is important to have an idea of what the maximum speed is in order to drive through a curve safely. Finally, to be able to ride a bike, we also know that it is an important concept to set the right focus based on the situation, such as to accelerate in order not to fall over or to slow down in order not to crash in a curve. These are all concepts that are still abstract, but that can be easily identified by convincing domain experts to articulate their tacit knowledge. In software engineering, there are already a large number of established formats that can be used, for example, to gather and capture hidden requirements from domain experts. Capturing concepts is differently placed in the development process and more focused on modeling the solution rather than the requirement and problem space, however, these methods (which typically come as a workshop) can be easily adapted. Examples can be found for instance in (Grapenthin et al. 2013).

A more practical description

Since the idea of concept-based engineering is quite abstract, we want to show in the following a more concrete example of how this can look like in practice and how it can be used to better understand the behavior of intelligent agents. The example scenario is the optimization of a warehouse. There is a sales curve for the product to be managed, which is shown in figure 2. Due to various exogenous influencing factors, there are variances in the measured sales (several different warehouses with the same product serve as the basis here). Each week a delivery can be ordered to increase the stock again. However, there is a possibility that the delivery will fail due to delays, weather, strikes or other problems, despite the order being placed. Finding the optimal order quantity can therefore not simply be calculated using the expected sales curve, but must take into account storage costs and the risks of delivery failure. In addition, there are data on public holidays and the inclusion of the product in advertising flyers, which allow a more accurate estimate than just using the curve itself.

Concept Articulation and Elicitation Our approach starts with a short creative workshop based on (Grapenthin et al. 2013), in which all participants try to articulate their knowledge in the form of analogies, important ideas or expectations. This conforms to the phase of externalization according to the SECI model (Nonaka and Takeuchi 1995). The result are important concepts which are relevant for the given problem. These, for example, could be in this case (formulated on a human level):

1. *Concept of full demand fulfillment:* It is important to have an idea of which order is needed in order to satisfy each demand. This is important because it is not meaningful to order about this amount.
2. *Concept of complete sale:* It is important to be able to estimate how much will be sold until the end of the season.

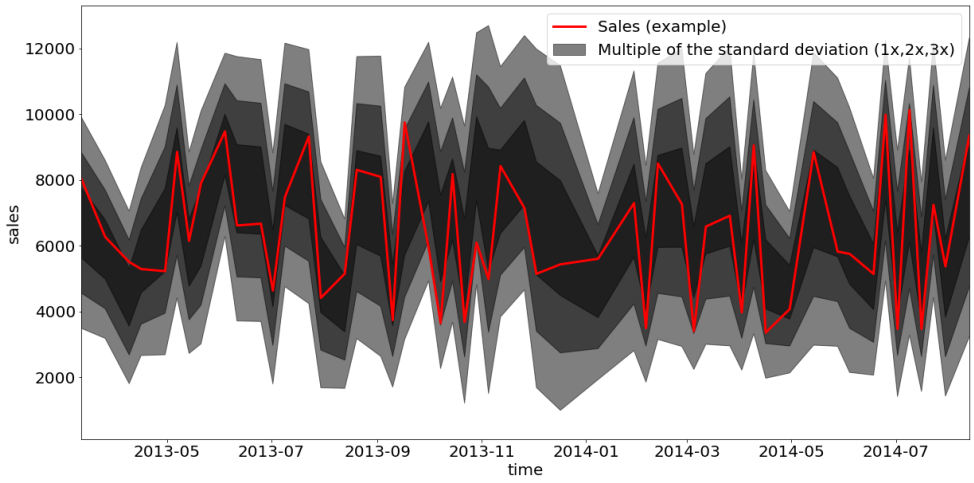

Figure 2: Exemplary sales curve and locally expected deviations for warehouse optimization

This is important because it does not make sense to order under this quantity.

3. *Concept of optimal balance:* If you have an idea of the quantity that will most likely always be sold and where the maximum is where every demand can be met, it is important to be able to find the right balance, which minimizes storage costs, etc. and at the same time maximizes customer satisfaction.

Furthermore, we can also capture statements about the environments of the concepts to be learned:

1. *Optimal Environment:* The Optimal Environment does not have any delivery failures. This environment is the simplest imaginable.
2. *Real Environment:* The real environment has delivery failures. The consideration of these should tend to increase order quantities and converge to the order quantities in an optimal environment if no failures are expected.

