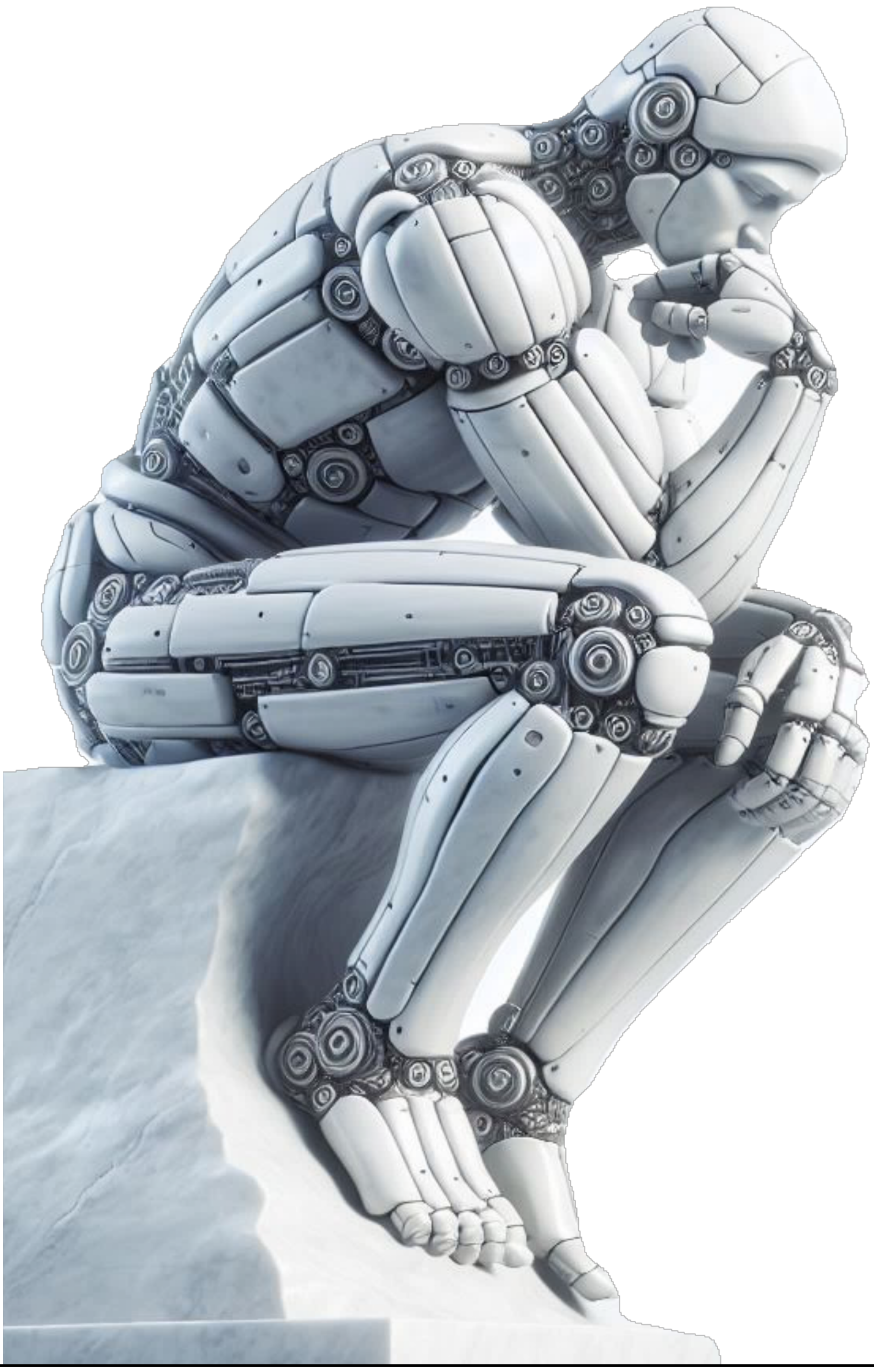
Knowledge Updating Strategies for Cumulative Learning of Markov Logic Networks

G.B.G. Potter

MSc thesis



Delft University of Technology

Cognitive Robotics Department

Knowledge Updating Strategies for Cumulative Learning of Markov Logic Networks

MSc thesis

April 16, 2024

G.B.G. Potter
Student number: 4573951

Supervisor TU Delft: Dr. ir. Joris Sijs
Supervisor TNO: Dr. ir. Gertjan Burghouts

Project duration: May 1 - April 16, 2024

Faculty of Mechanical Engineering (ME) · Delft University of Technology

The work in this MSc thesis report was supported by TNO and the Cognitive Robotics Department. Their cooperation is hereby gratefully acknowledged.

Abstract

Learning new concepts is a difficult task for autonomous robots. These robots can adapt to changes in the situations. To adapt to a situation, they should be able to determine the usefulness of objects around them. The usefulness of objects is highly dependent on situational context, making pre-programming of adaptation behaviour to all possible situations difficult. Automatic learning of this information by the robot from its own observations during deployment is a more feasible approach. We encode the usefulness of objects in logical statements that symbolise regularities in observations of objects in a Markov Logic Network. This network forms the basis from which the usefulness of newly observed objects or situations can be inferred by a robot. Each logical statement in the network has an associated weight that indicates how strong the robot believes the statement is true. These weights can be adjusted over time based on new observations made by the robot, strengthening its beliefs or creating a new belief for an unknown object in a process called cumulative learning. We extend the Markov Logic Network framework with a cumulative learning algorithm called MLN-CLA. This algorithm contains several different knowledge updating strategies to handle conflicts when incorporating new information. We demonstrate the ability of MLN-CLA to learn the usefulness of unknown objects over time. In addition, we demonstrate the abilities of MLN-CLA to incorporate new logical statements to better capture information about objects and situational contexts. Together, these two abilities enable robots to autonomously adapt to changing situations during deployment.

Contents

Glossary

List of Acronyms

BN	Bayesian Network
MLN	Markov Logic Network
ROS	Robot Operating System
FOL	First-order logic
MLN-CLA	Markov Logic Network Cumulative Learning Algorithm
MRF	Markov Random Field
ROC	Receiver Operating Characteristic
AUC	Area Under the Receiver Operating Characteristic (ROC) Curve
TPR	True Positive Rate
FPR	False Positive Rate
LLM	Large Language Model
ZSL	Zero Shot Learning
XAI	Explainable Artificial Intelligence
SRL	Stastical Relational Learning
KC	Knowledge Category
KL	Knowledge List
SVM	Support Vector Machines

List of Algorithms

Chapter 1

Introduction

When Karel Čapek first coined the term *robot* in his play *Rossum's Universal Robots*, human-like robots were a futuristic imagination. Čapek's play introduced robots as servants of humanity who ended up revolting and causing the extinction of humans. His robots were organic and had human intentions, something we can still only dream of today. Robotics research and development has come a long way since then, but we have not quite created robots capable of perfectly mimicking human behaviour as Čapek imagined. We do, however, have specialised robots capable of quickly and accurately carrying out a wide variety of tasks such as welding, cleaning and social care.

These robots are either pre-programmed or pre-trained for their tasks. Robots of the future that are capable of executing more than just a couple of tasks require more learning time. This work focuses on learning on the job for mobile robots in dynamic environments via affordance inference. Learning during operation, also called lifelong learning [1], enables robots to adapt to changes in their environments.

1-1 The robotic system and dynamic environments

To better understand how robots can be taught to display autonomous dynamic behaviour in changing environments, first the robotic system must be understood. A robotic systems consists of several modules. Figure 1-1 provides an overview of these modules and their respective functions. Most robots today are made with the purpose of fulfilling one or a select few tasks. These robots have great difficulty in adapting to changing environments and performing tasks in different domains. The models their perception and planning modules are based on do not cover all possible situations the robot could operate in. Unexpected objects may be observed as well as changes in the situation due to people, animals or other robots interacting with the environment. To overcome this limitation we have several options: we could constrain the operation environment of the robot such that it is capable of handling any scenario possible within the constrains, or we could capture all possible situations a robot will ever encounter in their perception and planning modules, or we could create adaptive modules with the ability to change plans based on contextual information obtained via the sensors of the robot.

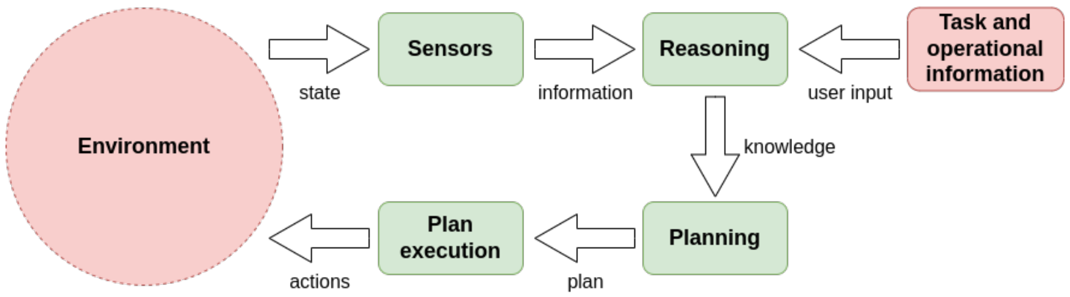


Figure 1-1: Overview of a robot system. A robot measures the state of the environment with its sensors. The sensor data is processed to extract information about the environment. This information in combination with task and operational information provided by the user, is used to reason about the current situation. The conclusions are used as input for the planning module. This module outputs a plan of actions to complete a task. The plan is executed with actions that change the state of the environment. The green colour represents internal systems and red external systems.

