Probabilistic Logic Guided Soft Sensor Learning via Reward Prediction in Markov Decision Processes

Lernen von Soft-Sensoren durch Belohnungsvorhersage in Markov-Entscheidungsprozessen mittels probabilistischer Logik

Bachelor thesis in the department of Computer Science by Marcus Kornmann
(Student ID: 2439750)
Date of submission: October 16, 2024

1. Review: Prof. Dr. Kristian Kersting
2. Review: Simon Kohaut M.Sc.
Darmstadt

TECHNISCHE UNIVERSITÄT DARMSTADT

Computer Science Department

Artificial Intelligence and Machine Learning Lab

Erklärung zur Abschlussarbeit gemäß § 22 Abs. 7 APB TU Darmstadt

Hiermit erkläre ich, Marcus Kornmann, dass ich die vorliegende Arbeit gemäß § 22 Abs. 7 APB der TU Darmstadt selbstständig, ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt habe. Ich habe mit Ausnahme der zitierten Literatur und anderer in der Arbeit genannter Quellen keine fremden Hilfsmittel benutzt. Die von mir bei der Anfertigung dieser wissenschaftlichen Arbeit wörtlich oder inhaltlich benutzte Literatur und alle anderen Quellen habe ich im Text deutlich gekennzeichnet und gesondert aufgeführt. Dies gilt auch für Quellen oder Hilfsmittel aus dem Internet.

Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Mir ist bekannt, dass im Falle eines Plagiats (§ 38 Abs. 2 APB) ein Täuschungsversuch vorliegt, der dazu führt, dass die Arbeit mit 5,0 bewertet und damit ein Prüfungsversuch verbraucht wird. Abschlussarbeiten dürfen nur einmal wiederholt werden.

Bei einer Thesis des Fachbereichs Architektur entspricht die eingereichte elektronische Fassung dem vorgestellten Modell und den vorgelegten Plänen.

Darmstadt, 16. Oktober 2024

M. Kornmann

Abstract

This thesis introduces an end-to-end learning framework for the development of soft sensors, which infer hidden variables from available data within Markov Decision Processes (MDPs), offering a novel approach to enhance automation, data efficiency, and environmental independence. Unlike traditional methods that rely heavily on isolated learning and extensive labeled data for each sensor, our framework employs a strategy centered around reward prediction from trajectories, reducing the dependency on labeled data and computational resources. Our methodology is based on the integration of soft sensors into the reward prediction mechanism through the use of customizable probabilistic logic components that describe state transitions and reward formations. This integration of domain expertise and deep learning techniques allows for simultaneous learning of multiple soft sensors in a multi-task learning fashion. The effectiveness of our framework is demonstrated through its application to the Frozen Lake game, where we successfully design and train sensors capable of extracting and interpreting complex information from game states represented as images.

Contents

1 Introduction

Soft sensors, virtual sensors that use data obtained from other sources to predict hidden variables, have become very popular, particularly within the context of enhancing automation, operational efficiency, and the overall quality of products [1], as well as in process monitoring and fault detection [2]. Soft sensors constitute an interesting approach for integrating machine learning algorithms with domain-specific expert knowledge [3].

Soft sensors in the industrial sector typically rely on numerical data from physical sensors. In contrast, soft sensors for tasks like real-time environmental interpretation, such as in autonomous driving, handle visual data and often integrate this with information from additional sources, like LiDAR [4]. By analyzing visual inputs, these sensors can for example detect obstacles or recognize traffic signals, enhancing navigational decisions. While traditional approaches to sensor training often focus on isolated learning, recent advances suggest a shift towards end-to-end learning methods [5, 6].

This transition to end-to-end approaches also addresses the limitations of training soft sensors individually, which typically requires large amounts of labeled data, computational resources and sensor-specific domain knowledge to structure the learning problem effectively. By utilizing Markov Decision Processes (MDP) to model the environment, our approach simplifies the need for labeled data to just reward prediction, focusing on trajectories and their associated rewards. This strategy allows for integrating soft sensors into the reward prediction mechanism, eliminating the requirement for individually labeled data for each sensor.

In this work, a model capable of accurately predicting rewards from given trajectories is constructed. Under the premise that the reward function encapsulates comprehensive environmental information, it becomes possible to concurrently learn various sensors in the background. The primary objective of this thesis is to learn soft sensors through the predictive analysis of rewards. To achieve this, the state-of-the-art framework DeepProbLog [7] is used to embed sensors in expert knowledge formulated in probabilistic logic, leveraging both domain knowledge and deep learning techniques.

We propose a framework designed to learn a plurality of soft sensors simultaneously in an end-to-end fashion. Using a set of trajectories or the underlying MDP, rules for each unique reward are crafted that make use of the sensors. This setup leverages the sensors for reward prediction, making sure they evolve through the learning process themselves. Within this framework, soft sensors can have multiple input data sources of any type. We successfully show the effectiveness of the method in the Frozen Lake game, where we design and learn sensors capable of discerning information from game states depicted as images.

The remainder of this thesis is structured as follows. The next chapter provides an overview of the relevant literature and situates our work within the broader academic context. Following this, we delve into the *Foundations* chapter, discussing the essential mathematical background and introduce DeepProbLog, a probabilistic logic programming language that combines deep learning with probabilistic logic programming. The *Methods* chapter outlines the models deployed in this thesis, offering insights into the methodologies that underpin the work. Moving forward to the *Experiments* chapter, we showcase practical applications and demonstrate the effectiveness of our approach. In the concluding chapters, we engage in a discussion of our findings and the limitations of our method. Finally, we propose directions for future studies, paving the way for further exploration in this field.

2 Related Work

2.1 Soft Sensor Learning

A soft sensor or a virtual sensor is a software-based system that uses data obtained from other sensors to calculate or predict some variable that is not directly measured by a physical sensor [8]. This type of sensor is often used in environments where hardware sensor deployment is not possible or too costly to predict physical quantities in an online manner [9].

Soft sensors can be divided into two classes: model-driven sensors that depend on the (e.g. physical) background of processes and data-driven processes that solely rely on the output produced by other sensors in the process [8]. By utilizing image data alongside positions calculated through a transition model, our methodology integrates data-driven processes with the contextual understanding of model-driven sensors.

The applications of soft sensors range from measuring the process output quality over the estimation of hidden variables in biological or chemical processes to water quality monitoring. Liu et al. use a Hidden Markov Model with Principal Component Analysis and the Fireworks Algorithm to build a soft sensor for predicting the acrolein conversion in the production of acrylic acid [10]. In this work, we craft soft sensors that use a transition model and visual representations of a game to detect properties of the current game state.

Dilmi uses Support Vector Regression, Extreme Learning Machine, Elman Recurrent Neural Network, and Multilayer Perceptron models combined with an infinite impulse response digital-low-pass filter to estimate the calcium concentration of water [11]. Jia et al. use a graph convolutional network to find localized spatial–temporal correlations to understand the process variables and enhance the process quality [12]. Unlike the work stated before, in this thesis, we train multiple soft sensors simultaneously using a single dataset of trajectories from a Markov Decision Process. The approach, which integrates

sensors within probabilistic logic to predict trajectory rewards, trains the soft sensors while learning the reward predictor.

2.2 Image De-Rendering

De-rendering is part of the larger class of ”image-to-text” problems dealing with methods mapping an input image to some sequence of output tokens, i.e. it is the process of converting an image to a domain-specific language that can be easily changed, stored, or compared to other images [13].

It has been applied for the task of raster text editing in display media using a vectorization model for parsing and reconstructing text, style, and background parameters to generate resolution-independent output [14]. Similarly, in this thesis, we need to localize and extract meaningful information from 2D images. However, our approach differs. We employ soft sensors, each designed to extract dedicated features, embedded within a probabilistic logic framework.

Different publications use image de-rendering in 3D object reconstruction from single images. Wimbauer et al. use a weakly supervised method that only relies on a rough initial shape that can decompose a single image of an object into shape material and global lighting parameters to de-render 2D images into a decomposed 3D representation [15]. Pavllo et al. use a hybrid approach of Neural Radiance Fields and GANs without the need for multiple views or ground-truth poses to reconstruct 3D shapes, poses, and appearance from single images [16]. Wu et al. introduce HyperDreamer which generates high-resolution, fully-viewable, and renderable 3D models, using 360° mesh modeling, fine-grained semantic segmentation, and data-driven priors that are editable using text-based guidance [17]. In this thesis, we are interested in crafting sensors that can detect specific properties from a visual 2D game state representation, e.g. the location of obstacles. By focusing on 2D representations, our approach bypasses the complexities of 3D reconstruction, allowing for a more focused approach to extract actionable insights relevant to game strategy and decision-making.

Image de-rendering has also been applied to simultaneously estimate the illumination and reflectance from a single image using a diffusion model [18] or to convert sketches to objects, for example by using a dual-modality, two-way reinforcement learning framework for training models [13].

2.3 Visual Relationship and Attribute Detection

Visual Relationship Detection is the technique of extracting relationships in the form of <subject, predicate, object> triplets from images. Plesse et al. combine visual, semantic, and spatial features in a fully connected layer to predict relationships based on the nearest neighbors of the query, training their model by maintaining a set of prototypes that is updated for each pair of objects. They rely on datasets that already provide bounding boxes for objects [19]. In this work, a transition model in probabilistic logic is used to generate positional information for objects of interest in an image. We employ a set of soft sensors that use positional and spatial features to detect relationships between objects in the game to the game board and thereby also relationships between the objects.

Zhu et al. use a method that combines object detection utilizing Faster R-CNN with a feature- and label-level prediction, forming the relationship prediction [20]. Zhao et al. employ a bottom-up approach, utilizing a Vision Transformer for object detection to generate embeddings, and a feed-forward network for determining bounding boxes. Subsequently, it leverages a transformer decoder to predict interactions between object pairs, producing relationship embeddings that are then classified to determine the type of and localize the relationship [21]. Anastasakis et al. propose an approach combining self-supervised learning with Masked Bounding Box Reconstruction for learning object relationships, and a fine-tuning phase integrating encoder outputs with word and spatial embeddings for predicate class prediction [22]. Unlike the other work, where relationships are explicit outputs, in this work, each soft sensor represents a single type of relationship.

Besides predicting the relationship triplets, Linag et al. also predict attributes of objects as tuples (object, attribute). They use a sequential decision-making process based on a semantic action graph and Faster R-CNN for object detection. The model is trained using a deep Q-Network for selecting actions in a model-environment [23]. While our soft sensors hold information about the relationships between objects and the game board, they also hold attribute information about objects. A sensor that detects whether an agent reached the goal, it captures a relationship between the agent and the environment but also describes the agent's victory attribute.

2.4 Constrained Reinforcement Learning

Constrained Reinforcement Learning (CRL) is an extension of traditional Reinforcement Learning. The goal remains to learn an optimal policy but additionally constraints are introduced that restrict the agent from taking certain actions in specific states, making it achieve its objective within a defined set of rules.

A common field of application of CRL is the implementation of safety constraints that prevent an agent from taking hazardous actions like walking into an obstacle or moving in a way that could cause self-damage, especially during the exploration phase [24]. One way of achieving this goal is shielding, introduced by Jansen et al. Their method restricts agents from taking actions with a risk of safety violations, unless the probability for a safety issue, determined by probabilistic temporal logic constraints, is less than a state-dependent threshold γ. The method is applied to PAC-MAN and a service robot [24]. Hunt et al. go one step further and apply shielding in a continuous state space by incorporating a neural learner [25]. Finally, Yang et al. enforce safety directly in the policy. Therefore, they denote safety specifications as background knowledge and encode the interaction with the agent in a probabilistic logic program that can be compiled into a differentiable structure with optimizability for a single loss function [26]. In this work, we employ standard reinforcement learning for data collection, subsequently training a deep learning model that integrates constraints through probabilistic logic to train a set of soft sensors. Assuming the sensors are trained on a set of collected trajectories, the final sensors can be used to better detect safety hazards when training a real agent and their outputs can be used to define constraints.

CRL has also been applied in other fields. For example, Ammanabrolu et al. apply it for solving Interactive Fiction games, tasks that require the agent to generate contextually relevant natural language by utilizing a knowledge graph-based state and a template-based action space [27]. Korivand et al. put constraints on the reward function by incorporating rewards from reference motion data from Inertial Moment Unit sensors and trajectory optimization rewards for simulating human locomotion more realistically [28]. To handle decentralized fault tolerance control in interconnected non-linear systems, Zhao et al. use reinforcement learning, introducing constraints through an improved cost function [21].

2.5 Multi-Task Learning

Muli-Task Learning (MTL) is a technique of learning a set of similar regression or classification tasks in parallel while using a shared representation [29, 30]. MTL has been widely applied across various computer science areas, notably in Natural Language Processing (NLP) [31, 32] and Computer Vision [33, 34]. Other work has leveraged a combination of NLP and visual tasks, for example for visual question answering, caption-based image retrieval, and visual grounding by using hierarchical representations with predictions made based on the representations of the corresponding hierarchy level [35] or by training a single model for all these tasks simultaneously on a dozen datasets [36].

MTL has been effectively applied to a diverse range of tasks, demonstrating its versatility. For instance, it has been used in medicine to identify substances that might interfere with enzymes critical for metabolizing drugs and toxins [37] or to analyze and integrate complex biological data from different cellular components, enhancing the understanding of gene regulation and biological diversity [38]. Furthermore, it has shown promise in anomaly detection in videos [39], multimodal sentiment analysis [40] and predicting traffic speed from incomplete real-world data [41].

In this work, an advanced approach is employed that goes beyond the traditional use of a single neural network for MTL. By leveraging shared global features while also accommodating task-specific features, this method enables a more nuanced and efficient learning process [42]. This dual-focus approach allows for the extraction and utilization of commonalities across tasks to enhance generalization, while simultaneously honing in on the unique aspects of each task to improve specificity and performance.