What should be emphasized here is that all this knowledge is simple knowledge that does not even need much experience in this field and can be formulated more or less by anyone who is more involved with it. In our experience, such abstract knowledge is relatively easy to find in many environments.

Concept Mapping and Combination

Knowledge, especially at the conceptual level, is often formulated very informally. In the second step, this must now be transferred to the technical space. The optimization of stock levels can be formulated as a Markov Decision Process. At each time t there is a condition which is described by the current stock, the date, as well as by further information like coming holidays and inclusion in advertising flyers. The action area is continuous and consists of the selection of the requested goods. Goods are sold every step of the way. The sales quantity follows a distribution based on historical data. With a given probability, the delivery is canceled and the order is not implemented and must be balanced by the goods in stock or the subsequent order (this essentially results in the temporary order problem).

The question is, how are the identified concepts and environmental variations reflected on a technical level? There are different possibilities here. The following is a non-exhaustive list that should to be filled in the future research:

- *Modified Reward Functions:* The goal can be changed by another reward function
- *Different environmental parameters:* Individual characteristics of the environments can, for example, be switched off
- *Modified observations:* Some observations can be supplemented or removed
- *Modified actions:* The action space can be simplified or extended
- ...

In the example used, we can do the mapping as follows:

- *Concept of full demand fulfillment:* Since the sales volume is generated by a probability distribution, there is no upper bound value here. However, the concept can be represented by an environment in which the reward function weighs storage costs very low and costs for the non-fulfillment of demands very high.
- *Concept of complete sale:* The same applies here with reversed weighting.
- *Concept of optimal balance:* Here the weighting in the reward function is carried out realistically. In addition, the observation is enriched by the decision of the other two concepts.

- *Optimal Environment:* This is the original environment with a different parameter: the default probability for deliveries is zero.
- *Real Environment:* This is the original environment.

The result of the mapping of concepts captured from the business point of view to the technical level is a decomposition on two levels, which is shown for the example used here in figure 3. First in the form of concepts, which represent stand-alone or hierarchical tasks and further in the form of different environments, which represent the training ground. The first is what is trained in the literature for example by Hierarchical Reinforcement Learning. The latter is referred to in the literature as Curriculum Learning, where the complexity of training environments is increasing. Using these methods, we can now train the identified concepts in a technical way. The result, in this case, are six artifacts: In each case, one concept (agent) trained on the optimal environment and in each case, the continued version trained on the realistic environment. Since the goal of this position paper is not the concrete selection and configuration of the algorithms, the exact execution is not considered here and it is assumed that the training has been successfully completed in order to focus on the explainability part. Details on how to train agents and machine learning models through Hierarchical Reinforcement Learning and Curriculum Learning can be found here (Gudimella et al. 2017; Meyer and Gruhn 2019; Kulkarni et al. 2016; Frans et al. 2017) or here (Bengio et al. 2009; Hacohen and Weinshall 2019; Florensa et al. 2017).

Using Concepts to Explain Decisions and Behavior

The resulting artifacts in the form of hierarchical sub-agents and intermediate training results according to different environments or curriculum steps. The individual models are still *black-boxes*, but the big advantage is that we know exactly how they interact or should have evolved. This fact can now be used at this point to understand the decisions made and to explain what otherwise would not have been possible without further effort. To do this is not further complex. It is only important to have an understanding of what the technical intention behind the individual concepts and training steps was. That is why it is so important to create them as described by a knowledge-driven process. With the idea of how the concepts are related based on the existing conceptual knowledge and when they are expected to interact differently or in other ways, questions can now be posed to the model and can be answered. We now show this in our example in two cases.

1. *Question 1: Does the agent estimate delivery failures as probable?* There are two versions of the final agent due to the decomposition that was performed. First the version trained on the optimal environment and then the version trained on the real environment (Curriculum Learning). We also know from human knowledge that as the probability of delivery failure increases, the order quantity must be increased in order to continue to meet demand. As a result, the difference between the two versions is an optimal measure for this question: $q_1(x_t) = optimalbalance_{realenv}(x_t) - optimalbalance_{optimalenv}(x_t)$, where in this case a higher value describes a higher expectation of a delivery failure.
2. *How certain is the agent that his order is actually being sold?* One of the most important concepts identified was to be able to estimate what is approximately the maximum order quantity that must be placed to meet any demand. If the agent is sure he can sell everything, the more he must tend to actually place this maximum order quantity. So here we can formulate the following score: $q_2(x_t) = fulldemand_{realenv}(x_t) - optimalbalance_{realenv}$, where a smaller value describes a higher certainty.