The first option is how robots have been built traditionally. A social robot for human companionship is not able to also work as a receptionist of a hotel without some adaptations. Figure 1-2 illustrates the perception-action cycle of a non-adaptive robot. A non-adaptive robot can only react to changes in the environment, but it does not have an inherent understanding of why it applies a certain action upon a certain change in the environment.

The second option is an impossible task, how can we know during the design of a robot all the possible situations it will encounter? Which edge cases do we have to account for? One can only account for a limited number of situations. Large machine learning models, such as Large Language Model (LLM), are trained on vast amounts of data to capture as many situations and edge cases within a model as possible. However, this approach is simply limited by the amount of data available. Such models show impressive cognitive reasoning abilities in various contexts. Clearly they capture nuanced patterns within different contexts and are able to leverage these patterns to display adaptive behaviour. Such a system is shown in figure 1-3, which illustrates the perception-action cycle of a robot with cognitive reasoning abilities.

The third option is the most feasible approach of the three. Self-learning and adapting models can generalise knowledge and incorporate new knowledge to solve problems in new situations based on their current knowledge. The idea is to endow a robot with some initial knowledge and give it the tools to make sense of the natural world during operation. This is the essence of the lifelong learning paradigm.

These tools are methods for capturing new knowledge from observations[1] of actions and their effects, storing new knowledge within the existing knowledge, combining existing knowledge concepts to form new concepts, updating existing knowledge and more. These methods are built in such a way that the cause and the meaning of certain observations can be converted to knowledge. In theory this could scale to infinity, i.e. as long as new data comes in, a robot could indefinitely learn new behaviours and context cues. Such a system in shown in figure 1-4, which illustrates a self-learning perception-action system.

[1]The terms observations, data and evidence are used interchangeably.

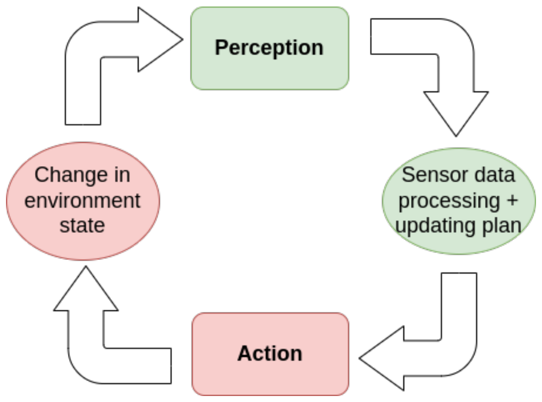

Perception
Sensor data processing + updating plan
Action
Change in environment state

Figure 1-2: Perception-action cycle overview. The green colour represents internal events and the red colour external events.

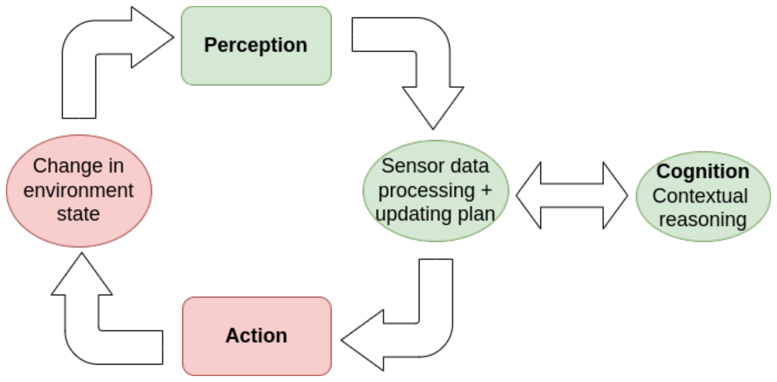

Perception
Sensor data processing + updating plan
Cognition
Contextual reasoning
Action
Change in environment state

Figure 1-3: The perception-action cycle extended with cognition. A cognition module adds reasoning capabilities for adaptive planning instead of purely reactive planning within limited environments. The green colour represents internal events and the red colour external events.

However, one of the problems with robots is that they do not *know* what the objects they interact with in the environment are. When we think of a door we immediately think of its function: to obstruct a passageway when closed and to allow passage when opened. The function of objects is different for each user. Depending on the capabilities of a user, the same object can have different functions. An apple can function as food for us, but for a robot with an arm or a 'ball-throwing device' it can function as a ball for throwing. Robots must understand the context of an object in an environment to be able to tell how an object can be used.

Notice how the function of an object is heavily dependent on context. Both the capabilities and the state of the environment influence the function of an object to an agent, that is a robot, human, animal etc. Furthermore, the configuration and physical properties of an object influence the function or possible actions associated with it in a given situation [2]. If a ball is stored inside a box, it cannot be thrown. It must first be taken out of the box before it can be thrown. Similarly, if the ball is made of lead, it is too heavy to throw. These examples show how knowing the mere name or label of an object is not enough to use for planning purposes in robots.

Creating a cognitive model capable of reasoning over object properties is a task well suited

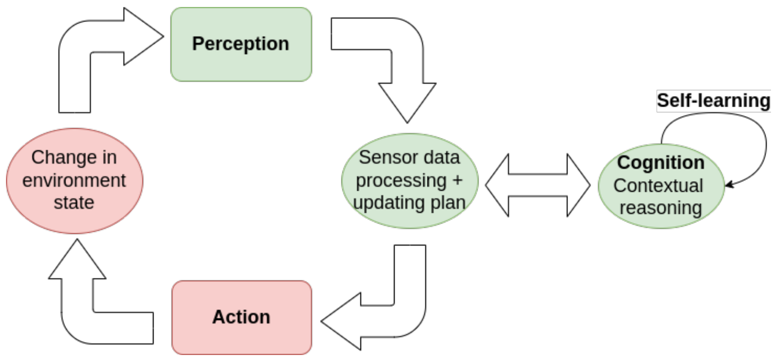

Perception
Self-learning
Change in environment state
Sensor data processing + updating plan
Cognition
Contextual reasoning
Action

Figure 1-4: The perception-action cycle extended with cognition and self-learning. A cognition module adds reasoning capabilities for adaptive planning instead of purely reactive planning. The self-learning cognition can adapt to potentially infinite situations. The green colour represents internal events and the red colour external events.

for machine learning. Recently, several works have explored the user interpretability or transparency of machine learning models in a research field called Explainable Artificial Intelligence (XAI) [3, 4, 5]. Robots that are deployed in dangerous environments or work with humans must be trustworthy to not cause any harm to humans. In this respect, a core tenet of XAI is to make machine learning models and robotic decision processes explainable to a user [6]. A user might ask "why did the robot just execute that specific action?". In machine learning some models are complex and the relation between input and output can become unclear. These types of models are called 'black box' models. Tracing the origin of an action back in black box models is a difficult task for anyone.

An example of a recent robotic reasoning system is SayCan [7]. In SayCan an LLM is used as a black box task planner. It converts natural language inputs into action commands it has learned from large amounts of data. How this conversion comes about for specific actions is obfuscated by the LLM model. Works like SayCan use post-hoc explainability techniques like saliency maps and feature attribution to peer into the black box model [5]. However, these explanation techniques provide insightful information for developers but are too difficult to interpret for users.

Making machine learning model decisions explainable starts at the representation level of data. More abstract representations of raw data are better interpetable by user as we can more intuitively map meaningful semantic symbols to them. Reasoning at a symbolic level allows for better explainability. In symbolic reasoning raw data is abstracted into symbols representing the data, such as `Heavy` representing a range of masses or `Green` representing a certain range of RGB colours. The semantic meaning of these symbols are easy to understand for an operator. Symbols can be chained and fit together in a syntax to form a language that machines can use to reason about and people can use to read. Some languages allow for logic to be applied to symbols. This is called symbolic reasoning with logic. Robots that have an understanding of symbols and logic can use the language and reasoning to build their knowledge bases.

Finding all possible correlations between symbols is an incredibly complex problem. Deep learning approaches hide this complexity in the neural network architecture and node weights. A more transparent and interpretable approach to finding, modelling and exploiting correlations between object properties, state and other contextual information can be found in the Stastical Relational Learning (SRL) research area. In this field of research symbolic relations are learned using weights to indicate the relation strengths or beliefs of a model in the correlation being true. In chapter 2 we compare the two main SRL approaches.

1-2 Encoding knowledge in symbols

The main advantage of using an abstract symbolic language to encode data is the ability to reason. In practice standard, machine learning models such as neural networks require datasets of millions of images to learn salient features related to the problem. Salient features of objects are for example their shape, size, mass etc. In symbolic reasoning models much smaller datasets can be used to encode salient features. These symbolic datasets can even consist of just a few megabytes of text. Compare that to the several hundred gigabytes of image data required for YOLO [8], SAM [9], LLMs [10, 11, 7] etc. The context range of these models is extremely large due to their large datasets. Symbolic reasoning models have a context range determined by their functionality, which is usually much less general than YOLO or SAM. However, the strength of symbolic reasoning models lies in their ability to be easily be extended to other context domains without much retraining.