In MTL, Auxiliary Task Learning (ATL) involves incorporating additional tasks that are less important or even irrelevant. This approach utilizes a shared encoder to process input data, with separate decoders dedicated to each task, including the auxiliary ones. The inclusion of these auxiliary tasks aims to enrich the model's learning, enhancing its ability to generalize across tasks by leveraging a broader spectrum of features extracted from the shared data representation [43]. In this work, we will adopt the concept of ATL by using a shared convolutional encoder and task-specific decoders to enhance our model's learning capability and generalization across tasks.

3 Foundations

3.1 Propositional Logic

Our exposition of propositional logic closely follows the treatment presented in [44], which readers may consult for a more in-depth exploration of the subject. In the context of propositional logic (PL), the discussion centers around tuples of truth values and the assignment of these values to PL variables. Specifically, in this framework, the truth values are binary: 0 denotes false, and 1 denotes true. These truth values are treated as constants in PL, sometimes represented by symbols such as $\top$ for true and $\bot$ for false. The variables in PL, typically denoted as $p, q, r, ..., p_1, p_2, ...$, form the foundation upon which formulas describing boolean conjunctions are built, thereby characterizing PL as the logic of these formulas.

Key symbols employed in PL include $\neg$, which is the junctor for negation and $\wedge, \vee$, which are the junctors for conjunction and disjunction respectively. Using these junctors, more junctors can be formed, for example, $\rightarrow$ defined as $p \rightarrow q := (\neg p \vee q)$ or $\leftrightarrow$ defined as $p \leftrightarrow q := ((\neg p \wedge \neg q) \vee (p \wedge q))$. Given a set of PL variables $\mathcal{V}$, the set of formulas over $\mathcal{V}$ denoted as AL($\mathcal{V}$) is constructed inductively:

- atomic formulas: 0, 1, p for any $p \in \mathcal{V}$
- negation: $\neg\varphi \in$ AL($\mathcal{V}$), if $\varphi \in$ AL($\mathcal{V}$)
- conjunction, disjunction: for $\varphi, \psi \in$ AL($\mathcal{V}$) also $(\varphi \wedge \psi)$, $(\varphi \vee \psi) \in$ AL($\mathcal{V}$)

A $\mathcal{V}$-Interpretation $\mathfrak{I}$, is a function from $\mathcal{V}$ to the boolean set $\mathbb{B} := \{0, 1\}$. It interprets each PL variable p as true if $\mathfrak{I}(p) = 1$ and false if $\mathfrak{I}(p) = 0$.

$$\begin{aligned} \mathfrak{I} : &\mathcal{V} \rightarrow \mathbb{B} \\ &p \mapsto \mathfrak{I}(p) \end{aligned} \qquad \mathfrak{I} \text{ interprets } p \text{ as } \begin{cases} \text{true, if } \mathfrak{I}(p) = 1 \\ \text{false, if } \mathfrak{I}(p) = 0 \end{cases}$$

For every formula $\varphi \in \mathrm{AL}(\mathcal{V})$, given a $\mathcal{V}$-Interpretation $\mathfrak{I}$, a truth value $\varphi^{\mathfrak{I}} \in \mathbb{B}$ is assigned. The assignment follows specific rules:

- atomic: $0^{\mathfrak{I}} := 0$, $1^{\mathfrak{I}} := 1$, $p^{\mathfrak{I}} := \mathfrak{I}(p)$
- negation: $(\neg\varphi)^{\mathfrak{I}} := 1 - \varphi^{\mathfrak{I}}$
- conjunction: $(\varphi \wedge \psi)^{\mathfrak{I}} := \min(\varphi^{\mathfrak{I}}, \psi^{\mathfrak{I}})$
- conjunction: $(\varphi \vee \psi)^{\mathfrak{I}} := \max(\varphi^{\mathfrak{I}}, \psi^{\mathfrak{I}})$

Truth tables are an essential method in propositional logic for displaying the truth value of formulas under a given $\mathcal{V}$-Interpretation $\mathfrak{I}$. They systematically exhibit how a formula's truth value changes with different combinations of truth values assigned to the variables in $\mathcal{V}$. Each row in a truth table represents a unique assignment of truth values to these variables, reflecting a possible $\mathfrak{I}$. The columns correspond to the formulas and their sub-formulas. This method allows for a clear visual representation of the way logical connectives within a formula interact under various interpretations.

A given $\mathcal{V}$-Interpretation $\mathfrak{I}$ satisfies φ, denoted as $\mathfrak{I} \models \varphi$, if and only if the interpretation of φ under $\mathfrak{I}$ is equal to 1, that is, $\varphi^{\mathfrak{I}} = 1$. The concept extends to sets of formulas Φ as well, where $\mathfrak{I}$ satisfies Φ, denoted as $\mathfrak{I} \models \Phi$, if for every formula $\varphi \in \Phi$: $\mathfrak{I} \models \varphi$. "$\models$" is called *model relation*. A formula φ is *satisfiable* if there exists at least one interpretation $\mathfrak{I}$ such that $\mathfrak{I} \models \varphi$. Similarly, a set of formulas Φ is satisfiable if there is an interpretation $\mathfrak{I}$ for which $\mathfrak{I} \models \varphi$ for every φ in Φ. The collection of all such satisfiable formulas is known as *SAT*.

When a $\mathcal{V}$-Interpretation $\mathfrak{I}$ satisfies a formula φ, we say that φ implies ψ, or $\varphi \models \psi$, if $\mathfrak{I} \models \varphi \Rightarrow \mathfrak{I} \models \psi$ for all $\mathfrak{I}$. This *logical implication* or *consequence* also applies to sets of formulas Φ such that $\Phi \models \psi$.

In the context of an infinite set of formulas Φ, the set is considered satisfiable if and only if every finite subset $\Phi_0 \subseteq \Phi$ is satisfiable. This principle is known as the *Compactness Theorem*, or the *Finiteness Theorem*, in propositional logic.

A formula is considered to be *universally valid*, or a *tautology*, if it is satisfied by every interpretation $\mathfrak{I}$. For example, a formula ψ is universally valid, expressed as $\models \psi$, if it is of the form $\psi = \neg\varphi \vee \varphi$, which is always true regardless of the interpretation.

Satisfiability is the dual notion to universal validity. Specifically, a formula φ is universally valid if and only if its negation $\neg\varphi$ is not satisfiable.

Two formulas φ and ψ are deemed logically equivalent, denoted as $\varphi \equiv \psi$, if for every interpretation $\mathfrak{I}$, $\mathfrak{I} \models \varphi$ if and only if $\mathfrak{I} \models \psi$.

To efficiently evaluate formulas, propositional logic employs two standard forms: *Disjunctive Normal Form* (DNF) and *Conjunctive Normal Form* (CNF). DNF is a disjunction (logical OR) of conjunctions (logical AND) of literals, while CNF is a conjunction of disjunctions of literals. Every formula in PL can be transformed into an equivalent formula in either DNF or CNF. The negation of a CNF is equivalent to a DNF and vice versa. Additionally, the conjunction over an empty set and the disjunction over an empty set are defined respectively as true ($\bigwedge \emptyset \equiv 1$) and false ($\bigvee \emptyset \equiv 0$), serving as logical identities for these operations. In addition to these forms, the concept of a *clause*, which is a finite disjunction of literals, plays a significant role in propositional logic. A specialized type of clause is the *Horn clause*, characterized by having at most one positive literal, e.g. $(\neg\varphi_1 \wedge \neg\varphi_2 \wedge \varphi_3)$. Horn clauses are noteworthy because the satisfiability of a set of Horn clauses can be determined efficiently.

Propositional logic, while powerful for analyzing and constructing logical arguments with finite and well-defined structures, has its limitations. Its primary constraint is the inability to express statements about objects or their properties, relationships and quantifiers (like ”all” or ”some”).

3.2 First Order Logic

Our discourse on first-order logic is structured upon the framework provided in [44], which serves as a reference for further details. First Order Logic (FOL), also known as Predicate Logic, extends Propositional Logic (PL) by incorporating predicates and quantification. This addition allows FOL to express properties of and relations between objects, overcoming the limitations faced by PL. It is powerful for deductive reasoning, allowing conclusions to be drawn from premises using rules of inference.

FOL talks about structures based on a Signature S that is a set of constants (like $c, d, e, \ldots$), a set of function symbols (such as $f, g, \ldots$), and a set of relation symbols (e.g., $P, Q, R, \ldots$), each designated with specific arities. These elements, along with a set of variables $V := \{x, y, z, \ldots, x_1, x_2, \ldots\}$, constitute the foundational vocabulary of FOL. A S-Structure $\mathcal{A}$ is defined as $\mathcal{A} := (A, c^{\mathcal{A}}, \ldots, f^{\mathcal{A}}, \ldots, R^{\mathcal{A}}, \ldots)$ where $A \neq \emptyset$ serves as the universe. Constant symbols $c \in S$ are interpreted as $c^{\mathcal{A}} \in A$, each n-ary function symbol $f \in S$

is represented as $f^{\mathcal{A}} : A^n \to A$, and each n-ary relation symbol $R \in S$ corresponds to $R^{\mathcal{A}} \subseteq A^n$.

Using the signature S and the set of variables V, the set of terms over S ($T(S)$) can be defined inductively:

- $x \in T(S)$ for each variable $x \in V$
- $c \in T(S)$ for every constant symbol $c \in S$
- if $f \in S$ is a n-ary function symbol, and if $t_1, \ldots, t_n \in T(S)$, then $ft_1, \ldots, t_n \in T(S)$

Besides the symbols used in signatures (constants, functions, relations) and the junctors used by PL ($\neg$, $\wedge$, $\vee$, $\to$, $\leftarrow$, $\leftrightarrow$), FOL also introduces the symbol $=$ for equality as well as the quantifier symbols $\forall$ (universal quantifier, "for all") and $\exists$ (existential quantifier, "there exists").

An Interpretation $\mathfrak{I}$ in FOL is defined as $\mathfrak{I} = (\mathcal{A}, \beta)$, using the S-Structure $\mathcal{A}$ and the assignment function $\beta : V \to A$. The function β is essential for determining the truth value of expressions. The notation $\mathfrak{I}[x \mapsto a] = (\mathcal{A}, \beta[x \mapsto a])$ describes a specific interpretation where the variable x is assigned the value $a \in A$. $\beta[x \mapsto a]$ is the modified assignment function, denoted as β' with $\beta'(y) = a$ if $y = x$ and $\beta'(y) = \beta(y)$ else.

The function $^{\mathfrak{I}} : \mathrm{FO}(S) \to \mathbb{B}$ adopts the same inductive definitions for negation, conjunction and disjunction as found in PL and extends these by atomic definitions for terms $t_i \in T(S)$ and definitions for quantifications:

- atomic: $(t_1 = t_2)^{\mathfrak{I}} = 1$ iff. $t_1^{\mathfrak{I}} = t_2^{\mathfrak{I}}$
  $(Rt_1 \ldots t_2)^{\mathfrak{I}} = 1$ iff. $(t_1^{\mathfrak{I}}, \ldots, t_n^{\mathfrak{I}}) \in R^{\mathcal{A}}$
- quantifications: $(\exists x\varphi)^{\mathfrak{I}} = \max(\varphi^{\mathfrak{I}[x \mapsto a]} : a \in A)$
  $(\forall x\varphi)^{\mathfrak{I}} = \min(\varphi^{\mathfrak{I}[x \mapsto a]} : a \in A)$

Just like in PL, an interpretation $\mathfrak{I}$ satisfies a formula φ if $\varphi^{\mathfrak{I}} = 1$. If $\mathfrak{I}$ satisfies a set, we write $\mathfrak{I} \models \Phi$. This concept, along with the rules for logical implication, universal validity, and logical equivalence in FOL, is analogous to those in PL. Additionally, FOL introduces the concept of *satisfiability equivalence*. Two formulas φ, φ' are called satisfiability equivalent if φ satisfiable if and only if φ' satisfiable. Analogous, this concept extends to sets of formulas.

The application of reduction techniques, where a set of formulas in FOL is translated into a set of formulas in PL, leads to the formulation of the Compactness Theorem for FOL.

This theorem in FOL is analogous to the Compactness Theorem in PL. Furthermore, the reduction approach gives an important duality: $\Phi \models \varphi$ iff. $\Phi \cup \{\neg\varphi\}$ not satisfiable.

While the satisfiability problem in Propositional Logic (SAT(PL)) is a decidable problem, SAT(FOL) is undecidable. This can be illustrated through a classic reduction from the undecidable Halting Problem for Turing machines to the problem of determining the satisfiability of sets in FOL. Essentially, if SAT(FOL) were decidable, it would provide a means to solve the Halting Problem, which is a well-known impossibility.

Despite its powerful expressiveness, FOL cannot articulate certain higher-level concepts that require more than one level of quantification, such as properties of sets, which are addressed by higher-order logics. Moreover, FOL is not able to handle uncertainty or probabilistic scenarios. In the upcoming sections, we will introduce the intersection of FOL with probabilistic reasoning.

3.3 ProbLog

ProbLog [45, 46, 47] extends the programming language Prolog with probabilities. A ProbLog program $\mathcal{P}$ consists of a finite set of probabilistic facts $\mathcal{F}$ and a finite set of rules $\mathcal{BK}$ in the form of clauses. A probabilistic fact $p_i :: f_i$ denotes the probability p_i of fact f_i being true. For example, $0.2 ::$ *goal_right* states that the probability of the goal being in the cell right of the agent is 0.2. It is assumed that probabilistic facts are independent of each other. Clauses are deterministic rules of the form $h :- b_1, \ldots, b_n$ where the literal h is true if all b_i are true. Atoms $q(t_1, \ldots, t_n)$ consist of a predicate q and the terms t_i. A literal is an atom or its negation $\neg q(t_1, \ldots, t_n)$. A term t can either be a constant c or a variable V.

To use multi-valued variables, to indicate the probability of the goal for each direction for example, one can use annotated disjunctions (ADs), a disjunction of atoms each annotated with a probability $p_1 :: h_1; \ldots; p_n :: h_n$. The heads h_i are mutually exclusive. In case of the goal, it could be

$0.05 ::$ *goal(left)*$; 0.15 ::$ *goal(down)*$; 0.20 ::$ *goal(right)*$; 0.15 ::$ *goal(up)*$; 0.45 ::$ *goal(none)*.