By using the identified concepts, we are able to visualize and represent the decisions of the intelligent agent at any time. The result of the questions can be easily visualized. Since the exact value range of the respective results does not have to be known intuitively, we use a z-score to standardize the results with $z(x_t) = \frac{x_t - \mathbb{E}[x]}{\sigma(x)^2}$. This allowed us to simply map the results to a color range and display them in an appropriate visualization (a really simple prototype can be seen in Figure 4). In contrast to most of the related work in the field of XAI, it is not necessary to explain the algorithms themselves and the capabilities of the algorithms themselves do not have to be restricted.

Conclusion and Future Work

In this position paper, we discussed how we use a concept-based engineering approach to explain the decisions and behavior of intelligent agents based on reinforcement learning. Decomposing the agent's training into concepts based on tacit knowledge of domain experts simplifies development and reveals options to inject domain-specific knowledge into the agent. The defined and trained concepts can be used to reason over the agent's behavior and its decisions as their interrelations and interactions are known. The motivated idea differs from most current approaches to explainable AI systems (XAI) in that it essentially does not operate on a purely technical level, but rather on the conceptual level known to humans. This makes this approach on the one hand more meaningful and on the other hand enables the unrestricted use of complex machine learning methods, such as reinforcement learning, since no special requirements are placed on the algorithms.

Since this is primarily a position paper, the work is still at an early stage. However, we hope to encourage the community to take further steps in this direction and see this approach as a good complement to the more technical solutions currently available for XAI.

For future work, several interesting tasks remain. First attempts to merge typical software engineering and machine learning development activities into combined process models have been published recently (e.g. Hesenius et al. (2019) or Amershi et al. (2019)), which aim to guide developers

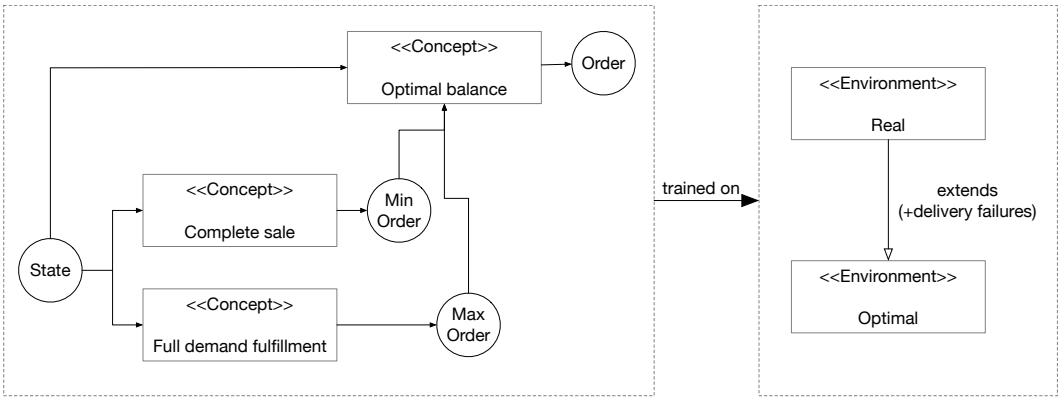

Figure 3: Articulated concepts and environments and their final technical mapping

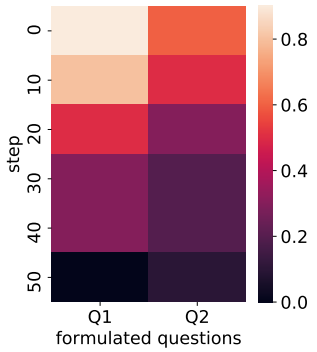

Figure 4: Simple visualization of some example states of a trained agent by questions formulated using previously identified concepts.

through the overall development cycle. However, they tend to define necessary activities on a rather abstract level, we thus plan to develop a more specific approach to create intelligent agents. Furthermore, we aim to develop tools and artifacts that support teams in deriving the necessary knowledge to train algorithms and to identify helpful concepts.

References

Abbeel, P.; Coates, A.; Quigley, M.; and Ng, A. Y. 2007. An application of reinforcement learning to aerobatic helicopter flight. In *Advances in neural information processing systems*, 1–8.