Salient features can thus be encoded in symbolic knowledge models. A method for converting salient features into abstract symbols is to assign each feature to a variable such as 'a' or 'x'. Each variable is an abstract representations of one feature. A model can be created by establishing relations between these variables. Such a model can be used to answer questions about the variables and how they relate to each other. To reliably generate the same answer to the same questions, the conversion of salient features into abstract symbols must be standardised. This is done by applying a syntax, i.e. grammar, to the conversion. This syntax determines what types of symbols can exist and how symbols can be related to each other.

One such a syntax is called propositional logic. This logic consists of statements that are either true or false. These statements can be combined with logical connectives such as $\wedge$ (AND), $\vee$ (OR), $\neg$ (NEGATION), $\Rightarrow$ (IMPLIES) and $\Leftrightarrow$ (EQUALS) to create relations between propositions. An example of valid propositional logic is: $P \Rightarrow Q$. This statement is true unless P is true and Q is false. This conclusion can be reached via an inference mechanism called *modus ponens*). In this example, P and Q are symbols that can refer to natural language statements like "It is raining" (P) and "It is cloudy" (Q). Grounding these symbols in natural language is called an *interpretation* of P and Q. From these simple statements complex knowledge can be derived.

A limitation of propositional logic is its inability to deal with *quantifiers* to express sentences such as "There exists an object with an ear" or "All entities with ears are mugs". A more expressive logic is required to capture these sentences in logic. First-order logic (FOL) is such a logic. FOL adds the quantifiers $\forall$ (for all) and $\exists$ (exists) to propositional logic. FOL also adds *predicates*. Predicates are logical symbols representing properties or relations over their arguments. Predicates come in the form $P(A)$ or $P(A, B)$, meaning the predicate P applies to its argument A, a constant. Constants such as (A) can be replaced by variables (a) to represent that a relation applies to a set of constants, where the variable a is also called the *domain* of A[2]. Predicates can have one or more arguments, e.g. $P(A, b, C, d)$. The number of arguments of a predicate is called the *arity* of the predicate. A predicate can range over both variables and constants. A *ground predicate* or *ground atom* is a predicate of which all arguments that were variables are replaced by constants. Thus only constant arguments remain, for example $P(A, B, C, D), B \in b, D \in d$ is a ground atom.

With FOL a more diverse logic can be encoded in knowledge models. Dependencies between properties of objects or actions that can be applied to objects can be defined with predicates. The sentence "All heavy and large objects cannot be picked up" can be represented in FOL as $\forall$`obj IsHeavy(obj)` $\wedge$ `IsLarge(obj)` $\Rightarrow \neg$`Pick(obj)`. Note that the representation of this sentence in FOL is not unique. The same sentence can be translated differently: $\forall$`obj HasProperty(obj,Heavy)` $\wedge$ `HasProperty(obj,Large)` $\Rightarrow \neg$`Action(obj,Pick)` or $\forall$`obj HasMassAndSize(obj, Heavy, Large)` $\Rightarrow \neg$`PossibleAction(obj, Pick)`. Which representation to choose depends on the requirements of the problem to be captured in logic. In general, simpler formulas and predicates with lower arity are preferred as they are easier to understand and are faster to solve.

Thus it is clear that knowledge can be compactly represented in logic. Although not as intuitive and explainable as natural language, FOL represents a trade-off between machine interpretability and human interpretability. Modelling knowledge in logical formulas that capture certain regularities in the world has two major drawbacks: contradicting formulas and the Boolean interpretation of logic.

In lifelong learning, new formulas can be incorporated in the knowledge base of all known formulas. A new formula could contradict with existing formulas. Large knowledge bases consist of many formulas. These formulas often interact with each other via common dependencies. If a contradicting formula is accepted into the knowledge base it invalidates other formulas, resulting in a collapse of the knowledge base.

The second drawback is related to the fact that a formula in FOL can either be true or false, no other values are possible. If a formula in FOL is true, it means that the situation described in the formula always holds true for any situation. For example, take the formula `HasSize(Ball, Small)` $\Rightarrow$ `Throw(Ball)`. In natural language the formula expresses 'every small ball can be thrown'. This formula contains an implicit temporal component. Every small ball can *always* be thrown. In most situations this rule holds true. However, what if a ball is made of a very heavy material? Or what if the ball is stuck between other objects? Can it still be thrown? Depending on the situation it either can or cannot be thrown.

A symbolic knowledge model based on pure FOL formulas cannot handle these kind of situations. A potential solution to these drawbacks is to add a probability to a formula for it to be true in some situations and to be false in other situations. Such a model can then handle contradicting and soft formulas. A Markov Logic Network (MLN) is such a model that unifies probabilistic reasoning with FOL.

[2]Unless noted otherwise, all predicate arguments starting with an uppercase character are constants and those starting with a lowercase character are variables. This syntax follows the Alchemy convention described in appendix 8.

Weight	Formula	Meaning
1.3	`Robot(x)` $\Rightarrow$ `Think(x)`	If x is a robot, it can think
2.8	`Human(x)` $\Rightarrow$ `Think(x)`	If x is a human, it can think
-4.5	`Human(x)` $\Leftrightarrow$ `Robot(x)`	If x is a human, it is also a robot

Table 1-1: A Markov Logic Network consisting of three formulas. Each formula has a weight attached to it. The weight indicates how often a formula is true relative to the other formulas in the MLN.

1-2-1 Markov Logic Networks

A Markov Logic Network (MLN) is a knowledge base of first-order logic formulas with a weight attached to each formula [12]. MLNs can compactly represent regularities in the world and allow reasoning over these regularities. The weight of a formula in the knowledge base is a measure of how likely that formula is to occur given observations of the world.

In table 1-1 an example MLN is given consisting of three formulas. In this model the formulas do not conflict logically but semantically seem incorrect. The latter is reflected in the formula weights. The third formula has a negative weight, indicating that the formula is false most of the time.

These MLNs can be used to construct a cognitive model for robotic applications. MLN formulas can capture knowledge in various knowledge domains. In this work we focus on the task of object affordance inference based on description of objects. In chapter 2 we provide more in depth reasons for choice to use Markov Logic Networks to learn relations between object properties and affordances.

1-3 Affordance inference

Humans intuitively know how to remove obstacles to reach their goals. For example, if we encounter a pile of bricks we instinctively know what to do to reach our goal: pick up each brick and put them out of the way or shove the whole pile aside. Robots do not have this quality yet. In several science disciplines the phenomenon of 'immediately knowing what to do in certain situations' is known as *affordance inference* [13]. Affordance inference can be applied to anything. In this work we focus on affordance inference for objects.

The term *affordance* was first coined by Gibson in his 1966 book *The Senses Considered as Perceptual Systems* [14], to be further expanded upon in *The Ecological Approach to Visual Perception* [15]. Gibson defined affordance as *what the environment offers, provides or makes available to an animal*. For example, a small tunnel affords a child to crawl through, but it does not afford a man to crawl through as he does not fit. Gibson's notion of affordances takes into account the capabilities and intentions of an agent: to a child an apple affords to be thrown, but also affords nourishment. However, to a robot an apple does not afford nourishment at all as it is incapable of processing food. In the robotics context the animal is replaced by a robot that perceives its environment through a number of different types of sensors. The robot brain can fuse this sensor data to reason about action possibilities present in the environment.

The inclusion of a cognitive affordance inference step in the perception-action system of a robot allows it to reason more effectively about its world beliefs. It makes robots more robust to unseen situations or encounters. Enhancing a robot with this ability effectively allows it to adapt to a change in task parameters.

1-4 Zero shot

The affordance concept is tightly coupled with the Zero Shot Learning (ZSL) machine learning setting. Affordances are often derived from properties of objects as they are relatively invariant features. Whether an object has certain parts, a certain size or mass influences greatly what affordances it has to an agent. Looking at object properties (parts, state etc.) that define an object class, such as *teapot*, instead of just the object class allows for more fine-grained affordance predictions. In the teapot example, the classic machine learning models predict the class of the teapot based on similar looking classes seen during training. Predicting affordances requires more detailed properties of an object to be taken into account as a ceramic teapot can thrown, but a cast iron teapot might not be as it is too heavy.

Furthermore, robots usually do not have enough sensors on board to derive a full description of an object. For example, a robot might only have one camera. Based on monocular camera images only it becomes difficult to accurately estimate the density of an object. Furthermore, the environments that mobile robots operate in are often partially known and dynamic. It is unreasonable to assume that the objects a robot encounters during operation all fall within the classes seen during training. A robot must still be able to know how to use an object without it being required to have explicit of what the object is. Zero shot learning provides a solution to this problem.

ZSL – in reference to one- and few-shot learning – is the task of classifying samples from classes[3] not seen during (supervised) training. Essentially, in zero shot learning a classification model is served a description of a sample, in this case is an object, and must infer its class based on the object description, contextual information or semantic similarity of text [16]. Evidently, the more detailed a description is the better the classifier performs. However, often only a partial description of an object is available.