Inference

In our examination of ProbLog's inference process [45, 7], we will utilize a straightforward example centered around an agent navigating a grid world. The agent has a probability of 0.5 of making a move to an adjacent cell at each step. For this single move, the cell the agent lands on can have different outcomes: there's a 0.4 probability that the cell is the goal state and a 0.7 probability that it is a hole. Both the goal and the hole signify the end of the game and in these scenarios, the agent is rewarded. If the agent, however, lands on a plain cell, the game continues without a reward. For the implementation, see figure 3.1. Our primary query in this model is to determine the probability of the game concluding in this single step — specifically, the likelihood that the agent will land on a cell that is either the goal or a hole.

```
0.4 :: goal.
0.7 :: hole.
0.5 :: move.
reward :- goal.
reward :- hole.
game_over :- reward, move.
```

Figure 3.1: The grounded ProbLog Program

The Inference process in ProbLog is composed of four steps. First, the logic program is grounded with respect to the query, i.e. all ground instances the query depends on are created. In the example the grounded program is the same as the original program in figure 3.1. The second step rewrites the program into a propositional logic formula that establishes the veracity of the query by correlating it with the truth values of the probabilistic facts within the program. In our example, this formula is given as $\mathit{game\_over} \leftrightarrow \mathit{move} \wedge (\mathit{goal} \vee \mathit{hole})$. In the third step, the Sentential Decision Diagram (SDD) [48] for the query is compiled, which facilitates the efficient evaluation of the query. Figure 3.2 illustrates the SDD for the query *game_over*. In this diagram, the rounded gray rectangles represent variables corresponding to probabilistic facts. The red rounded rectangle at the top denotes the query atom, and the white rectangles are logical operators. The final step is the bottom-up evaluation of the SDD. In Figure 3.2, intermediate results of this evaluation are displayed in the rectangles adjacent to the nodes. As we can observe, the SDD features multiple paths leading to identical results, introducing a *noisy or* scenario that complicates discerning the specific contributions to

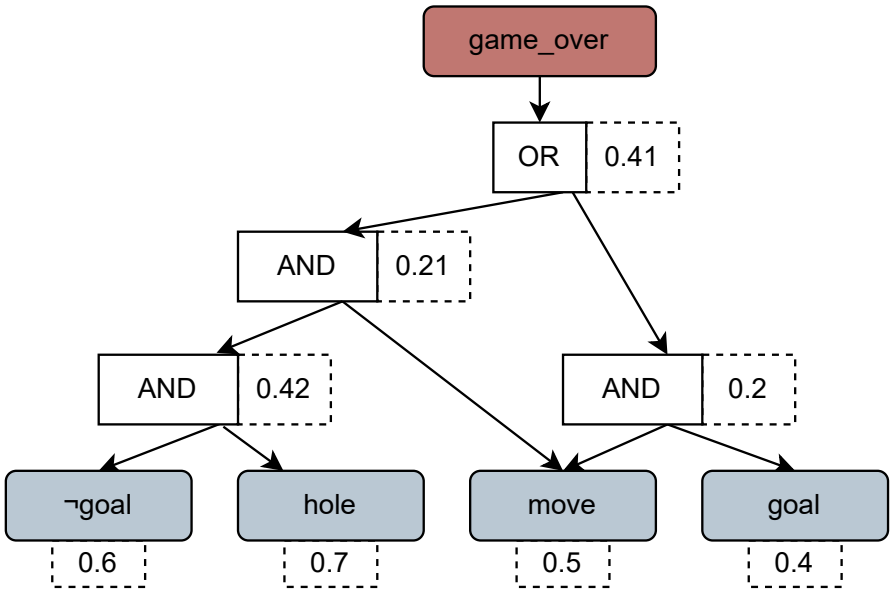


Figure 3.2: SDD for query *game_over*. The rounded gray rectangles represent variables corresponding to probabilistic facts. The red rounded rectangle at the top denotes the query atom, and the white rectangles are logical operators. During the bottom-up evaluation, the probability for the query is calculated based on the ground atoms using the probability semiring. *AND* corresponds to arithmetic multiplication, while *OR* corresponds to arithmetic addition of the probabilities. The intermediate results are shown in the dotted rectangles adjacent to the nodes.

the outcome. ProbLog employs the probability semiring, denoted as $(\mathbb{R}_{[0,1]}, +, \times, 0, 1)$, utilizing standard addition and multiplication to determine the likelihood of a query based on the SDD formulated for it. Instead of the probability semiring, it is also possible to use the log semiring $(\mathbb{R}_+ \cup \{\infty\}, \oplus_l, +, \infty, 0)$ with $a \oplus_l b := \log(e^a + e^b)$ and $+$ being the standard arithmetic addition.

3.4 aProbLog

Previously, it was mentioned that ProbLog operates on the probability semiring. In contrast, aProbLog can work with any commutative semiring. Rather than assigning probability labels to facts, aProbLog employs a labeling function to directly assign values from the selected semiring to facts as well as their negations and integrates these values through the use of semiring addition and multiplication within the SDD [7].

aProbLog [49] is a generalization of the ProbLog language that allows for labeling algebraic facts with a wide variety of elements from a semiring, including probability values, reals, polynomials, boolean functions, or data structures. It consists of a *commutative semiring* $(\mathcal{A}, \oplus, \otimes, e^{\oplus}, e^{\otimes})$, a finite set of *ground algebraic facts* $F = \{f_1, \ldots, f_n\}$, a finite set BK of *background knowledge clauses* that are definite clauses and a *labeling function* $\alpha : L(F) \to \mathcal{A}$.

For a given set of ground facts $\mathcal{J}$, the set of literals is defined as $L(\mathcal{J}) := \mathcal{J} \cup \{\neg f | f \in \mathcal{J}\}$. The set of interpretations is given as $\mathcal{I}(\mathcal{J}) := \{S | S \subseteq L(\mathcal{J}) \wedge \forall l \in \mathcal{J} : l \in S \leftrightarrow \neg l \notin S\}$. The label of a complete interpretation $I \in \mathcal{I}(F)$ is given as $\mathbf{A}(I) = \bigotimes_{l \in I} \alpha(l)$. For a given query q, the label is defined as $\mathbf{A}(q) = \mathbf{A}(\mathcal{I}(q)) = \bigoplus_{I \in \mathcal{I}(q)} \bigotimes_{l \in I} \alpha(l)$

3.5 Deep ProbLog

DeepProbLog [7] extends a ProbLog program by a set of ground neural annotated disjunctions (nADs). Consider a set of atoms $b_1, \ldots, b_m$, a vector of ground terms $\vec{t} = t_1, \ldots, t_k$ representing the inputs to the neural network, a predicate q and a vector of possible output values of the network $\vec{u} = u_2, \ldots, u_n$. A neural network with identifier m_q is defined as follows, using the nn notation for 'neural network':

$$nn(m_q, \vec{t}, \vec{u}) :: q(\vec{t}, u_1); \ldots; q(\vec{t}, u_n) :- b_1, \ldots, b_m$$

Within the DeepProbLog framework, neural networks can be utilized without restrictions on their architecture, type, or size. The principal stipulation is that the output layer must be normalized — typically through a softmax function for classification purposes — a condition that aligns with the networks implemented in this work.

The inference process in DeepProbLog is the same as in ProbLog, except when a neural predicate is encountered during grounding. In this case, the necessary inputs are fed

through the network and the normalized outputs are used as the probabilities of the ground AD.

As an example, we can use handwritten digit recognition. In this scenario, the network receives an image of a digit as input and transforms it into a probability distribution, representing a prediction of the digit depicted.

$$nn(m_{\text{pos}}, \boxed{3}, [0, 1, \ldots, 9]) :: \text{pos}(\boxed{3}, 0); \text{pos}(\boxed{3}, 1); \ldots; \text{pos}(\boxed{3}, 9).$$

Learning

Joint training of the parameters of probabilistic facts and neural networks in DeepProbLog programs happens in the *learning from entailment* [50] setting, that is, the model is trained on a set of logical queries with known probabilities of being true. Given a DeepProbLog Program with parameters θ, a set of query-probability pairs $\mathcal{Q} := \{(q_i, p_i)\}$, the goal is to find optimal parameters θ^* so that the loss function L is minimized:

$$\theta^* = \arg\min_{\theta} \frac{1}{|\mathcal{Q}|} \sum_{(q,p)\in\mathcal{Q}} L\left(P_\theta(q), p\right)$$

DeepProbLog, unlike ProbLog which utilizes the standard probability semiring with conventional addition and multiplication, employs the aProbLog extension integrated with a gradient semiring. In this semiring, each element is represented as a tuple $\left(p, \frac{\partial p}{\partial x}\right)$, where p denotes the probability and $\frac{\partial p}{\partial x}$ is the partial derivative of this probability concerning a specific parameter x. This setup is particularly useful in the context of a probabilistic fact with a learnable probability, denoted as $t(p_i) :: f_i$ where p_i is the probability of the fact. In a ground program with N parameters, these parameters are compiled into a parameter vector $\vec{x} = [x_1, \ldots, x_N]^\top$.

The operations $\oplus$ and $\otimes$ within the gradient semiring are defined based on the probability calculations of ProbLog and the gradient computations obtained through derivative rules. They are mathematically expressed as:

$$(x_1, \vec{x_2}) \oplus (y_1, \vec{y_2}) = (x_1 + y_1, \vec{x_2} + \vec{y_2})$$
$$(x_1, \vec{x_2}) \otimes (y_1, \vec{y_2}) = (x_1 y_1, y_1 \vec{x_2} + x_1 \vec{y_2})$$

In this framework, the neutral elements are given by $e^\oplus = (0, \vec{0})$ and $e^\otimes = (1, \vec{0})$.

In preparation for applying gradient descent for learning parameters, it is essential to first transform the ProbLog program into an aProbLog format. This involves expanding the label of each probabilistic fact, denoted as $p :: f$, to encompass both the probability p and its gradient vector relative to the probabilities of all probabilistic facts within the program. The transformation can be represented as follows:

$$
\begin{aligned}
L(f) &= (p, \vec{0}) && \text{for } p :: f \text{ with fixed } p \\
L(f_i) &= (p_i, \boldsymbol{e}_i) && \text{for } t(p_i) :: f_i \text{ with learnable } p_i \\
L(\neg f) &= (1 - p_i, -\nabla p) && \text{with } L(f) = (p, \nabla p) \\
L(f_i) &= (m_q(\vec{t})_i, \boldsymbol{e_i}) && \text{for } nn(m_q, \vec{t}, \vec{u}) :: f_j, \ldots, f_k \text{ a nAD}
\end{aligned}
$$

In this framework, the vector $\boldsymbol{e}_i$ is characterized by having a value of 1 at the i-th position and 0s elsewhere. When dealing with fixed probabilities, their gradient remains unaffected by any parameters and is therefore zero. In other scenarios, the semiring labels mentioned earlier are employed. To maintain well-defined ADs, where the probabilities of facts within the same AD sum to 1, these probabilities are re-normalized after every gradient descent update. This step is specifically necessary for non-neural ADs. In the context of neural predicates, a softmax layer inherently guarantees a normalized distribution. Neural predicates are distinctive in their operation: they serve as an interface, with each side—the logic and the neural network — treating the other as a black box. From the logic perspective, it is possible to compute the gradient of the loss relative to the neural network's output, even though the network's internal parameters remain concealed. This externally computed gradient is adequate to initiate backpropagation, which in turn computes the gradients for the neural network's internal parameters. During the gradient computation process within aProbLog, probabilities associated with nADs are held constant.

3.6 Reinforcement Learning

Markov Decision Processes

The *agent* learns and decides in the *environment*. To interact with the environment, the agent uses *actions* that trigger a response by the environment called *rewards*. A Markov Decision Process (MDP) is defined as a tuple $M = (\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \gamma)$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space. Given $a \in \mathcal{A}$ and $s, s' \in \mathcal{S}$. $\mathcal{T}$ is the transition matrix with $\mathcal{T}(s, a, s') = P(s'|s, a)$, $\mathcal{R}(s, a, s')$ is the immediate reward after taking action a in state s, leading to the next state s' and γ is the discount factor. To decide, which action to take,

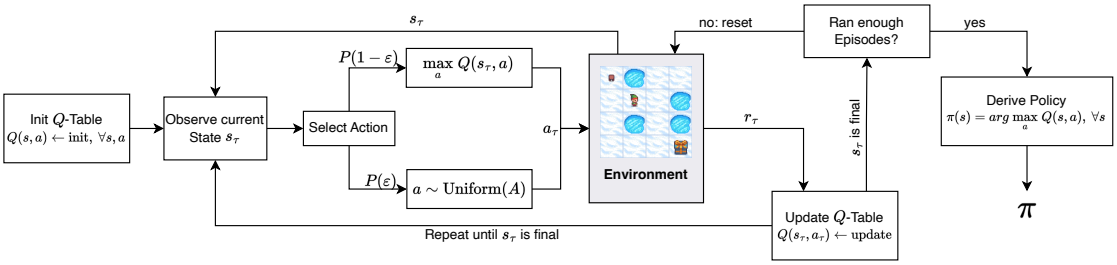


Figure 3.3: **Overview of the Q-Learning Process**. This flowchart outlines the steps of the Q-Learning algorithm, from initialization of the Q-table through the selection of actions and updating of Q-values to the derivation of the policy after sufficient episodes.

the agent uses a policy $\pi : \mathcal{S} \times \mathcal{A} \to [0, 1]$ where $\pi(a|s)$ denotes the probability of taking action a in state s. The discount factor weighs the importance of immediate rewards (γ close to 0) against future rewards (γ close to 1) to prioritize short-term gains or to plan for long-term benefits. [51]

Q-Learning

Q-Learning [51, 52] is a model-free off-policy reinforcement learning algorithm. The core of Q-Learning is to learn a function called the Q-function, which estimates the value of taking a given action in a given state. Being model-free allows the algorithm to learn to make optimal decisions without a model of its environment. As an off-policy algorithm, the behavior policy used for exploration in Q-Learning operates independently of the estimation policy being assessed and refined.