Amershi, S.; Begel, A.; Bird, C.; DeLine, R.; Gall, H.; Kamar, E.; Nagappan, N.; Nushi, B.; and Zimmermann, T. 2019. Software engineering for machine learning: A case study. In *Proceedings of the 41st International Conference on Software Engineering: Software Engineering in Practice*, ICSE-SEIP '19, 291–300. Piscataway, NJ, USA: IEEE Press.

Amodei, D.; Olah, C.; Steinhardt, J.; Christiano, P.; Schulman, J.; and Man, D. 2016. Concrete Problems in AI Safety. *arXiv:1606.06565 [cs]*. arXiv: 1606.06565.

Bach, S.; Binder, A.; Montavon, G.; Klauschen, F.; Müller, K.-R.; and Samek, W. 2015. On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation. *PloS one* 10(7):e0130140.

Bengio, Y.; Louradour, J.; Collobert, R.; and Weston, J. 2009. Curriculum Learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, 41–48. New York, NY, USA: ACM. event-place: Montreal, Quebec, Canada.

Boz, O. 2002. Extracting decision trees from trained neural networks. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, 456–461. ACM.

Craven, M., and Shavlik, J. W. 1996. Extracting tree-structured representations of trained networks. In *Advances in neural information processing systems*, 24–30.

Florensa, C.; Held, D.; Wulfmeier, M.; Zhang, M.; and Abbeel, P. 2017. Reverse Curriculum Generation for Reinforcement Learning. *arXiv:1707.05300 [cs]*. arXiv: 1707.05300.

Fong, R. C., and Vedaldi, A. 2017. Interpretable explanations of black boxes by meaningful perturbation. In *Proceedings of the IEEE International Conference on Computer Vision*, 3429–3437.

Frans, K.; Ho, J.; Chen, X.; Abbeel, P.; and Schulman, J. 2017. Meta learning shared hierarchies. *arXiv preprint arXiv:1710.09767*.

Friedman, J. H. 2001. Greedy function approximation: a gradient boosting machine. *Annals of statistics* 1189–1232.

Grapenthin, S.; Book, M.; Gruhn, V.; Schneider, C.; and Vlker, K. 2013. Reducing Complexity Using an Interaction Room: An Experience Report. In *Proceedings of the 31st ACM International Conference on Design of Communication*, SIGDOC '13, 71–76. New York, NY, USA: ACM. event-place: Greenville, North Carolina, USA.

Grondman, I.; Busoniu, L.; Lopes, G. A.; and Babuska, R. 2012. A survey of actor-critic reinforcement learning: Standard and natural policy gradients. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 42(6):1291–1307.

Gudimella, A.; Story, R.; Shaker, M.; Kong, R.; Brown, M.; Shnayder, V.; and Campos, M. 2017. Deep Reinforcement Learning for Dexterous Manipulation with Concept Networks. *arXiv preprint arXiv:1709.06977*.

Guidotti, R.; Monreale, A.; Ruggieri, S.; Turini, F.; Giannotti, F.; and Pedreschi, D. 2019. A survey of methods for explaining black box models. *ACM computing surveys (CSUR)* 51(5):93.

Hacohen, G., and Weinshall, D. 2019. On The Power of Curriculum Learning in Training Deep Networks. *arXiv:1904.03626 [cs, stat]*. arXiv: 1904.03626.

Henderson, P.; Islam, R.; Bachman, P.; Pineau, J.; Precup, D.; and Meger, D. 2018. Deep reinforcement learning that matters. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

Hesenius, M.; Schwenzfeier, N.; Meyer, O.; Koop, W.; and Gruhn, V. 2019. Towards a software engineering process for developing data-driven applications. In *Proceedings of the 7th International Workshop on Realizing Artificial Intelligence Synergies in Software Engineering*, RAISE '19, 35–41. Piscataway, NJ, USA: IEEE Press.

Johansson, U., and Niklasson, L. 2009. Evolving decision trees using oracle guides. In *2009 IEEE Symposium on Computational Intelligence and Data Mining*, 238–244. IEEE.

Kaelbling, L. P.; Littman, M. L.; and Cassandra, A. R. 1998. Planning and Acting in Partially Observable Stochastic Domains. *Artif. Intell.* 101(1-2):99–134.

Kulkarni, T. D.; Narasimhan, K.; Saeedi, A.; and Tenenbaum, J. 2016. Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. In *Advances in neural information processing systems*, 3675–3683.

Lillicrap, T. P.; Hunt, J. J.; Pritzel, A.; Heess, N.; Erez, T.; Tassa, Y.; Silver, D.; and Wierstra, D. 2015. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*.