Farhadi *et al.* [2] introduced a paradigm shift in object classification by *describing* objects rather than *naming* objects. Descriptions of objects consists of common and discriminating attributes or properties of objects[4]. Common properties are often shared between similar classes of objects, whereas discriminating properties are often unique to a class. Figure 1-5 illustrates how object properties can be used to reason about similar looking objects.

Zero shot learning and lifelong learning work well together. Zero shot learners can handle the sparse and incomplete data that is typically found in lifelong learning. A Markov Logic Network can be used as a zero shot learner: it contains the initial FOL formulas that relate object properties to affordances. Subsequently, a lifelong learning technique called cumulative learning is used to expand the initial formulas, properties and affordances to incorporate newly observed relations, properties and affordances.

[3]We use *class* to refer to an object, i.e. a zebra or ball

[4]The words *properties* and *attributes* are used interchangeably to denote concepts (parts, physical, visual and state) belonging to an object.

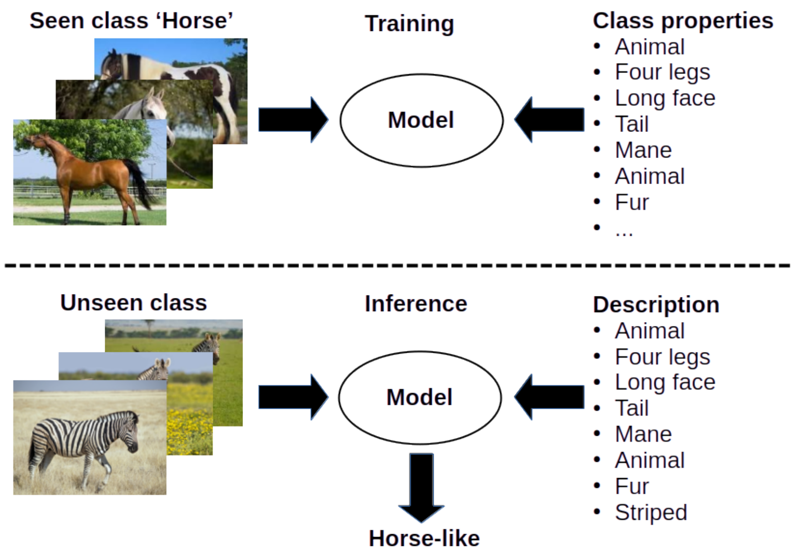


Figure 1-5: An overview of ZSL. The horse class has certain properties associated with it. These are captured in a learned model. During inference an unseen class is encountered. Based on a description the model can infer that the descriptions are similar. Thus it concludes that the object in the image is horse-like. It can attribute a random class name like 'c44' to the description, but without ever encountering a class label for the description the model cannot know that we call the class 'zebra'.

1-5 Cumulative Learning

The cumulative learning process uses prior knowledge to increase the breadth of available *a priori* knowledge over time [17]. Unlike traditional batch learning methods where models are trained once on static datasets, a cumulative learner improves its model incrementally as described in Figure 1-6. A cumulative learner applies inductive reasoning or inference to its observations to generate hypotheses. The form these hypotheses take on depends on the knowledge base. These hypotheses are compared against the prior-knowledge. Based on a set of rules, loss or other mechanic, the knowledge base is updated. In addition, the knowledge base can also be queried to predict any unknown knowledge based on the observations.

Embodied robots can apply cumulative learning to continuously learn on the job. Cumulative learning allows robots to not only learn from interactions with their environments, but to also passively learn from observations of their surroundings. For example, when a robot observes multiple times that a television is located in front of a couch, it can learn that there is a correlation between the location of a television and a couch. The next time this robot observes a couch it can infer that there is a high probability of a television being present in front of the couch.

This process is called *updating the beliefs* of a robot. Beliefs are fragments of a priori knowledge that are associated with a weight, a value to indicate its likelihood of being true. Updating the beliefs is thus the process of adjusting these weights based on observations of the world. In our previous example, if more television-in-front-of-couch situations are observed, the weight is increased. A weight is decreased if opposite situations are observed, i.e. televisions not in front of a couch. Using this method a robot can verify and adjust its internal cognitive model

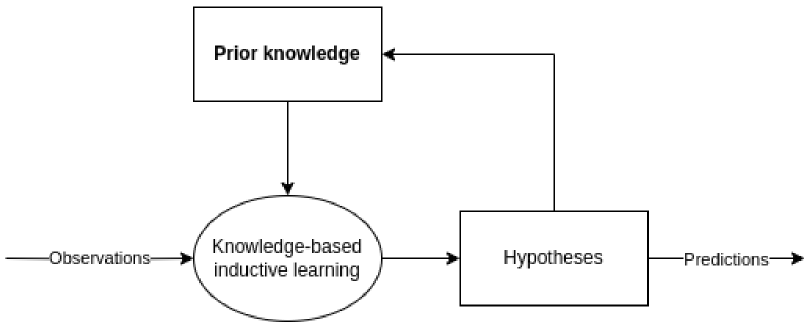


Figure 1-6: The cumulative learning process builds upon prior knowledge via inductive learning and hypothesis proposals to learn new prior knowledge from observations [17].

of the world with observations of the real world. Cumulative learning leverages this belief updating method to expand a basis of pre-programmed knowledge.

1-5-1 Aspects of Cumulative Learning

Thórisson *et al.* emphasise that knowledge acquisition in cumulative learning is incremental and sequential [18]. It is a continuous process of accepting new information and unifying it with existing information in small steps. Another important aspect of cumulative learning is that it must happen automatically. An ideal learner does not require human input. Thórisson *et al.* set out three core aspects of cumulative learning:

- Memory and knowledge management
- Temporal capacity and granularity
- Generality

Memory and knowledge management is making sure the storage capacity of a device is not exceeded. The knowledge based and newly learned knowledge must thus be efficiently stored. Several methods can be employed to reduce the size or increase the quality of the knowledge base: old-new unification, forgetting, compressing and correcting.

The core component *temporal capacity and granularity* is relevant for accepting new information. When should new information be accepted? How much information should be accepted before a new learning step is initiated? Ideally, a cumulative learner can accept any amount of information at any time to learn new concepts. In practice, robots often only observe small amounts of information relative to large datasets. A cumulative learner should be able to incorporate knowledge from both small and large datasets at any time.

Generality is the core component dealing with input data modality, domains of knowledge, knowledge transfer and abstracting concepts from current knowledge. Knowledge generality is essential for robots that must be able to adapt to dynamic environments. Generalised knowledge can be applied to tasks in unseen domains via knowledge transfer. Leveraging multiple modes of data input such as camera, audio, radar, LiDAR, force, temperature etc. can greatly increase the adaptation abilities of a robotic system.

Automatically learning new knowledge incrementally has the significant benefit of not requiring all knowledge to be learned all at once without dynamic updating. As a result, cumulative learners can function in partially known and dynamic environments.

Automatic learning is not without drawbacks. Catastrophic forgetting and erroneous unification are detrimental to the cumulative learning process. Catastrophic forgetting refers to the overwriting of (parts of) old knowledge during the learning of new knowledge, thus losing potentially critical information necessary to complete a task. Several mitigation methods to catastrophic forgetting exists such as rehearsal and freezing, but none offer a complete solution [19, 20].

Erroneous unification is another challenge in cumulative learning. When false information is fed to a system it can be erroneously incorporated in the existing knowledge. This can cause corruption of the knowledge base and may result in the failure of the system. A potential solution is to verify new information before it is allowed to interact with existing knowledge. Another potential solution is to apply a measure of truth certainty to a hypothesis formulated from observations. Then new knowledge that is more certain to be true overwrites old knowledge.

Within the context of object affordance inference, cumulative learning can be applied to adapt to new objects, new action capabilities and newly observed effects without having to retrain a model from scratch. Ideally, a robot could use cumulative learning during active deployment to learn the affordances for objects it has never seen before.

1-6 Research goal

In summary, encoding object properties and affordance knowledge in logical formulas and probabilistic reasoning balances the trade-off between machine and human interpretability. Combining Markov Logic Networks with cumulative learning leads to a powerful machine learning model that can learn new object affordances during operation. A natural application of a cumulatively learning MLN is the zero shot learning setting. This leads to our research goal: *Extending Markov Logic Networks with cumulative learning capabilities for zero shot object affordance inference.*

In this research we propose a cumulative learning extension for Markov Logic Networks called MLN-CLA. The algorithm allows Markov Logic Networks to adapt to and incorporate new evidence in the knowledge base. We showcase an application of MLN-CLA to a zero shot object affordance prediction setting for robots.

To achieve the research goal we demonstrate how MLN-CLA conforms to the three aspects of the cumulative learner as described in section 1-5-1 and how it does not. Specifically, in terms of memory and knowledge management we introduce unique knowledge updating strategies to merge new knowledge into the existing knowledge base. Additionally, we extend the Knowledge List and Knowledge Category concepts of Cui *et al.* [21] to manage and cluster knowledge efficiently [5]. We address the temporal capacity and granularity, and generality of MLN-CLA in our experiments. To reduce the complexity of the research goal, we limit our focus to creating a discrete and sequential cumulative learner. Furthermore, we assume that all predicates and domains are known a priori.