The Q-function that tells how good it is to take an action a in a state s is defined as:

$$Q(s, a) = R(s, a) + \gamma \mathbb{E}_{s' \sim P(\cdot|s,a)} \left[\max_{a' \in A} Q(s', a') \right].$$

For selecting an action a_t at every time step, Q-Learning uses its behavior policy, often a ϵ-greedy policy, to balance between exploration (selecting a random action) and exploitation (taking the best-known action). By doing so, the behavior policy ensures that the algorithm not only exploits the current best strategy for immediate rewards but also explores enough to discover potentially better strategies, enhancing its learning and performance over time.

Using an evaluation policy that only exploits what it already knows would not be able to learn a good estimation policy, as it would be trapped in a limited portion of the state space. To learn the Q-function, Q-Learning uses the following update rule with a learning rate α:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[R(s_t, a_t) + \gamma \max_{a' \in A} Q(s_{t+1}, a') - Q(s_t, a_t) \right].$$

To extract the policy π for each state from the learned Q-function, we simply take the argmax over all actions in a state to get the highest expected return, i.e.

$$\pi(s) = \arg\max_a Q(s, a) \quad \forall s$$

Figure 3.3 presents a high-level flowchart illustrating the Q-Learning process, while Algorithm 1 provides the corresponding pseudocode implementation, detailing the algorithm's step-by-step execution.

We will use Q-Learning as an oracle for generating a dataset.

Algorithm 1 Pseudocode for Q-Learning

$\alpha \leftarrow$ any value in $(0, 1]$
$\epsilon \leftarrow$ small value > 0
$Q(s, a) \leftarrow$ init $\forall s, a$
$Q(terminal, \cdot) \leftarrow 0$
loop for each episode:
    $s \leftarrow$ init
    repeat for each step of episode:
        $a \leftarrow$ selected action using policy derived by Q (e.g., ϵ-greedy)
        Take action a, observe r and s'
        $Q(s, a) \leftarrow Q(s, a) + \alpha \left[r + \gamma \max_{a'} Q(s', a') - Q(s, a) \right]$
        $s \leftarrow s'$
    until s is terminal
end loop

4 Methods

In this chapter, we introduce the methodologies employed to develop and refine soft sensors using reward prediction from a given state of a world (for example a grid world) and the agent's position in this world as a strategic tool.

Required Inputs

The method hinges on either directly utilizing a MDP or a set of trajectories derived from the MDP's operation as input data. Specifically, we consider an MDP defined by the tuple $M = (\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \gamma)$, representing the state space, action space, transition probabilities, reward function, and discount factor, respectively. Alternatively, a set of trajectories represented as $(s_{\tau_i}, s_{\tau-1_i}, a_{\tau-1_i}, s_{\tau-2_i}, a_{\tau-2_i}, \ldots, s_{0_i}, a_{0_i}; r_i)$, can be used, where each trajectory captures a sequence of states, actions and the associated rewards.

Furthermore, the model requires the 3-tuple $(\mathbf{T}, \mathbf{R}, \mathbf{S})$ as input, where **T** denotes the transition probabilities expressed in probabilistic logic, **R** is the reward function also articulated in probabilistic logic and **S** represents a set of soft sensors. **T** needs to be designed as a rule that takes all previous actions and outputs the most likely current position. **R** is a rule that takes a state and position and outputs the reward for this combination.

Assumptions

For a seamless integration within the framework, we need to make assumptions about the state transition logic, soft sensor design and reward function.

State Transitions: We assume that the state transitions within the environment are known a priori and can be described, or at least approximated, using a probabilistic logic

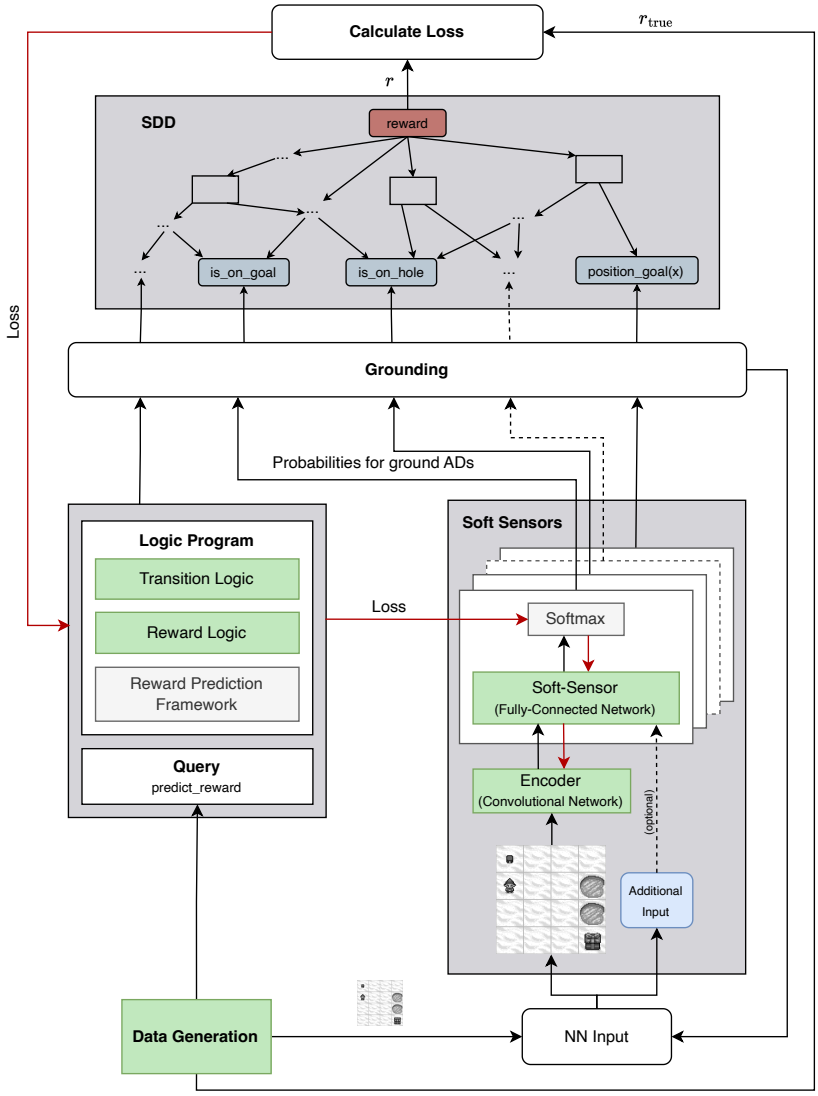


Figure 4.1: **Learning Process**. Depicted here is the soft sensor learning framework, marking user-defined elements in green. The cycle begins with data generation, producing reward prediction queries and logging action-state-reward sequences. Transition and reward logic define environmental rules. Training entails grounding this logic to the queries and deducing AD probabilities through soft sensors. An SDD is then crafted and evaluated for reward prediction. Loss calculated from this evaluation informs updates to the logic program and sensor parameters.

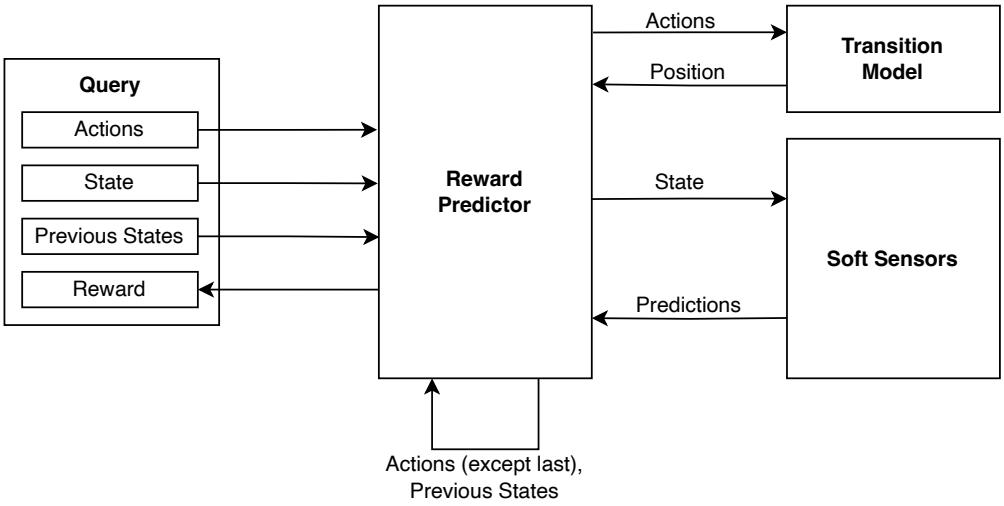


Figure 4.2: **Data Flow for Reward Prediction Query**. The figure displays the flow of data through various components in response to a specific query for reward prediction. It details how actions, state, and previous states are fed into the reward predictor, which in turn utilizes the outputs from the transition model and soft sensors based on the state information to calculate the expected reward.

framework. This assumption is fundamental for modeling the dynamics of the system and predicting future states based on past actions.

Soft Sensor Design: The methodology relies on soft sensors for classification, typically implemented as neural networks, to produce a normalized output distribution, such as softmax. Soft sensors for regression tasks cannot be applied. The normalized output format is crucial for the seamless integration of the sensors into the DeepProbLog framework.

Reward Function: It is assumed that the reward function is constrained such that rewards depend solely on the current state, as defined by the agent's position in the world. The rewards have to be discrete because the logic requires a rule for every distinct reward. Moreover, to fully leverage the information contained in entire trajectories, the reward rule is applied recursively, starting from the final state and moving backward through the trajectory. We assume that each interim step in a trajectory provides a reward of 0 until the final state is reached. This approach allows for a cumulative assessment of the trajectory's overall reward based on the sequence of actions taken.

Training Process

The process begins by combining the transition logic **T** and the reward logic **R** within the framework's logic structure, followed by the embedding of soft sensors into this logic. In cases where an MDP is provided, an agent is learned using a suitable reinforcement learning algorithm. This trained agent is then employed to generate trajectories that serve as data points for further analysis. If a set of trajectories is already available, the data generation phase is bypassed.

Subsequently, these trajectories are utilized to construct queries aiming to predict the most probable reward for given sequences of states and actions, formalized as $P(R = r|s_\tau, s_{\tau-1}, a_{\tau-1}, s_{\tau-2}, a_{\tau-2}, \ldots, s_0, a_0)$. For each query, the logic program is grounded, a process that includes supplying the necessary inputs to the soft sensors. These sensors then output probabilities for the ground ADs. An important strategy in our training is a recursive use of the reward predictor on preceding states within an episode, anticipating a reward of 0 for all but the final step. This method is intended to enrich the model's understanding of each state's role and significance throughout an episode, rather than isolating its learning to the agent's immediate context. Figure 4.2 shows the recursive use of the reward predictor.

With a grounded program in place, an SDD is constructed to facilitate the prediction of rewards. The outputs from this SDD are used to compute the loss, which is subsequently fed back into the logic program and, from there, to the sensors to adjust their weights. This feedback loop ensures that the system can iteratively refine its soft sensors.

Upon completing the training process, a set of calibrated soft sensors will be obtained, ready for deployment without being tied to reward prediction metrics.

5 Experiments

5.1 Problem Setting

Figure 5.1: **Position Definition**. In this work, we adopt a row-wise numbering convention for every cell within the grid world.

The primary objective of this work is to use DeepProbLog for learning soft sensors on images. For this purpose, we use the “Frozen Lake“ game from the gymnasium library, a library that provides a standard API for single-agent reinforcement learning environments [53]. In Frozen Lake, the agent acts in a grid world using the actions *left*, *down*, *right* and *up*. The game board is themed as a frozen lake with holes in random cells. If the agent falls into one of the holes, he is set back to the spawn position. The agent begins in the top-left corner with the objective of reaching the gift that is originally located on the bottom-right cell. In this work, the game board generation has been changed so that the goal can be in any cell that is not the start. This allows for a better generalization of the soft sensors due to a larger sample variety. The reward is binary, 1 for reaching the

goal and 0 else. We slightly adjust the reward function so that it gives -1 when the agent falls into a hole. This change is done so that the model can better differentiate between different game states to enhance the sensor learning process. Depending on the setting, the environment is slippery, which means that whenever the agent takes an action, there is only a 33.33% chance that he walks in the desired direction. Otherwise, he slips in a perpendicular direction. In our experiment, we use the slippery version and a version with deterministic state transitions. Throughout the experiments, we will talk about the agent's positions or the positions of holes and the goal. A mapping from grid world cells to positions is depicted in figure 5.1.

To effectively build soft sensors in this setting, our approach involves a multi-faceted input system for the model. It receives a visual representation of the current game state, provided as an image, along with all actions executed before reaching that state and the representations of the previous states. The model's training is focused on predicting the reward based on these inputs. Since the model incorporates neural predicates that are designed to interpret specific aspects of the game, such as determining the agent's position given a visual representation of the state, the neural networks representing these predicates are also trained.

5.2 Data Collection and Pre-Processing

The main goal of our research is not to train an agent to solve the Frozen Lake game, but rather to learn soft sensors from data that is generated in a reinforcement learning setting. For this reason, we can use a pre-trained agent to generate a comprehensive dataset, see figure 5.2 for more details. Here, our focus lies on gathering a broad range of data points, necessitating a strategy that ensures a good exploration of the game board. A purely stochastic agent would not be sufficient for our purpose as its random behavior makes it highly unlikely to reach cells near the goal, thereby limiting the diversity of the data. On the other hand, a standard Q-Learning agent following the optimal path to the goal would only provide a limited subset of potential data as it would repeat the same optimal moves.

To address these limitations, an exploration factor ϵ is introduced into the Q-Learning algorithm. This ϵ-greedy approach allows the agent to occasionally deviate from the optimal path by taking random actions. To make sure the agent reaches cells close to the goal, the agent runs on the same board multiple times while ϵ is decreased over time. To maintain good data quality duplicates are removed.