Meyer, O., and Gruhn, V. 2019. Towards Concept Based Software Engineering for Intelligent Agents. In *Proceedings of the 7th International Workshop on Realizing Artificial Intelligence Synergies in Software Engineering*, RAISE '19, 42–48. Piscataway, NJ, USA: IEEE Press. event-place: Montreal, Quebec, Canada.

Murdoch, W. J., and Szlam, A. 2017. Automatic rule extraction from long short term memory networks. *arXiv preprint arXiv:1702.02540*.

Ng, A. Y.; Coates, A.; Diel, M.; Ganapathi, V.; Schulte, J.; Tse, B.; Berger, E.; and Liang, E. 2006. Autonomous inverted helicopter flight via reinforcement learning. In *Experimental Robotics IX*. Springer. 363–372.

Nonaka, I., and Takeuchi, H. 1995. *The knowledge-creating company: How Japanese companies create the dynamics of innovation*. Oxford university press.

Parbhoo, S.; Bogojeska, J.; Zazzi, M.; Roth, V.; and Doshi-Velez, F. 2017. Combining Kernel and Model Based Learning for HIV Therapy Selection. *AMIA Summits on Translational Science Proceedings* 2017:239–248.

Pyeatt, L. D., and Howe, A. E. 2001. Decision Tree Function Approximation in Reinforcement Learning. In *Proceedings of the third international symposium on adaptive systems: evolutionary computation and probabilistic graphical models*, volume 2, 70–77.

Ribeiro, M. T.; Singh, S.; and Guestrin, C. 2016. Why should i trust you?: Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, 1135–1144. ACM.

Saltelli, A. 2002. Sensitivity analysis for importance assessment. *Risk analysis* 22(3):579–590.

Schulman, J.; Levine, S.; Abbeel, P.; Jordan, M.; and Moritz, P. 2015. Trust region policy optimization. In *International Conference on Machine Learning*, 1889–1897.

Schulman, J.; Wolski, F.; Dhariwal, P.; Radford, A.; and Klimov, O. 2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

Silver, D.; Hubert, T.; Schrittwieser, J.; Antonoglou, I.; Lai, M.; Guez, A.; Lanctot, M.; Sifre, L.; Kumaran, D.; Graepel, T.; Lillicrap, T. P.; Simonyan, K.; and Hassabis, D. 2017a. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *CoRR* abs/1712.01815.

Silver, D.; Schrittwieser, J.; Simonyan, K.; Antonoglou, I.; Huang, A.; Guez, A.; Hubert, T.; Baker, L.; Lai, M.; Bolton, A.; Chen, Y.; Lillicrap, T.; Hui, F.; Sifre, L.; van den Driessche, G.; Graepel, T.; and Hassabis, D. 2017b. Mastering the game of Go without human knowledge. *Nature* 550(7676):354–359.

Simard, P. Y.; Amershi, S.; Chickering, D. M.; Pelton, A. E.; Ghorashi, S.; Meek, C.; Ramos, G.; Suh, J.; Verwey, J.; Wang, M.; and Wernsing, J. 2017. Machine Teaching: A New Paradigm for Building Machine Learning Systems. *arXiv preprint arXiv:1707.06742*.

Turner, R. 2016. A model explanation system. In *2016 IEEE 26th International Workshop on Machine Learning for Signal Processing (MLSP)*, 1–6. IEEE.

van der Waa, J.; Robeer, M.; van Diggelen, J.; Brinkhuis, M.; and Neerincx, M. 2018. Contrastive explanations with local foil trees. *arXiv preprint arXiv:1806.07470*.

Xu, K.; Ba, J.; Kiros, R.; Cho, K.; Courville, A.; Salakhudinov, R.; Zemel, R.; and Bengio, Y. 2015. Show, attend and tell: Neural image caption generation with visual attention. In *International conference on machine learning*, 2048–2057.

Yeh, Y.-c.; Huang, L.-y.; and Yeh, Y.-l. 2011. Knowledge management in blended learning: Effects on professional development in creativity instruction. *Computers & Education* 56(1):146–156.

Yosinski, J.; Clune, J.; Nguyen, A.; Fuchs, T.; and Lipson, H. 2015. Understanding neural networks through deep visualization. *arXiv preprint arXiv:1506.06579*.

Zilke, J. R.; Mencía, E. L.; and Janssen, F. 2016. Deepred–rule extraction from deep neural networks. In *International Conference on Discovery Science*, 457–473. Springer.