[5]See section 2-3

1-7 Outline

In Chapter 2 several relevant methods in the literature are discussed to give an overview of the status of the research field. The essential theoretical background of Markov Logic Networks is explained in Chapter 3. Here an in depth explanation is given of the main functions of MLNs: weight and structure learning. With an understanding of the theoretical base, the MLN-CLA algorithm is proposed for cumulative learning with MLNs in Chapter 4. In Chapter 5 two experiments are set out to test the functionality and performance of this cumulative learning algorithm as well as three knowledge updating strategies. Subsequently, in Chapter 6 the results of the experiments are described and analysed. The limitations of MLN-CLA and experiments are discussed in Chapter 7.

Chapter 2

Related work

In this chapter the context of extending MLNs with cumulative learning is discussed. The MLN is fit in the context of Stastical Relational Learning (SRL). A similar approach, the Bayesian Network (BN), is compared against MLN, what are their differences and similarities? Furthermore, what are the In addition to exploring SRL, several cumulative learning approaches are discussed in this chapter.

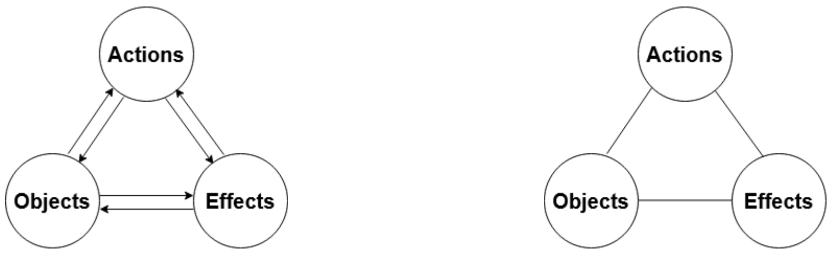
2-1 SRL

SRL is the research field of creating models of complex relational structures that can deal with uncertainty. These models are often based on a logic system such as first-order logic. Using the universal quantifier general properties and relations within a domain of knowledge can be captured in SRL models. The dependency structures of knowledge domains are often encoded in graphs. Graphs are compact representations of a distribution over a multi-dimensional space [22]. These are applied in search engines, object identification, causal inference, social networks and many other machine learning fields. The two main branches within SRL are Markov networks and Bayesian networks [23].

2-1-1 Bayesian Networks

Bayesian networks are graphical representations of a set of random variables and their conditional dependencies. They can model uncertain domains for probabilistic reasoning. In robotics BNs are commonly applied in perception, decision and planning and localisation and mapping. In perception BNs are often used to model relations between several objects in a scene.

In a Bayesian network the effect of a switch on a light bulb can be modelled. Flipping the switch results in the light turning on. When a robot enters a room and observes that the light is on, it can query the BN for the cause of the light being on. An inference algorithm, such as belief propagation [24], applied to a BN outputs a probability that the switch caused the light to be on, given the observation.



(a) Bayesian network [25]. **(b)** Markov network [32].

Figure 2-1: Two different representations of object, action and effect interactions.

Montesano *et al.* used a Bayesian network to model the interaction between objects, actions and effects [25, 26] as described in figure 2-1a. Their model learns many relations between objects, actions and effects from observations of interactions with the environment. Based on a set of visual object properties such as size, colour and shape, the model can infer possible actions and their effects for unseen objects. Additionally, it can update its beliefs, i.e. the strength of variable interactions in the network, based on observations of these interactions. As a result, it can adjust its predictions over time as it learns from new interactions.

A major drawback to Bayesian networks is that due to their graph structure and dependency representation, they cannot handle cyclic dependencies without additional extensions. Cyclic dependencies occur when two variables have an direct or indirect (via other variables) on each other. In this example, actions cause effects, effects result in changes in object state and object states influence action possibilities. As a result, whilst Bayesian networks do capture the direct interactions between objects, action and effects, they cannot capture indirect interactions between objects, actions and effects. The Bayesian network of Montesano *et al.* cannot learn new actions or effects without explicitly programming these in as prior knowledge and training the whole network from scratch again. However, other works have extended Bayesian Networks to include lifelong learning [27, 28, 29, 30, 31].

2-1-2 Markov Networks

Markov networks are, similar to Bayesian networks, a form of graphical representation of uncertain domains that allow for probabilistic reasoning [33]. In contrast to Bayesian networks, Markov network relations are *undirected* as described in figure 2-1b. This property allows for cyclic dependencies to be modelled. In addition, Markov networks can better exploit symmetric dependencies. Symmetric dependencies occur when a dependency between two variables is bidirectional.

The Markov properties of the network, as explained in chapter 3, allow for cumulative learning of new objects, object properties, actions etc. without having to retrain the whole network from scratch. Only the part of the network that encodes relations for the new knowledge has to be retrained, leaving the rest of the network unchanged. In essence, Markov networks can handle updates to the network based on partial information without changing unrelated relations. These properties make Markov networks particularly suitable for modelling the complex relations in object affordances in combination with cumulative learning. In contrast, Bayesian networks require changes to their graphical structure to update the network to incorporate new information [28, 30].

These advantages are exploited in Markov Logic Networks introduced by Richardson and Domingos [12, 34]. Research on MLNs has mostly focused on more efficient learning [35, 36, 37, 38] and inference algorithms [39, 40, 41, 42, 43, 44].

Several modifications to the MLN framework have been proposed for specific situations and problems that MLNs struggle with. Jain, Barthels and Beetz proposed an alternative to the fixed parameter representation of domains in MLNs [45]. Mittal *et al.* propose adaptive MLNs, a solution to the problem of MLN performance dropping for significant domain size differences between training and testing evidence [46]. Malhotra and Serafini continue this line of work and show that for two variable MLNs the issues with different training and test dataset sizes and distributions can be solved [47]. David and *et al.* extend MLNs with temporal reasoning capacities [48, 49]. Nyga and Beetz combine MLNs with a knowledge taxonomy and fuzzy logic to reason over knowledge concepts not yet incorporated into the knowledge base [50].

MLNs have seen many applications to solve complex relational problems. Choi *et al.* employ MLNs for social network searching [51]. Papadopoulos and Tzanetakis use an MLN to analyse music by exploiting the structural relations between audio signals of music instruments [52]. Ha *et al.* predict player goals and intents from observed actions in video games using MLNs [53]. Zhu *et al.* build a knowledge base of object properties with an MLN to predict possible actions and human poses for unseen objects [54]. In each application complex relations are captured in a compact model. The Markov properties of MLNs allows these models to be combined into one MLN that can be used in each application, without the need for any knowledge transfer often required in other machine learning techniques.

2-2 Neural Symbolic AI

Other machine learning techniques combine logic with neural networks [55]. This field of research is called Neural Symbolic AI [5]. Different from MLNs, these techniques represent logical formulas implicitly with neural networks. Marra and Kuželka combine MLNs with a neural network to implicitly represent FOL formulas [56]. Torrey *et al.* combine reinforcement learning with MLNs to transfer learned policies from one task to another [57].

Other approaches in Neural Symbolic AI represent the logical operators such as equality $\Leftrightarrow$, negation $\neg$, implication $\Rightarrow$ and even the quantifiers $\forall$ and $\exists$ with neural networks. One such approach is proposed by Donadello and Badreddine *et al.* called Logic Tensor Network (LTN) [58, 59, 60]. Figure 2-2 visualises the implementation of a logical conjunction of two predicates with tensors. Domains in LTN can range over images. The input modalities of LTN are broader than MLNs, which only has one input modality.

Another Neural Symbolic approach to complex relation representation is the Large Language Model (LLM). LLMs, such as BERT [61] and GPT-3 [62], learn extremely complex relations from enormous amounts of data with the help of Transformers [63]. This approach comes with enormous computational costs [64], but has shown impressive performance and capabilities.

A significant drawback of neural symbolic AI is the loss of explainability in comparison to more classical machine learning approaches [6, 4]. The neural networks that form the basis of neural symbolic AI are black boxes. It is unclear how certain outputs came about based on the inputs when looking at the neural network model. Markov logic networks are grey

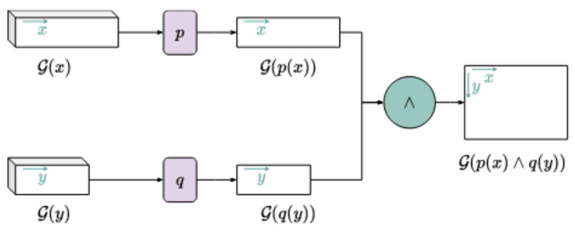


Figure 2-2: Visualisation of a conjunction of two predicates $p(x) \wedge q(x)$ over the variables x and y in a logic tensor network [60]. The conjunction operator performs an element wise operation on the predicate tensors $\mathcal{G}(p(x))$ and $\mathcal{G}(q(y))$.

box models. These are more explainable than black box models, but are not as transparent as white box models such as decision trees. Markov logic is more transparent than neural network logic because weights in Markov logic are attached to the formulas and not to a hidden, featureless nodes as in neural networks. Thus in Markov logic there is a more direct relation between the model output and the reasoning happening over the formula weights.