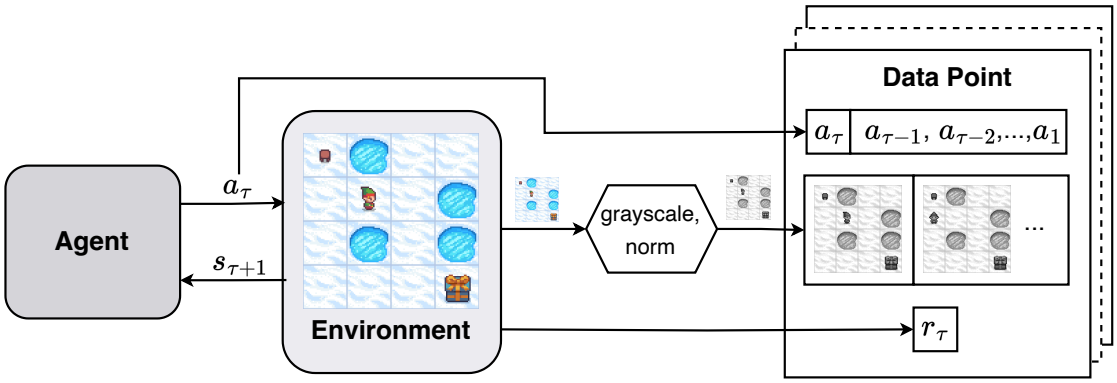
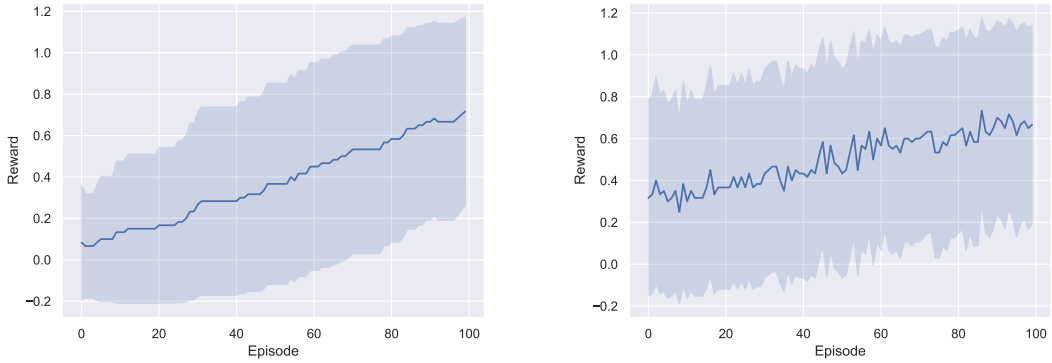


Figure 5.2: **Data Generation Pipeline**. A pre-trained agent is used to interact with the Frozen Lake environment. At each step, the taken action, along with the observed state (as an image) and the information from previous steps in the same episode are assembled into a data point.

To further enrich the dataset, data is collected from multiple randomly generated game boards. While the starting position remains constant, the number and location of the holes vary across these boards. To also randomize the goal position, we modify the original game board generation of gymnasium. For each unique board, a Q-Learning agent is trained, and data is generated using the ϵ-greedy method. Each data point consists of all states (as images) the agent has seen up to the final point, all actions taken, as well as the reward he got in the last step. To facilitate evaluation after training, the position of the goal and the positions of the holes are also collected.

The game board in our scenario is characterized by a single goal state and a limited number of holes, leading to an imbalanced dataset: numerous data points are associated with a reward of 0, and a smaller number reflects a reward of -1 (indicating the agent fell into a hole), and an even smaller fraction represents a reward of 1 (signifying the goal was reached). To address this imbalance, a combination of under- and oversampling techniques is employed [54]. Undersampling involves selecting a smaller representative subset from classes with an abundance of samples. In contrast, oversampling entails replicating samples from underrepresented classes. This dual approach ensures that we avoid using too few samples from large classes, while also preventing an excessive number of duplicates from small classes.

In preparation for neural network processing, each image img is converted to a grayscale



(a) Deterministic Environment

(b) Stochastic Environment

Figure 5.3: **Rewards for Q-Learning**. The rewards for the Q-Learning Agent for all random game boards used for training are averaged and plotted along with the standard deviation. 5.3a shows the rewards for a fully deterministic environment, while 5.3b shows the rewards for a stochastic environment.

image using the luminosity method:

$$img_{\text{grayscale}} = 0.299 \cdot img_{\text{red}} + 0.587 \cdot img_{\text{green}} + 0.114 \cdot img_{\text{blue}}$$

where img_{red}, img_{green}, and img_{blue} are the red, green and blue components of the image respectively. The grayscale image is then normalized to have values between 0 and 1 by dividing every pixel value by 255, following standard practice in machine learning. The actions, which are exclusively used within the logic program, do not require any modification.

Using this approach, two datasets are generated: One using deterministic state transitions and one using stochastic state transitions in the game. Figure 5.3 shows the rewards during the training phase of the Q-agents used for data generation. A set of short trajectories of the deterministic training data is displayed in figure 5.4 and a set of long trajectories of the stochastic training data is depicted in figure 5.5. Both the deterministic and the stochastic data set contain short and long trajectories.

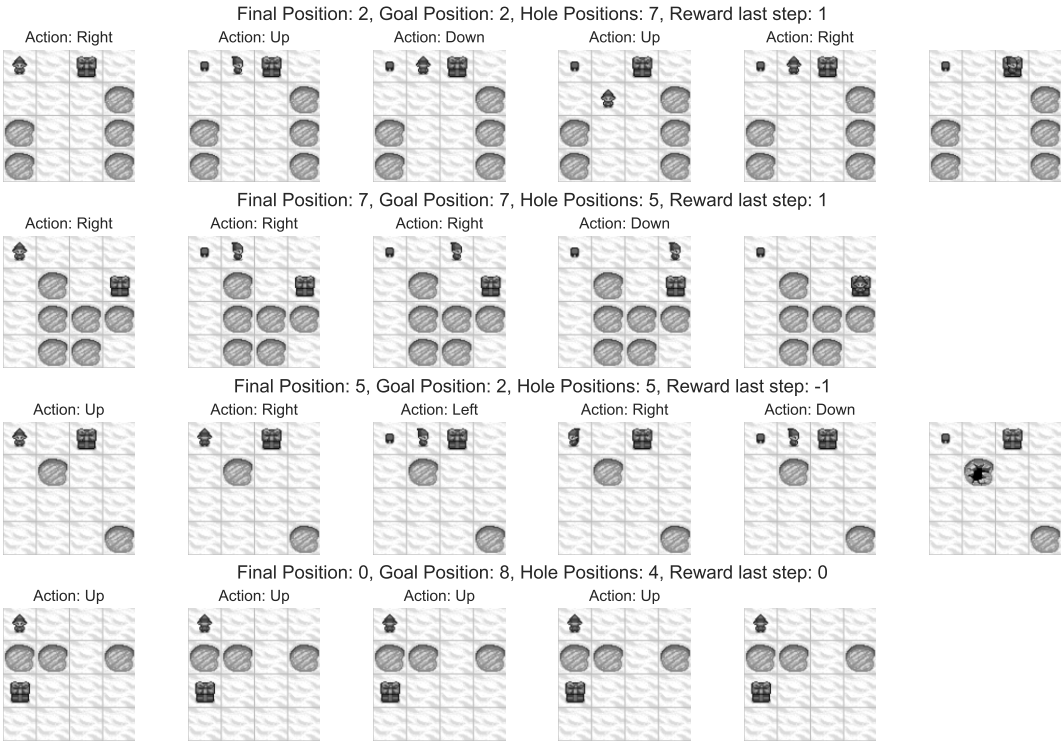


Figure 5.4: **Training Data: Short Trajectories**. A randomly selected subset of the deterministic training data showing shorter trajectories. Each row is one data point, where the images are the state and the actions are the ones taken in the corresponding state. The last image has no action as it is the final state (at least for the corresponding trace). The specified reward is the one that should be predicted, the final position, goal and hole positions are given for the evaluation of the independent soft sensors at the end.

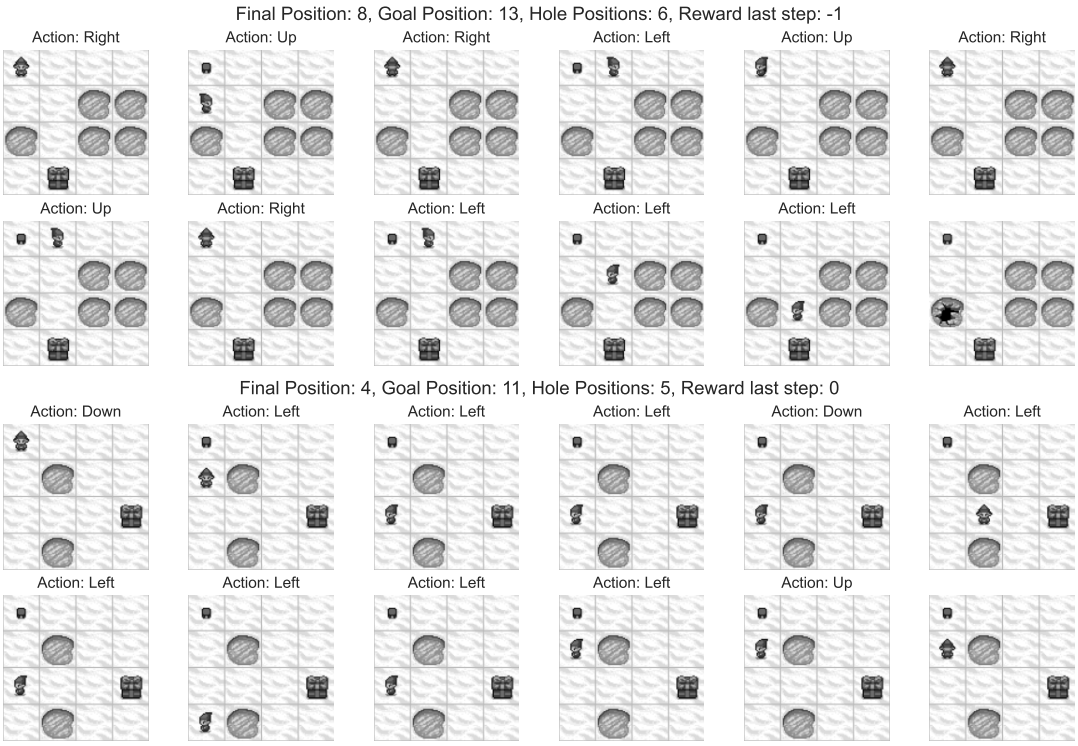


Figure 5.5: **Training Data: Long Trajectories**. A randomly selected subset of the stochastic training data showing longer trajectories. For each trajectory, the states are shown as images, and the corresponding actions are printed. The last image has no action as it is the final state (at least for the corresponding trace). The specified reward is the one that should be predicted, final position, goal and holes are given for the evaluation of the independent neural networks at the end.

5.3 Model

Our model integrates three principal components: (1) neural predicates as a connector between the probabilistic logic model and the neural networks (2) a deterministic transition model modeling the state transitions and (3) a reward predictor.

The transition model is used for position estimation, predicting the agent's current position given a start position and all the actions the agent has taken so far.

Neural Predicates

Our model employs six neural predicates, categorized into two distinct types, based on their input requirements and predictive capabilities. The first type includes predicates that analyze the image of the current state to make predictions about the scene as a whole. Within this group we have predicates designed to (1) identify the agent's position within the grid, (2) determine whether the agent is standing on the goal position (*is_on_goal*), (3) check whether the agent fell in a hole (*is_on_hole*) and one for determining the goal position (*goal_position*). For instance, in a 2×2 game board, the neural predicate m_{pos} predicting the agent being in cell 1 can be formalized as

$$nn(m_{\text{pos}}, \boxed{\text{🧍}}, [0, 1, 2, 3]) :: \text{pos}(\boxed{\text{🧍}}, 0); \text{pos}(\boxed{\text{🧍}}, 1); \text{pos}(\boxed{\text{🧍}}, 2); \text{pos}(\boxed{\text{🧍}}, 3).$$

Predicates in the second category require both the image of the current state and a specific position as inputs, allowing them to make predictions about individual cells in the grid world. This type of predicate is used to (1) check for the presence of a hole in a specified cell (*position_hole*) and (2) to evaluate whether a particular cell represents the goal (*position_goal*). Taking the 2×2 grid world example again, the neural predicate $m_{\text{pos\_hole}}$ predicting whether position 2 is a hole can be formalized as

$$nn(m_{\text{pos\_hole}}, \left[\boxed{\text{🧍}}, 2\right], [0, 1]) :: \text{pos\_hole}(\left[\boxed{\text{🧍}}, 2\right], 0); \text{pos\_hole}(\left[\boxed{\text{🧍}}, 2\right], 1).$$

Soft Sensors

Integral to each neural predicate within our model is a dedicated soft sensor for classification, in this work a neural network. The architecture employed in this work adheres to a classical encoder-decoder paradigm.

At the forefront of our neural network design is a convolutional encoder that consists of two convolutional layers, designed to extract the essential features from the input images. We use a single encoder that is shared by all sensors, allowing it to learn meaningful representations across different tasks.

Following the feature extraction, task-specific decoders for classification, constructed as Dense Networks, interpret the encoded features and produce the targeted predictions. Using a softmax layer, all decoders output a probability distribution over all possible classes (i.e., over the positions of the game board or a binary yes/no decision). Matching the two groups of neural predicates, there are two different types of decoders.

The decoders of the first type operate solely on the feature representations generated by the encoder. The classification networks for the position prediction and the goal position prediction output a probability distribution over all possible positions. The networks that predict whether the agent is on the goal or has fallen into a hole output a probability distribution over two categories: yes (the condition is met) and no (the condition is not met).

The second type of decoder incorporates a particular position into the prediction process. Originally, it was planned to combine the position with the feature representation of the encoder, but this implementation was incapable of learning the expected functions. To solve this issue, a different approach is applied, leveraging the grid characteristics of the environment. Instead of passing the image as a whole to the decoder, only the cell of interest is passed. To ensure that the same encoder can be used for all sensors, it is resized to the original image size after cutting out the cell. These networks always output a binary probability distribution. This distribution reflects the likelihood of the condition specified by the task-specific decoder, for example, if a position is a hole, being met or not for the given position.

The distinct structure of our neural network architecture is exemplified in Figure 5.6, showcasing the design of the agent position and goal position predictor.

Reward Predictor

The reward predictor constitutes a pivotal component of our model, assembling the conditions necessary for the agent to perceive specific rewards at distinct states. It operates by integrating the deterministic insights offered by the transition model with the predictions provided by the neural predicates. The reward determination is segmented into three distinct scenarios, dependent on the truth values of the neural predicates *is_on_goal*,

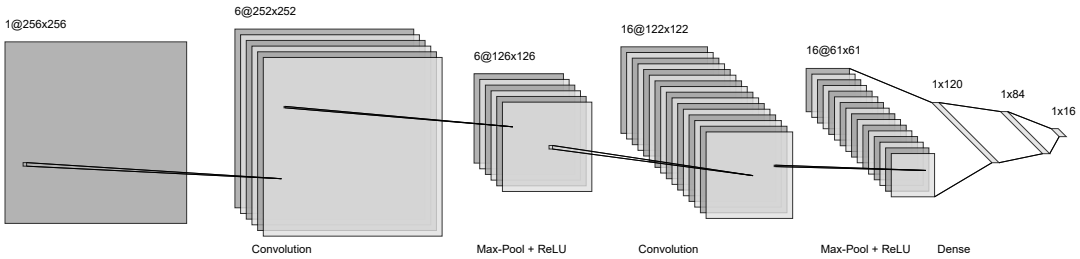


Figure 5.6: **Neural Network Architecture**. The architecture used in this thesis follows a classical encoder-decoder structure. We use a Convolutional Neural Network as a shared encoder and Dense Networks as task-specific classification decoders.

is_on_hole, *goal_position*(*Position*), *position_goal*(*Position*) and *position_hole*(*Position*). The *Position* is predicted by the transition model, using the trace of actions the agent has taken to reach the current state.

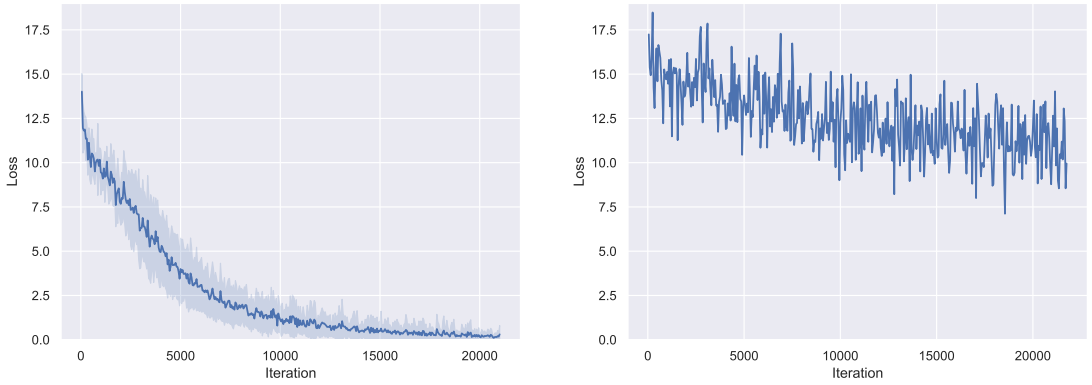
Reward $= 1$: The agent is granted a reward of 1 upon arriving at the goal. This is true if *is_on_goal* is true, *is_on_hole* is false, *goal_position*(*Position*) is true, *position_goal*(*Position*) is true and *position_hole*(*Position*) is false.

Reward $= -1$: The agent receives a penalty of -1 when he falls into a hole. In this case, the neural predicates have contrary outputs compared to the reward $= 1$ scenario, i.e. *is_on_goal* is false, *is_on_hole* is true, *goal_position*(*Position*) is false, *position_goal*(*Position*) is false and *position_hole*(*Position*) is true.

Reward $= 0$: The agent receives a reward of 0 if he neither stands on the goal nor fell into a hole. In this case, none of the neural predicates are true.

5.4 Training Process

Two separate training sessions on the model are conducted, each lasting 15 epochs. The first session utilizes the deterministic training data, while the second session uses the stochastic training data, presenting a more challenging scenario to test if a model using deterministic state transitions can make good predictions under uncertainty too. The



(a) Deterministic Training Data

(b) Stochastic Training Data

Figure 5.7: **Reward Prediction Loss**. Side-by-side evaluation of the loss decrease in reward prediction using deterministic vs. stochastic training data. In the deterministic case, multiple training runs are averaged and plotted along with the standard deviation and a smooth descent can be observed, while the stochastic case shows a lot of oscillation.

model is trained using the ADAM optimizer [55] with a learning rate of 10^{-5}. It has been found that within the DeepProbLog framework a very small learning rate is beneficial.

Additionally, to enhance the encoder's learning of representations, we incorporated auxiliary task learning, specifically for predicting the agent's position. This task's complexity varies with the training data, being straightforward with deterministic data due to predictable outcomes and the availability of concrete ground truth from the deterministic transition model. The encoder, designed to be task-independent and shared across all predictors, is refined using the position prediction task, illustrating how auxiliary tasks can serve as indirect mechanisms for boosting overall task performance.

Soft Sensor	Deterministic Training Data	Stochastic Training Data
Agent Position	0.9972	0.3165
Agent on Goal	1.0000	0.9944
Agent on Hole	0.9916	0.9860
Goal Position	0.7283	0.0420
Position is Goal	1.0000	0.9597
Position is Hole	0.9988	0.7626

Table 5.1: **Soft Sensor Accuracy**. Comparison of the accuracy of each soft sensor on a test set, depending on the used training data.

Soft Sensor	Deterministic Training Data	Stochastic Training Data
Position is Goal	1.0000	0.3557
Position is Hole	1.0000	0.0486

Table 5.2: **Soft Sensor Accuracy on positive Samples**. Accuracy is only calculated for cells where a goal/hole is.

5.5 Results

Evaluation

Unfortunately, the DeepProbLog framework does not support the tracking of individual neural predicates' losses but only for the complete model. This limitation necessitates a manual evaluation process for our soft sensors that calculates key metrics for each sensor on both the training and testing sets. Given our focus on the distinct performance of each sensor, this individualized evaluation is a critical component of our analysis.

Results for Deterministic Training Data

When trained on deterministic training data, the model demonstrated excellent performance in reward prediction. This high level of accuracy indicates that the model's soft sensors learned effectively from the deterministic data, with almost perfect predictions across the board, except for the sensor predicting the goal's position. A breakdown of the accuracy achieved by each sensor on the test set is shown in Table 5.1. Additionally,

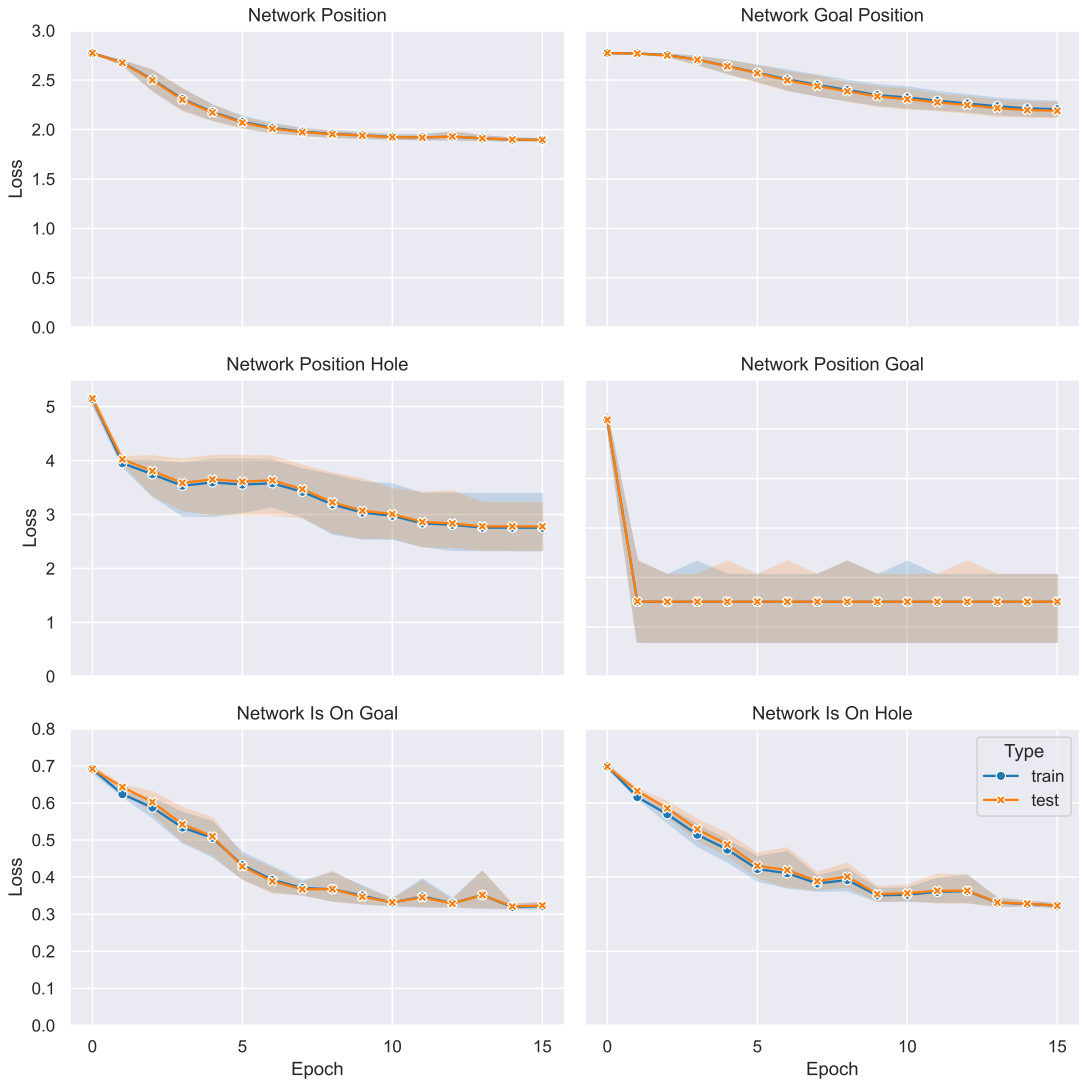


Figure 5.8: **Loss per Soft Sensor trained on deterministic data**. Each sub-figure shows the averaged loss for a specific soft sensor after each epoch for the training as well as the test data over multiple training runs, along with the standard deviation.

figure 5.7a depicts the training loss for the reward predictor, while figure 5.8 presents the training losses for each sensor throughout the training process. In figure 5.10a the confusion matrix for the position prediction is shown. It can be observed that the prediction is correct for most samples (the points lie on the diagonal) but the prediction for position 15 is not yielding good results.

Results for Stochastic Training Data

When the agent is trained on stochastic training data, a different pattern can be observed. Although the loss does not decrease as smoothly as with deterministic data, the model still achieves a respectable 83% accuracy in reward prediction. The loss curve is displayed in Figure 5.7b. This suggests a robustness to stochasticity, with certain soft sensors, such as *is_on_goal* and *is_on_hole* displaying accuracies potentially sufficient for accurate reward prediction. The sensor-specific accuracies are again shown in Table 5.1. The accuracies of networks determining whether a cell is a hole (*position_hole*) or the goal (*position_goal*) are also high, albeit somewhat misleading. This is because the accuracy is calculated across all cells, and when focusing solely on cells that actually contain a goal or hole, the accuracy for these predictions drops significantly, especially for *position_hole* predictions, see Table 5.2 for more details. This behavior indicates that these sensors tend to predict that there is no goal/hole in almost all cases. Analyzing the sensor-specific losses in figure 5.9 we observe that only the losses for *is_on_goal* and *is_on_hole* drop significantly. The loss for the other sensors is going down only very slowly, or even oscillates.

This discrepancy underscores the challenges posed by stochastic game transitions on the model's predictive capabilities. Particularly, the deterministic transition model used by our system struggles to accurately predict the agent's position in longer action traces, given the stochastic nature of the game data. This inaccuracy in position prediction directly impacts the learning of the position sensor and the networks responsible for predicting if specific cells are the goal or holes within the game environment. Figure 5.10b illustrates this through a confusion matrix for the position prediction after training, showing that while the model accurately predicts positions closer to the starting point, it becomes increasingly inaccurate for positions further away. This outcome highlights the limitations of relying on a deterministic transition model in a stochastic setting and suggests avenues for further research in improving model robustness and predictive accuracy.