2-3 Online learning

Cumulative learning is a form of online or lifelong learning, a form of model adaptation to changing data distributions. It requires categorisation of incoming data[65]. The classical machine learning Support Vector Machines (SVM) technique can be extended to learn cumulatively [66, 67]. SVMs are used for classification in unsupervised learning. An SVM predicts which data belongs to a class and which data falls outside that class based on a learned decision function. Multiple SVMs can be trained to solve multi-class problems. Online SVMs continuously adapt their decision functions based on incoming data. Convolutional Neural Networks have been extended to learn cumulatively in a similar manner [68].

Cui *et al.* applied a similar approach to MLNs [21, 69]. They classify incoming data based on the existing knowledge base. Based on the classification, each data sample is put into a category. Each category in their algorithm represents an independent MLN that can be trained on the new data. This approach allows Cui *et al.* to create knowledge clusters within a Knowledge List to manage the knowledge base. Their knowledge clusters can be individually updated without having to retrain the whole model. They show that their model outperforms cumulatively learned SVMs in inference time and accuracy.

The approach of Cui *et al.* is not a cumulative learner. They require incoming evidence to be labelled with the knowledge cluster it belongs to. Our work lifts this requirement to turn it into a cumulative learner. We do not classify evidence based on an externally applied label, but classify or cluster the evidence purely based on the evidence domains.

Several other works on online learning of MLNs exist. These mainly focus on online learning of MLN parameters. These assume the initial set of clauses in the model to be complete and correct. Huynh and Mooney introduced a system for online structure learning of MLNs called OSL [70] to additionally adjust the model structure and learn its parameters based on new evidence. Their approach takes into account the model predictions in comparison to the ground truth to correct false positive and negative predictions using relational pathfinding

[71]. Whilst OSL can generate new formulas, it mostly generates formulas that are subsets of existing formulas. It does not restrict its structure learning search space with the existing formulas. In our work the structure learning search space is restricted through knowledge clustering.

We adopt the Knowledge List of Cui *et al.* but develop a unique method for automatic knowledge clustering with a new algorithm for creating and merging Knowledge Categories. We employ novel knowledge updating strategies to incorporate new knowledge that conflicts with the existing knowledge base.

2-4 Summary

In this chapter the Statistical Relational Learning field is explored. Markov Logic Networks are similar but fundamentally different to Bayesian Networks, both in theory and practical applications. Recently, the field of Neural Symbolic AI has seen much development. Methods in this field can learn large amounts of relations from huge datasets and can have multiple input modalities such as images and text. A major drawback of these approaches is the loss of explainability compared to traditional SRL approaches due to the black box nature of neural networks. Online learning with MLNs has seen some proposed methods over the years for both parameter and structure learning algorithms to learn MLNs. In the next chapter the theoretical background of the MLN framework is discussed.

Chapter 3

Markov Logic Network

Markov Logic Networks unify probabilistic reasoning with logic [12, 34]. This type of machine learning model is a powerful tool to solve problems that can be captured in generalised rules. MLNs are particularly well suited for (zero shot) affordance prediction of objects based on object properties or attributes. In this chapter the theoretical background of Markov Logic Networks is explored.

Knowledge encoded in graphs

How can dependencies and independencies between sets of variables be represented explicitly? An elegant solution is to create a graph G, a set of vertices or nodes V with edges or links E between them. The vertices represent the variables. The links represent dependencies between variables. A lack of a link between variables explicitly represents the direct independency of variables. Graphs can compactly represent a large network or relations that can be queried to answer a variety of questions.

In the Statistical Relational Learning research field two graphical models are dominant: Bayesian Networks and Markov networks. Koller *et al.* describe the differences between these two popular frameworks in great detail [23]. In this work the focus lies on Markov Networks for their Markov properties.

3-1 Markov Networks

Markov Networks, often called Markov Random Field (MRF), are a type of graphical model for reasoning over variable dependencies. In essence, they form a model of the joint distribution of a set of variables. The basis for the joint distribution representation by graph models is the notion of conditional independence of variables [23]. Conditional independence describes the situation in which the probability of a hypothesis is unchanged by new information as given in definition 3.1.

Definition 3.1 (Conditional independence)**.** Let A, B, and C be random variables. If A is the hypothesis and B and C are given, conditional independence of A and B given C is defined as:

$$P(A \mid B, C) = P(A \mid C)$$

Equivalently, written in another form, conditional independence is directly related to the joint probability of A and B given C:

$$P(A, B \mid C) = P(A \mid C)P(B \mid C)$$

where $P(A, B \mid C)$ is the joint probability of A and B given C. For a distribution P the notation $(A \perp B \mid C)$ is shorthand to say that A is conditionally independent of B given C. Conditional independence also holds for sets of discrete random variables. $\square$

Conditional independence is the bridge between sets of conditional statements such as "if A and given C then B provides irrelevant information" and graph representations of these sets of statements.

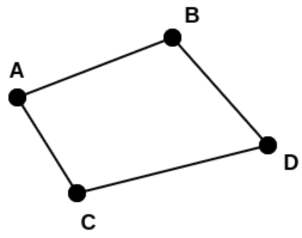
The Markov Network or MRF graph representation model $G(V, E)$ consists of a set vertices or vertices V, representing random variables having a Markov property and a set E of *undirected* edges, representing variable dependencies. Variables having a Markov property means that they do not depend on any past states, but only on the present state. In more detail, for the MRF case three Markov properties hold[1]:

1. **Pairwise Markov property**: any two variables that are not connected by an edge are conditionally independent when given all the other variables. That is, given variables A, B and C if $P(A \mid B, C) = P(A \mid C)$ is true, vertices A and B are not connected by an edge.
2. **Local Markov property**: a variable is independent of the rest of the variables given its direct neighbouring variables. Figure 3-1a describes this property for a Markov Network.
3. **Global Markov property**: similar to the local Markov property; two subsets of variables are conditionally independent given another *separating* subset of variables. Figure 3-1b provides an example of this property.

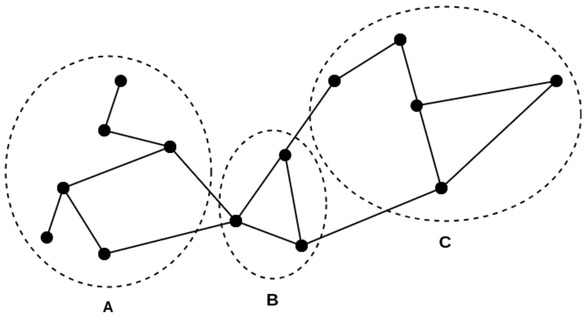
The global Markov property describes the notion of *separation* of sets of vertices in Markov Networks. If A, B and C are sets of vertices in the same Markov Network and A and C are both directly connected to B but not to each other, then A and C are separated from each other given B. In other words, if any path of vertices starting in A and terminating in C contains any vertex from B, A and C are separated by C. This is denoted as $(A \perp C \mid B)$. In figure 3-1b the global and local Markov properties are linked; for positive distributions the local and global Markov properties are equivalent [23]. These three properties are exploited in inference on MLNs and cumulative learning of MLNs.

Intuitively these Markov properties show that the influence of probabilities 'flow' or are propagated through the network. This 'flow' has no direction as Markov networks are undirected.

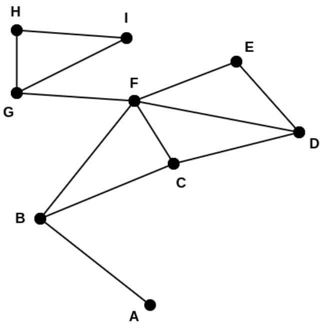
[1]For proofs of these properties the reader is referred to the work of Pearl and Paz [72]



(a) Local Markov properties example



(b) Global Markov property example



(c) Clique example

Figure 3-1: 3-1a demonstrates an example of local and pairwise Markov properties: $P(A \perp D \mid \{B, C\}), P(B \perp C \mid \{A, D\}), P(C \perp B \mid \{A, D\}), P(D \perp A \mid \{B, C\})$. A and D are conditionally independent according to the pairwise Markov property as they are not connected via an edge given B and C. In 3-1b the global Markov property is demonstrated by the set of vertices A being separated from C by B. There exists no path from A to C that does not contain a vertex from B. In 3-1c two cliques of size 3 are formed by the set of vertices $\{G, H, I\}$ and $\{B, C, F\}$. A clique of size 4 is formed by $\{C, D, E, F\}$.

The propagation of probabilistic influence is stopped by vertices that are part of the variables that are given. This concept is represented by the potential functions.

A Markov network contains a set of potential functions ϕ for each clique in the graph. A clique in an undirected graph is a set of vertices V such that every two distinct vertices in the set are connected via an edge. In other words, each possible pair of vertices in the set has an edge connecting the pair. Figure 3-1c demonstrates an example of a graph clique. Potential functions are non-negative, real-valued functions of the state of corresponding cliques. The overall joint distribution modelled by a Markov network is described in equation 3-1.

$$P(X = x) = \frac{1}{Z} \prod_{k}^{|Q|} \phi_k(x_{\{k\}}) \tag{3-1}$$

with $Q = \{q_1, q_2, \ldots, q_q\}$ the set of all cliques in the graph, k is the kth clique in the graph, $x_{\{k\}}$ the state of the variables (0 or 1 in the binary case) of the kth clique in the graph. Z, described in equation 3-2, is a normalisation function otherwise called the partition function.

$$Z = \sum_{x \in \mathcal{X}} \prod_{k} \phi_k(x_{\{k\}}) \tag{3-2}$$

In Markov networks, the clique potentials ϕ_k can be replaced by a sum of weighted features of the state transforming the joint probability into a log-linear model if $P(X = x) > 0$ for all variables x as shown in equation 3-3. This representation is more convenient for computation. An example of a *feature* in a Markov network is the grounded[2] version of a first-order logic formula.

$$P(X = x) = \frac{1}{Z} \exp\left(\sum_{i=1}^{N} w_i f_i(x) \right) \tag{3-3}$$

where N is the total number of features in the network. Each clique potential in eq. 3-1 is replaced by an exponentiated weighted sum of features of the variable states. In Markov Logic Networks the features of the variable states are the binary truth values of grounded versions of first-order logic formulas.

3-2 Markov networks and logic

Markov Logic Networks are based on Markov networks. In MLNs first-order logic formulas are converted to a graph. The formal definition of a Markov Logic Networks is given by Richardson and Domingos [12]:

Definition 3.2. A Markov Logic Network M is a set of pairs of formulas and weights (F_i, w_i) where F_i is a first-order logic formula and w_i is a real number. The set (F_i, w_i) together with the set of constants $K = \{k_1, k_2, \cdots, k_{|K|}\}$ define a Markov network $N_{M,K}$ (eq. 3-1 and eq. 3-2) as follows:

[2] See section 1-2. Grounding is the replacement of variables by constants in predicates.

1. The Markov network $N_{M,K}$ consists of a vertex for each possible grounding of each predicate appearing in M. The value of the vertex is binary, that is 1 if the ground atom is true, and 0 if false.
2. The Markov network $N_{M,K}$ consists of one *feature* $f_{i,j}$ for each possible grounding j of each formula F_i in M (see fig. 3-2). The value of this feature is 1 if the ground formula is true, and 0 otherwise. Each feature has a real valued weight w_i associated with it.

A feature in a Markov network is the representation of a ground first-order logic formula of the MLN in the graph. One formula can have multiple different grounded versions depending on the set of constants K as demonstrated in figure 3-2 and 3-3. Thus one formula can spawn multiple different features. Each feature has a truth value $f_{i,j} \in \{0, 1\}$ depending on the given evidence. The weight w_i of formula F_i is the same for each feature $f_{i,j}$ of the same formula F_i.

A set of ground atoms or predicates that have truth values assigned is called evidence. From definition 3.2, eq. 3-1 and eq. 3-2, the probability distribution over the evidence x and a ground Markov network $N_{M,K}$ is given by 3-4.

$$P(X = x) = \frac{1}{Z} \exp\left(\sum_{i=1}^{F} w_i n_i(x)\right) \tag{3-4}$$

where F is the set of formulas in the MLN, $n_i(x)$ is the number of true groundings of F_i in the evidence x [12].

Figure 3-2 shows an example of a ground Markov Logic Network of one formula consisting of four predicates ranging over five domains. The formula consists of only predicates and variables. The ground MLN consists of only the features of each formula in an MLN. The features are derived from the parent formula. Each feature is a possible grounding of the formula that can be either true or false. The number of true features in an MLN determines the parent formula weight. Features can overlap if two formulas range over one or more of the same domains as shown in figure 3-3. Each edge in the ground MLN represents an OR relation between the vertices as each formula in the MLN is converted to its conjunctive normal form (CNF) before the grounding operation. For example: $A \Rightarrow B$ turns into $\neg A \vee B$. Every first-order logic formula can be converted to a CNF equivalent [73].

Within the MLN framework, constant names are unique. However, it is possible in MLNs to add an `Equality(x, y)` to signify that `x` and `y` refer to the same thing or concept. This can be useful in applications such as object identification [12] or tracking, where it is important whether to different instances refer semantically to the same object.

3-2-1 Understanding weights

The weights associated with formulas are not intuitive to understand. In Markov Logic networks formula weights correspond to how strict a constraint on a formula it is. Each formula in an MLN is universally quantified, meaning that for all instances of a domain the formula holds true. The weights soften the universal quantification. If all weights in the MLN were equal and infinite, the MLN would represent a pure first-order logic knowledge base.

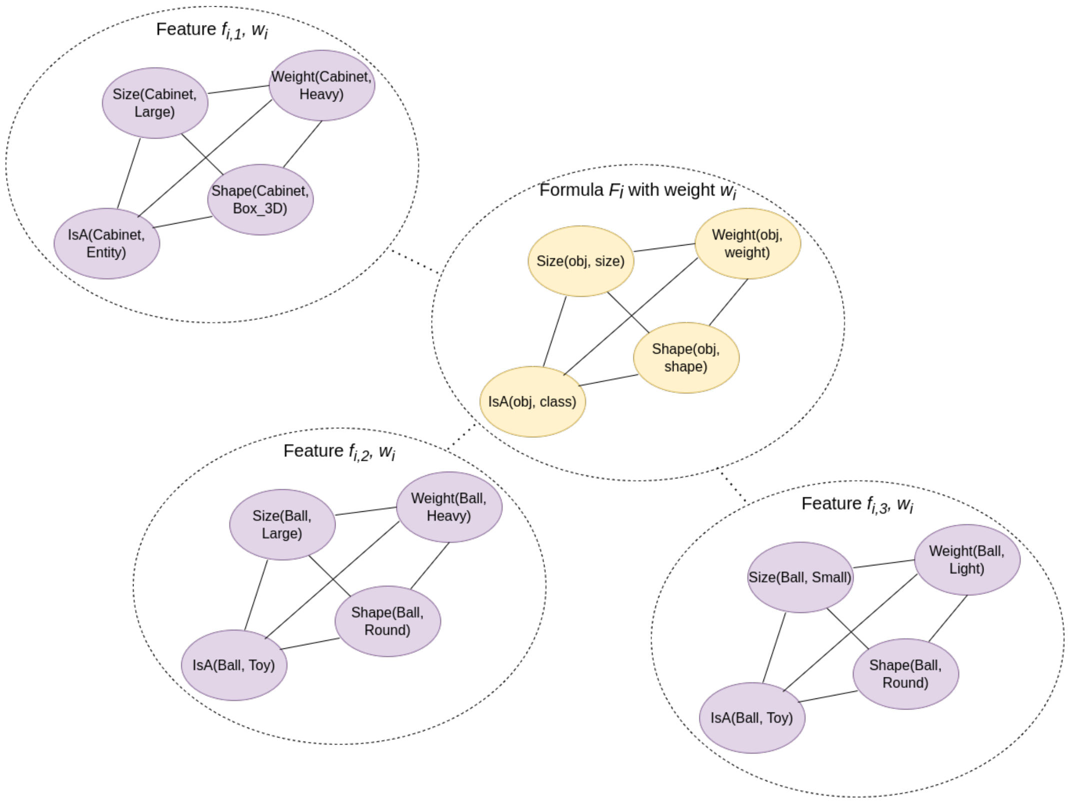


Figure 3-2: A ground MLN example for the formula `Size(obj,size)`$\wedge$`Weight(obj,weight)`$\wedge$ `Shape(obj,shape)` $\Rightarrow$ `IsA(obj,class)`. Each predicate in the formula corresponds to a vertex in the graph. Together the predicates of a formula form a clique. In a ground MLN only the ground features (purple) of a formula are present, not the formula itself (yellow).

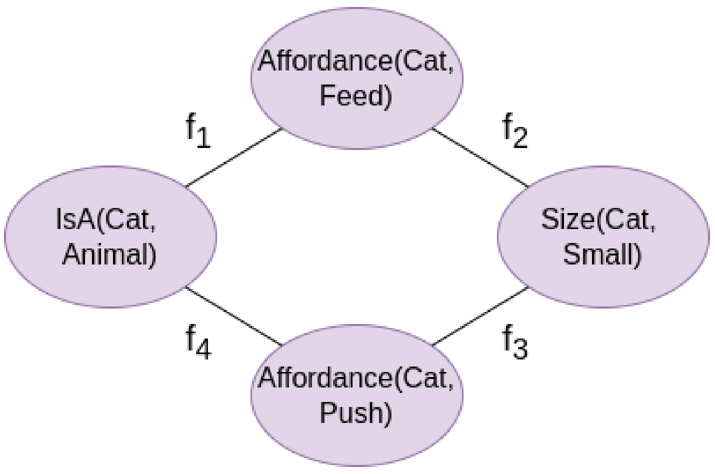


Figure 3-3: An example of overlapping features in a ground MLN for the formulas `IsA` $\Rightarrow$ `Affordance` and `Size` $\Rightarrow$ `Affordance`. The ground predicates connected by an edge appear together in the same formula and form a feature. f_1, f_4 are features of the first formula, f_2, f_3 are features of the latter. Features can overlap, creating links between formulas. f_1 and f_2 are of different formulas but are connected because of this overlap. When querying the affordance predicate, both the `IsA` and `Size` ground predicates are sampled for their truth value to answer the query.