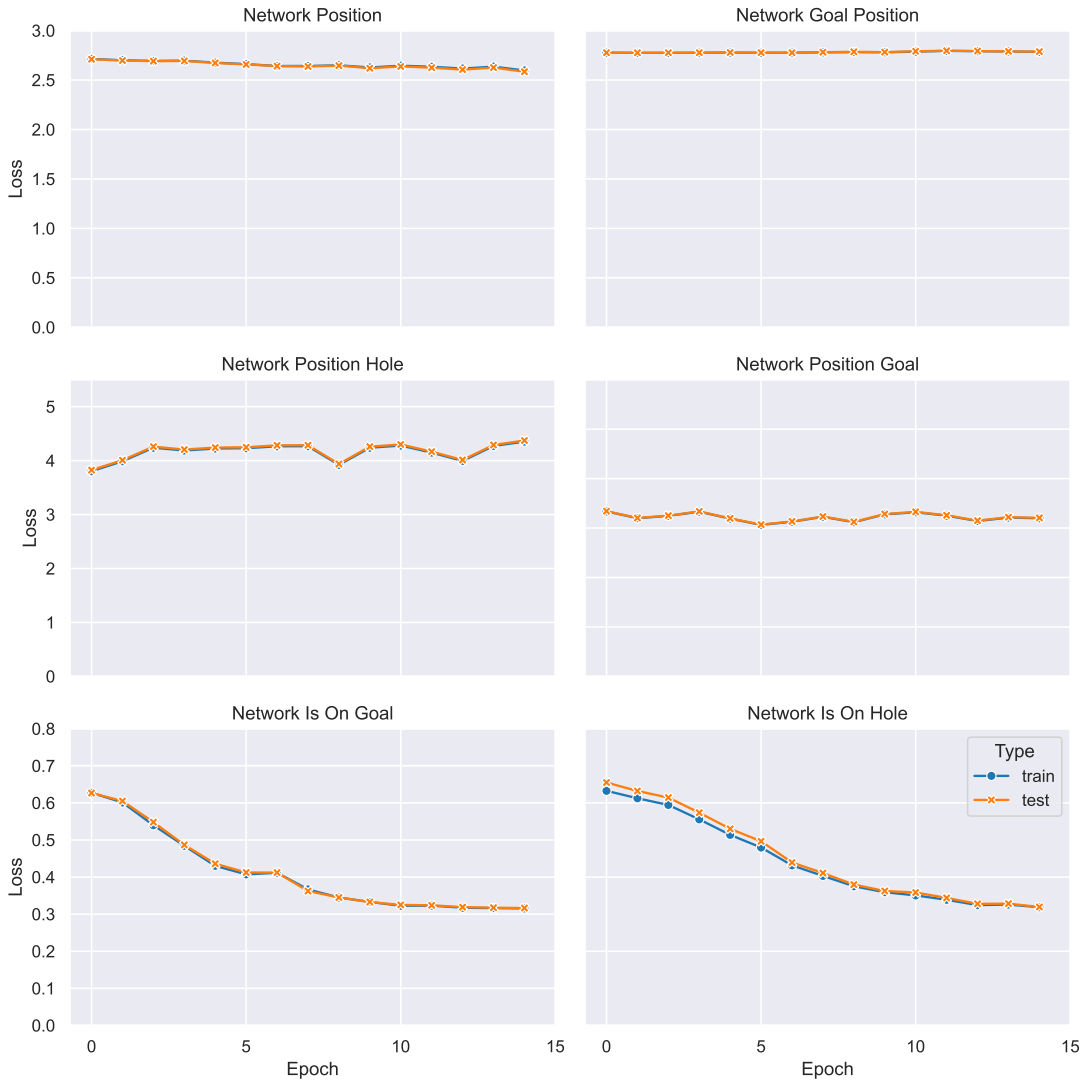
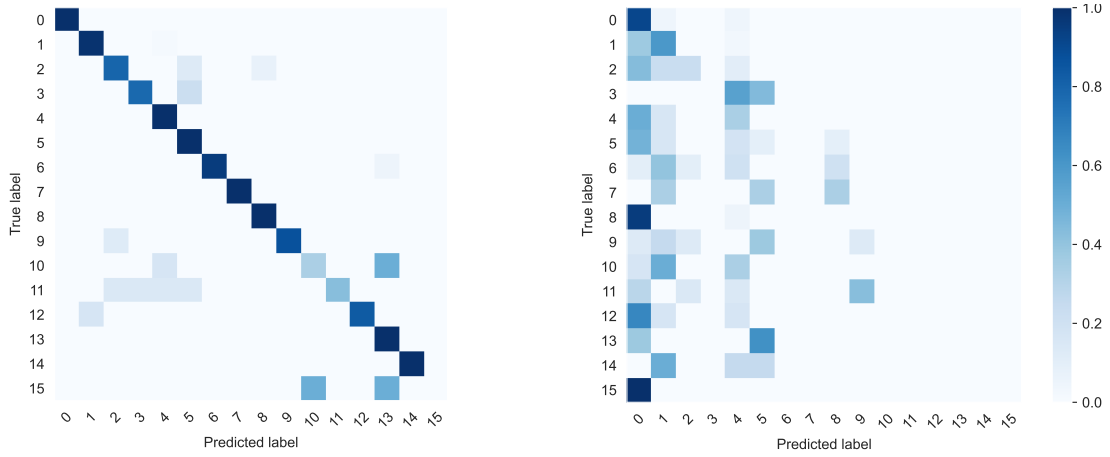


Figure 5.9: **Loss per Soft Sensor trained on stochastic data**. Each sub-figure shows the loss for a specific soft sensor after each epoch for the training as well as the test data. It can easily be noticed that only the sensors specifying whether the agent is standing on a special cell are learning.



(a) Deterministic Training Data

(b) Stochastic Training Data

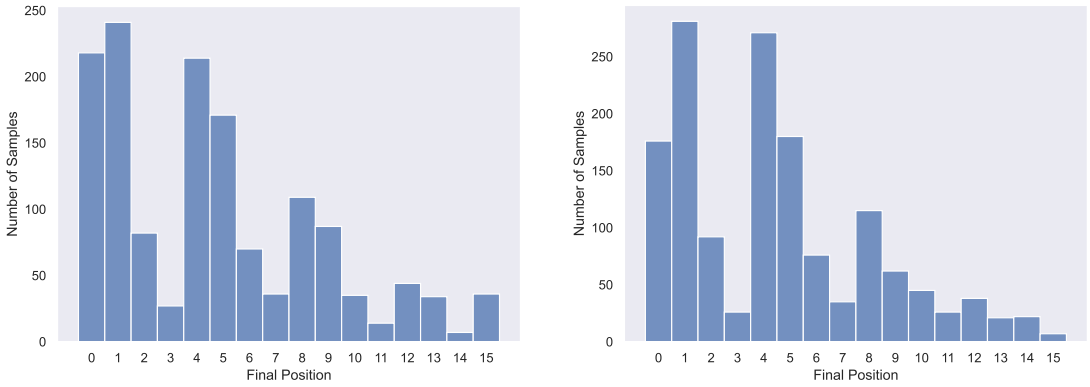
Figure 5.10: **Confusion Matrix for Position Prediction**. Each cell is normalized by the number of samples of the respective position class. In (a) we see that for deterministic training data, the position is predicted correctly in most cases. Most likely because of the imbalance of samples, position 15 is not predicted correct. In (b) it can be observed that the model shows higher accuracy for positions near the start but increasing discrepancies for distant positions. This is most likely due to the limitations of the underlying, deterministic transition model when working with stochastic data.

6 Discussion and Future Work

Discussion

We observe that our model has a very high accuracy for the reward prediction as well as for the individual soft sensors when trained on the dataset generated in a deterministic environment. This is an anticipated outcome given the deterministic nature of the underlying MDP. Consequently, our model's deterministic transition assumptions are harmonizing with the environment, ensuring that the positions used during training are consistently accurate. This alignment between model expectations and environmental behavior underscores the model's capability to learn and predict with high precision in controlled and predictable settings. In our analysis, the position prediction sensor yields good results, only for position 15 we can observe a discrepancy. This phenomenon can be principally lead back to the distribution of the agent's final position within the dataset. The implemented balancing measures are tailored exclusively towards the equilibration of rewards, neglecting the balance of properties of the game board, such as the position of the agent. As depicted in figure 6.1a, the distribution frequencies indicate that the data set contains samples for position 15. However, it is important to recognize that these instances represent only the terminal positions of each trajectory. The recursive training approach adopted in the training process leads to a sample for each step taken along the trajectory. Consequently, the number of samples of early positions is increased, whilst the proportion of samples for later positions, such as position 15, remains comparatively small.

When trained on a dataset generated in a stochastic environment, the model faces difficulties in tracing the agent's movements accurately due to the incompatibility of the stochastic environment and the deterministic transition model. Nevertheless, the model still manages to predict rewards with commendable accuracy. This success is largely due to the two sensors that can reliably determine if the agent has reached a goal or fallen into a hole. These sensors directly capture the immediate outcome — reward or penalty — associated with the last observed state. As a result of this correlation with the expected



(a) Deterministic Train Data

(b) Stochastic Test Data

Figure 6.1: **Final Position Distribution**. In (a) the distribution of the agent's final position of each trajectory in the deterministic training data is shown. Figure (b) shows the respective distribution for the stochastic training data.

reward, these sensors always receive strong and consistent training signals allowing for an easy training.

In our observations of the model's ability to predict positions, we've noticed it tends to perform better for locations near the starting point. This could be happening for a couple of reasons. First, the initial steps an agent takes are less affected by the randomness of the environment, so both the deterministic and stochastic models are more likely to agree on where the agent ends up. Second, our data isn't evenly spread out across all possible positions; it is balanced by the rewards the agent receives. Since the data is skewed this way, and because of the randomness of the game, we naturally end up with more information about the early game positions, see figure 6.1b for more details. This leads to the model being better informed about the start of the game than about the later parts.

Switching our model's current deterministic transition model to a stochastic transition model is theoretically possible, but the implementation in practice is challenging. The main issue is that the number of potential outcomes explodes dramatically. Every time the agent makes a move, it could not only proceed as planned but also slip away perpendicularly. This situation means that for every single action, there are three possible outcomes. Considering a sequence of actions, the complexity grows exponentially, for n actions there are 3^n potential paths to consider. This massive increase in the number of potential states makes it incredibly challenging to predict the outcome of even a relatively short sequence

of actions.

To manage the complexity introduced by considering a stochastic model, we need to rethink our current strategy of predicting an agent's position over its entire path, starting from the initial state. Currently, we calculate the position based on the entire trajectory from the start. However, with the transition to a stochastic model, a different approach is advisable. It is more practical to divide the trajectory into shorter segments that could consist of a limited number of actions, such as four, to enhance computational efficiency. However, this adjustment presents a new challenge: we can't simply start each segment's position calculation from the initial state because we need to know where each shorter trajectory begins. To solve this, we might need to employ a noisy sensor. One idea is to use the actual position data from the game environment (like those from a "Frozen Lake" scenario) as a sort of GPS to establish the starting point for each segment. Alternatively, we could develop a soft sensor trained on deterministic data to predict the starting positions of these shorter trajectories.

In our initial experiments, we used a different design for the soft sensors to determine if a particular spot is a goal or a hole, by combining the agent's position with the features derived from the encoder. Unfortunately, it turned out that this implementation is not able to accurately learn and perform the tasks set out for it. However, this misstep led us to an interesting discovery regarding the accuracy of our reward predictions. Even though these initial sensors were not performing well, our system was still able to predict rewards with surprising accuracy. The two sensors tended to oversimplify, assuming that any position is the goal position if the agent is standing on a goal and assuming that no position is the goal if the agent is not standing on the goal. Accordingly, the same logic applies to the prediction of whether a position is a hole or not. Basically, these two sensors learned to perform the same tasks as the *is_on_goal* and *is_on_hole* sensors. This revealed an important lesson: having accurate reward predictions doesn't necessarily mean all soft sensors of the system are working well as well. It is crucial to ensure that each sensor is doing its job correctly, without taking shortcuts or "cheating" the system.

Future Work

As discussed before, the complexity of stochastic environments calls for a more refined strategy in learning. By breaking down longer trajectories into shorter segments and employing a sophisticated method to pinpoint the starting position of each segment, we anticipate a significant improvement in the efficiency of the learning process.

Currently, soft sensors within our framework require upfront design and may not always be suited for their intended tasks. This limitation highlights the potential value of researching methods to automatically adjust the sensor architecture when it is found to be inadequate. Exploring ways to enable the framework to dynamically refine or reconfigure these sensors could significantly enhance their effectiveness and adaptability, ensuring they are always aligned with the specific demands of the task at hand.

Our framework has so far emphasized the agent's position as a fundamental aspect of the learning process. However, the diversity of environments extends far beyond those where location is the primary factor. Currently, the logic for transitions and rewards, along with the soft sensors, can be adjusted to accommodate needs beyond mere positioning. Nonetheless, it would be valuable to explore methods that adopt a broader perspective, incorporating a more extensive array of attributes.

One major component of the prediction process is the Sentential Decision Diagram, which utilizes logical representations, including the *noisy or* feature, to efficiently combine and evaluate various possible outcomes. Through the *noisy or*, there are multiple representations that can lead to the same outcome. It would be interesting to enhance control over the exploration process to better separate classes to refine the predictive capabilities of the framework

In our experiment, we have taken certain knowledge about the environment for granted, such as the specific dimensions of the grid world or the starting position of the agent. These assumptions can already be removed by adding adequate soft sensors and adjusting the probabilistic logic blocks accordingly. However, more elaboration on this topic could yield beneficial insights.

Our experiments have focused on learning across a spectrum of random, compact environments. Yet, the real-world scenarios often present a singular, expansive environment. Integrating a dynamically updated knowledge base into our learning process could be a game-changer, especially in complex, large-scale environments. This knowledge base would serve as a continuous learning loop, enhancing soft sensor learning with every iteration.

A considerable aspect of our existing model is built on the premise that both the transition model and the reward function are predefined and known, an assumption that does not hold in many real-world scenarios. While our current use of probabilistic logic allows for the learning of state transition probabilities, the adaptability of the reward prediction mechanism remains underdeveloped. Therefore, it becomes crucial to explore how the framework might be adapted to accommodate an unknown or partially known reward

function, ensuring it remains effective in environments where full information is not available.

7 Conclusion

In this thesis, we have introduced a complete end-to-end learning framework for training soft sensors. We base our learning process on the reward predictions of trajectories generated in MDPs. Therefore, the framework has customizable probabilistic logic components and any number of fungible soft sensors. The logic components are used for describing the state transitions and to determine how rewards are composed. Once set up, the framework can take a MDP or a set of trajectories generated within this MDP and predict the corresponding final reward for each trajectory. During the estimation process, it leverages the output of the transition model and the predictions of the soft sensors. The reward prediction loss is finally used to initialize backpropagation of the sensors. By solely focusing on the reward prediction, our approach overcomes the need for soft sensor-specific training data. We successfully applied our framework to the Frozen Lake game, training sensors to extract features from the game board. However, we also recognize limitations in using our framework in stochastic environments since the integration of a stochastic transition model within the logic components presents a non-trivial challenge.

Bibliography

[1] Yueyang Luo et al. “Data-driven soft sensors in blast furnace ironmaking: a survey”. In: *Front Inform Technol Electron Eng* 24.3 (Mar. 1, 2023), pp. 327–354. ISSN: 2095-9230. DOI: 10.1631/FITEE.2200366.

[2] Yasith S. Perera et al. “The role of artificial intelligence-driven soft sensors in advanced sustainable process industries: A critical review”. In: *Engineering Applications of Artificial Intelligence* 121 (May 1, 2023), p. 105988. ISSN: 0952-1976. DOI: 10.1016/j.engappai.2023.105988.

[3] Qingqiang Sun and Zhiqiang Ge. “A Survey on Deep Learning for Data-Driven Soft Sensors”. In: *IEEE Transactions on Industrial Informatics* 17.9 (Sept. 2021), pp. 5853–5866. ISSN: 1941-0050. DOI: 10.1109/TII.2021.3053128.

[4] Haibin Liu, Chao Wu, and Huanjie Wang. “Real time object detection using LiDAR and camera fusion for autonomous driving”. In: *Sci Rep* 13.1 (May 17, 2023), p. 8056. ISSN: 2045-2322. DOI: 10.1038/s41598-023-35170-z.

[5] Zheng Chai et al. “A Deep Probabilistic Transfer Learning Framework for Soft Sensor Modeling With Missing Data”. In: *IEEE Transactions on Neural Networks and Learning Systems* 33.12 (Dec. 2022), pp. 7598–7609. ISSN: 2162-2388. DOI: 10.1109/TNNLS.2021.3085869.

[6] Yuchen Zhao et al. “An End-to-End Multisource Information Fusion Framework for f-CaO Content Soft Sensing in Cement Clinker Burning Process”. In: *IEEE Transactions on Instrumentation and Measurement* 72 (2023), pp. 1–13. ISSN: 1557-9662. DOI: 10.1109/TIM.2023.3304693.

[7] Robin Manhaeve et al. “DeepProbLog: Neural Probabilistic Logic Programming”. In: *Advances in Neural Information Processing Systems*. Ed. by S. Bengio et al. Vol. 31. 2018. URL: https://proceedings.neurips.cc/paper_files/paper/2018/file/dc5d637ed5e62c36ecb73b654b05ba2a-Paper.pdf.

[8] Petr Kadlec, Bogdan Gabrys, and Sibylle Strandt. “Data-driven Soft Sensors in the process industry”. In: *Computers & Chemical Engineering* 33.4 (Apr. 21, 2009), pp. 795–814. ISSN: 0098-1354. DOI: 10.1016/j.compchemeng.2008.12.012.

[9] Vincent Brunner et al. “Challenges in the Development of Soft Sensors for Bioprocesses: A Critical Review”. In: *Frontiers in Bioengineering and Biotechnology* 9 (2021). ISSN: 2296-4185. URL: https://www.frontiersin.org/articles/10.3389/fbioe.2021.722202.

[10] Shuting Liu et al. “Soft sensor modelling of acrolein conversion based on hidden Markov model of principle component analysis and fireworks algorithm”. In: *The Canadian Journal of Chemical Engineering* 97.12 (2019), pp. 3052–3062. ISSN: 1939-019X. DOI: 10.1002/cjce.23520.

[11] Smail Dilmi. “Calcium Soft Sensor Based on the Combination of Support Vector Regression and 1-D Digital Filter for Water Quality Monitoring”. In: *Arab J Sci Eng* 48.5 (May 1, 2023), pp. 6111–6136. ISSN: 2191-4281. DOI: 10.1007/s13369-022-07263-w.

[12] Mingwei Jia et al. “Graph convolutional network soft sensor for process quality prediction”. In: *Journal of Process Control* 123 (Mar. 1, 2023), pp. 12–25. ISSN: 0959-1524. DOI: 10.1016/j.jprocont.2023.01.010.

[13] Ramakanth Pasunuru et al. “Dual Reinforcement-Based Specification Generation for Image De-Rendering”. In: (Mar. 2021). DOI: 10.48550/arXiv.2103.01867.

[14] Wataru Shimoda et al. “De-rendering Stylized Texts”. In: (Oct. 2021). DOI: 10.48550/arXiv.2110.01890.

[15] Felix Wimbauer, Shangzhe Wu, and Christian Rupprecht. “De-rendering 3D Objects in the Wild”. In: (Sept. 2022). DOI: 10.48550/arXiv.2201.02279.

[16] Dario Pavllo et al. “Shape, Pose, and Appearance from a Single Image via Bootstrapped Radiance Field Inversion”. In: (Mar. 2023). DOI: 10.48550/arXiv.2211.11674.

[17] Tong Wu et al. “HyperDreamer: Hyper-Realistic 3D Content Generation and Editing from a Single Image”. In: (Dec. 2023). DOI: 10.48550/arXiv.2312.04543.

[18] Yuto Enyo and Ko Nishino. *Diffusion Reflectance Map: Single-Image Stochastic Inverse Rendering of Illumination and Reflectance*. Dec. 2023. DOI: 10.48550/arXiv.2312.04529.

[19] François Plesse et al. “Learning Prototypes for Visual Relationship Detection”. In: *2018 International Conference on Content-Based Multimedia Indexing (CBMI)*. 2018 International Conference on Content-Based Multimedia Indexing (CBMI). Sept. 2018, pp. 1–6. DOI: 10.1109/CBMI.2018.8516557.

[20] Yaohui Zhu and Shuqiang Jiang. “Deep Structured Learning for Visual Relationship Detection”. In: *Proceedings of the AAAI Conference on Artificial Intelligence* 32.1 (Apr. 27, 2018). ISSN: 2374-3468. DOI: 10.1609/aaai.v32i1.12271.

[21] Long Zhao et al. *Unified Visual Relationship Detection with Vision and Language Models*. Aug. 20, 2023. DOI: 10.48550/arXiv.2303.08998.

[22] Zacharias Anastasakis et al. “Self-Supervised Learning for Visual Relationship Detection through Masked Bounding Box Reconstruction”. In: (Nov. 8, 2023). DOI: 10.48550/arXiv.2311.04834.

[23] Xiaodan Liang, Lisa Lee, and Eric P. Xing. “Deep Variation-Structured Reinforcement Learning for Visual Relationship and Attribute Detection”. In: 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). IEEE Computer Society, July 1, 2017, pp. 4408–4417. ISBN: 978-1-5386-0457-1. DOI: 10.1109/CVPR.2017.469.

[24] Nils Jansen et al. “Safe Reinforcement Learning Using Probabilistic Shields (Invited Paper)”. In: *DROPS-IDN/v2/document/10.4230/LIPIcs.CONCUR.2020.3*. 31st International Conference on Concurrency Theory (CONCUR 2020). Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. DOI: 10.4230/LIPIcs.CONCUR.2020.3.

[25] Nathan Hunt et al. “Verifiably Safe Exploration for End-to-End Reinforcement Learning”. In: (July 2, 2020). DOI: 10.48550/arXiv.2007.01223.

[26] Wen-Chi Yang et al. *Safe Reinforcement Learning via Probabilistic Logic Shields*. Mar. 6, 2023. DOI: 10.48550/arXiv.2303.03226.

[27] Prithviraj Ammanabrolu and Matthew Hausknecht. *Graph Constrained Reinforcement Learning for Natural Language Action Spaces*. Jan. 23, 2020. DOI: 10.48550/arXiv.2001.08837.

[28] Soroush Korivand, Nader Jalili, and Jiaqi Gong. “Inertia-Constrained Reinforcement Learning to Enhance Human Motor Control Modeling”. In: *Sensors* 23.5 (Jan. 2023), p. 2698. ISSN: 1424-8220. DOI: 10.3390/s23052698.

[29] Rich Caruana. “Multitask Learning”. In: *Machine Learning* 28.1 (1997), pp. 41–75. DOI: 10.1023/A:1007379606734.

[30] B. Bakker and T. Heskes. “Task Clustering and Gating for Bayesian Multitask Learning”. In: *Journal of Machine Learning Research* 4 (Jan. 2003), pp. 83–99. DOI: 10.1162/153244304322765658.

[31] Shijie Chen, Yu Zhang, and Qiang Yang. “Multi-Task Learning in Natural Language Processing: An Overview”. In: (Sept. 2021). DOI: 10.48550/arXiv.2109.09138.

[32] Zhihan Zhang et al. “A Survey of Multi-task Learning in Natural Language Processing: Regarding Task Relatedness and Training Methods”. In: (Feb. 2023). DOI: 10.48550/arXiv.2204.03508.

[33] Carl Doersch and Andrew Zisserman. “Multi-task Self-Supervised Visual Learning”. In: (Aug. 2017). DOI: 10.48550/arXiv.1708.07860.

[34] Apoorv Khattar, Srinidhi Hegde, and Ramya Hebbalaguppe. “Cross-Domain Multitask Learning for Object Detection and Saliency Estimation”. In: *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. June 2021, pp. 3634–3643. DOI: 10.1109/CVPRW53098.2021.00403.

[35] Duy-Kien Nguyen and Takayuki Okatani. *Multi-task Learning of Hierarchical Vision-Language Representation*. Dec. 2018. DOI: 10.48550/arXiv.1812.00500.

[36] Jiasen Lu et al. “12-in-1: Multi-Task Vision and Language Representation Learning”. In: (June 2020), pp. 10434–10443. DOI: 10.1109/CVPR42600.2020.01045.

[37] Thanh-Hoang Nguyen-Vo et al. “iCYP-MFE: Identifying Human Cytochrome P450 Inhibitors Using Multitask Learning and Molecular Fingerprint-Embedded Encoding”. In: *J. Chem. Inf. Model.* 62.21 (Nov. 2022), pp. 5059–5068. DOI: 10.1021/acs.jcim.1c00628.

[38] Xin Tang et al. “Explainable multi-task learning for multi-modality biological data analysis”. In: *Nat Commun* 14.1 (May 2023), p. 2546. DOI: 10.1038/s41467-023-37477-x.

[39] Mariana-Iuliana Georgescu et al. “Anomaly Detection in Video via Self-Supervised and Multi-Task Learning”. In: (Sept. 2021). DOI: 10.48550/arXiv.2011.07491.

[40] Wenmeng Yu et al. “Learning Modality-Specific Representations with Self-Supervised Multi-Task Learning for Multimodal Sentiment Analysis”. In: *Proceedings of the AAAI Conference on Artificial Intelligence* 35.12 (May 2021), pp. 10790–10797. DOI: 10.1609/aaai.v35i12.17289.

[41] Ao Wang et al. “Traffic Prediction With Missing Data: A Multi-Task Learning Approach”. In: *IEEE Transactions on Intelligent Transportation Systems* 24.4 (Apr. 2023), pp. 4189–4202. DOI: 10.1109/TITS.2022.3233890.

[42] Shikun Liu, Edward Johns, and Andrew J. Davison. *End-to-End Multi-Task Learning with Attention*. arXiv:1803.10704 [cs]. Apr. 2019. DOI: 10.48550/arXiv.1803.10704.

[43] Lukas Liebel and Marco Körner. *Auxiliary Tasks in Multi-task Learning*. May 2018. DOI: 10.48550/arXiv.1805.06334.

[44] Michael R A Huth and Mark D Ryan. *Logic in Computer Science: Modelling and Reasoning about Systems*. Cambridge University Press, 2002. ISBN: 052165602.

[45] De Raedt and Kimmig. *Probabilistic (logic) programming concepts*. 2015. URL: DOI:10.1007/s10994-015-5494-z.

[46] Luc De Raedt, Angelika Kimmig, and Hannu Toivonen. “ProbLog: a probabilistic prolog and its application in link discovery”. In: *Proceedings of the 20th international joint conference on Artifical intelligence*. Jan. 2007, pp. 2468–2473.

[47] Daan Fierens et al. “Inference and learning in probabilistic logic programs using weighted Boolean formulas”. en. In: *Theory and Practice of Logic Programming* 15.3 (May 2015), pp. 358–401. DOI: 10.1017/S1471068414000076.

[48] Adnan Darwiche. “SDD: A New Canonical Representation of Propositional Knowledge Bases.” In: IJCAI International Joint Conference on Artificial Intelligence. Jan. 1, 2011, pp. 819–826. DOI: 10.5591/978-1-57735-516-8/IJCAI11-143.

[49] Angelika Kimmig, Guy Van den Broeck, and Luc De Raedt. “An Algebraic Prolog for Reasoning about Possible Worlds”. In: *Proceedings of the AAAI Conference on Artificial Intelligence* 25 (Aug. 2011), pp. 209–214. ISSN: 2374-3468. DOI: 10.1609/aaai.v25i1.7852.

[50] Luc De Raedt et al. “Learning Probabilistic and Logical Models”. In: *Statistical Relational Artificial Intelligence: Logic, Probability, and Computation*. Ed. by Luc De Raedt et al. Synthesis Lectures on Artificial Intelligence and Machine Learning. Springer International Publishing, 2016, pp. 93–103. ISBN: 978-3-031-01574-8. DOI: 10.1007/978-3-031-01574-8_7.

[51] Richard S Sutton and Andrew G Barto. *Reinforcement Learning: An Introduction*. 2nd ed. Bradford Books, 2018. ISBN: 9780262039246.

[52] Christopher J. C. H. Watkins and Peter Dayan. “Q-learning”. In: *Machine Learning* 8.3 (May 1992), pp. 279–292. ISSN: 1573-0565. DOI: 10.1007/BF00992698.

[53] Mark Towers et al. *Gymnasium*. Mar. 2023. DOI: 10.5281/zenodo.8127026.

[54] Małgorzata Bach, Aleksandra Werner, and Mateusz Palt. "The Proposal of Undersampling Method for Learning from Imbalanced Datasets". In: *Procedia Computer Science*. Knowledge-Based and Intelligent Information & Engineering Systems: Proceedings of the 23rd International Conference KES2019 159 (Jan. 2019), pp. 125–134. ISSN: 1877-0509. DOI: 10.1016/j.procs.2019.09.167.

[55] Diederik P. Kingma and Jimmy Ba. "Adam: A Method for Stochastic Optimization". In: *CoRR* (Dec. 22, 2014). URL: https://www.semanticscholar.org/paper/Adam%3A-A-Method-for-Stochastic-Optimization-Kingma-Ba/a6cb366736791bcccc5c8639de5a8f9636bf87e8.