MLN weights represent the log probability of the evidence satisfying the formula. A higher formula weight corresponds to a greater difference in log probability between evidence that satisfies the formula and evidence that does not, all other variables being equal. A formula is satisfied if it is true at least once given the evidence. In other words, the weight of a formula is the log odds between a world where the formulas is true and world where the formula is false. For example take figure 3-3. Numerically, if we assign a weight of 2.5 to the formula 'a cat can be fed', then a world where n cats can be fed is $e^{2.5n}$ more probable to occur than a world where no cat can be fed.

Formulas with weight zero can be safely ignored or removed from the MLN as they do not influence the probability distribution [46]. Weights can be negative as well. A negative weight indicates that the opposite of the corresponding formula is true. First-order logic formulas can simply be negated to flip the weight to positive again: $(-w_i, F_i) \equiv (w_i, \neg F_i)$.

Weight interpretation becomes more complicated if formulas share variables. If two formulas range over the same variables or domains, such as in figure 3-3, there is no longer a one-to-one correspondence between weights and probabilities of formulas as explained previously [12]. Although weights no longer directly correspond to formula probabilities, the probabilities of all formulas, given the evidence, do still collectively determine the weights. This allows the weights to be learned from evidence using weight learning algorithms.

3-3 Inference

Inference is the process of reaching a conclusion on a question or query. For MLNs a query is in the form of a predicate. Any question that can be asked of an MLN is of the form described in equation 3-5 meaning "return the probability of a formula F_1 being true given that another formula F_2 is true". A formula or query can also be a unit clause, i.e. consist of a single predicate.

$$\begin{aligned} P(F_1 \mid F_2, M, K) &= \frac{P(F_1 \wedge F_2 \mid M, K)}{P(F_2 \mid M, K)} \\ &= \frac{\sum_{x \in \mathcal{X}_{F_1} \cap \mathcal{X}_{F_2}} P(X = x \mid M, K)}{\sum_{x \in \mathcal{X}_{F_2}} P(X = x \mid M, K)} \end{aligned} \tag{3-5}$$

where F_1, F_2 are first-order logic formulas, M is an MLN containing at least F_1 and F_2, K is a finite set of constants defining the grounded M_K, x the evidence $\mathcal{X}_{F_i}$ is the set of worlds where F_i is true, and $P(X = x \mid M, K)$ is given by equation 3-4. Again, directly calculating equation 3-5 is #P-complete and thus intractable. Fortunately, approximation algorithms exist such that inference is tractable.

The output of inference given a query predicate on an MLN given the evidence is the marginal probability of the query predicate. If the number of formulas or number of constants in domains of an MLN are large, inference over the whole network becomes intractable. However, the Markov properties of Markov networks enable faster inference. In Markov networks, a variable is independent of the rest of the network given its direct neighbouring variables. The set of a variable and its direct neighbouring variables together is all that is required to be sampled to answer a query during inference on MLNs [12]. This set is called the Markov blanket of a vertex or variable in a graph.

The joint probability distribution can be sampled or queried to give the marginal probability of the query being true. The marginal distribution gives the probability distribution of a subset of variables in a network. The marginal probability of a query is the probability that the query is true, independent of the state of the rest of the network. This probability is the output of inference on a network.

Given a set of evidence, the probability distribution of the query predicate y for evidence x is:

$$P_{ground}(Y = y \mid X = x; w) = \frac{1}{Z_x} \exp\left(\sum_{i=1}^{m} w_i n_i(x, y)\right) \tag{3-6}$$

where X is the set of all evidence, Y is the set of predicates in the MLN, m is the number of formulas, w_i is the weight of the formula F_i and $n_i(x, y)$ is the number of satisfied groundings of F_i for a given pair of $(x, y) \in (X, Y)$. Z_x is the normalisation constant given by equation 3-7

$$Z_x = \sum_{y'} \exp\left(\sum_{i=1}^{m} w_i n_i(x, y')\right) \tag{3-7}$$

Given equation 3-6 and equation 3-7 the marginal probability of a ground atom A being *true*, that is $A = 1$, can be calculated via equation 3-8.

$$P_{ground}(A = 1) = \frac{Z_{A=1}}{Z_{A=0} + Z_{A=1}} = \frac{1}{1 + \frac{Z_{A=0}}{Z_{A=1}}} \tag{3-8}$$

where $Z_A = a$ means the normalisation constant is constrained to the value a in all truth assignments of the variables in Z.

Inference is ran in two situations: for training and when querying a trained MLN. During training, inference is ran on the current model to compute the training set likelihood of the current weights and the current gradient. During querying, an already learned model is used to run inference on. For querying in non-real time applications the inference time is not highly relevant and thus the task of optimising the inference algorithm is less important. For inference during training a highly efficient algorithm has significant impact on training time. An efficient inference algorithm preferably does not trade off accuracy for speed.

In this research the default MC-SAT algorithm [74] of Alchemy[3] is used for inference. MC-SAT combines Markov Chain Monte Carlo (MCMC) with satisfiability solving[4]. Our cumulative learning algorithm introduced in chapter 4 can work with any inference algorithm compatible with MLNs.

[3]See section 5-1.

[4]For more information the reader is referred to the work by Poon and Domingos [74].

3-4 Learning methods

In Markov Logic Networks the formula weights and the formulas themselves can be learned from evidence using machine learning. Learning weights or the structure of MLN formulas is done based on one or multiple databases. These are files containing evidence of the relations between constants. These relations are encoded in the predicates declared in MLNs. A database consists of only ground atoms, i.e. no variables or clauses may occur in a database. This and other rules are described in section 3-4-3. For cumulative learning the closed world assumption holds for databases. This implies that any ground atom not appearing in the database is assumed to be false. The reverse also holds; if any ground atom appears in a database, it is assumed to be true [12]. Effectively, an evidence database of n ground atoms is a vector $x = (1, 0, 1, 1, ..., x_n)$ where each entry is the truth value of the ground atom.

3-4-1 Formula weight learning

Formula weights determine which formula is more likely to be true given evidence. The optimal weights can be learned from data. The optimal weights are the *maximum a posteriori* (MAP) weights or the weights that maximise the product of their prior probability and the evidence likelihood [38]. Formula weights that can be learned have corresponding evidence in a given database, otherwise they are not learned. During formula weight learning the log-likelihood is optimised [12]. Richardson and Domingos demonstrate that the derivative of the log-likelihood with respect to its weight is the difference between $n_i(x)$, the number of true groundings of the ith formula in the evidence database x, and the expectation of the formula over all possible evidence databases x' according to the current model as described in equation 3-9. Here $P_w(X = x')$ is the current likelihood computed using the current weight vector $w = \{w_1, w_2, \dots, w_{|F|}\}$. x is an evidence database of the set of all possible databases x'. $\frac{\partial}{\partial w_i} \log P_w(X = x)$ is the gradient of one formula weight. The gradient over all weights of the MLN is the vector $\mathbf{g}$. During learning, weights are updated in each learning step according to equation 3-10.

$$\frac{\partial}{\partial w_i} \log P_w(X = x) = n_i(x) - \sum_{x'} P_w(X = x') n_i(x') \tag{3-9}$$

$$\mathbf{w_{t+1}} = \mathbf{w_t} - \eta \mathbf{g} \tag{3-10}$$

However, as Richardson and Domingos point out, counting the number of true groundings of a formula in a database is intractable. They state that counting the number of true groundings of a formula is #P-complete in the length of the clause. No algorithm exists to solve #P-complete problems. In addition, computation of the expected number of true groundings $\sum_x' P_w(X = x') n_i(x')$ is also intractable as it requires inference over the model that is yet to be learned.

Richardson and Domingos propose a different approach: optimising the pseudo-log-likelihood as proposed by Besag [75] instead of the log-likelihood. The pseudo-log-likelihood is the product of the log-likelihood of each variable given the variables of neighbouring variables in the evidence. For large domains the number of true groundings of a formula $n_i(x)$ is counted approximately by uniformly sampling groundings of the formula. If a sampled ground formula is true given the evidence, then $n_i(x)$ increases by one. Sampling only a small portion of possible formula groundings from the data enables approximation of the true distribution. The number of samples to take is a hyper-parameter that can be tuned. The true $n_i(x)$ can thus be approximated to calculate the log-likelihood with.

This original formula weight learning algorithm of Richardson and Domingos is called *generative* weight learning. After each iteration the gradient vector $\mathbf{g}$, calculated for each formula with eq. 3-9, is scaled by a learning rate η to update the weight vector of all formulas $\mathbf{w}$ resulting in the new weight vector $\mathbf{w_{t+1}}$, as described in Equation 3-10. The learning rate η determines how fast the weights converge. A large learning rate allows faster convergence towards the global optimum, but it can cause overshoot resulting in oscillation of the gradient direction. Choosing a smaller learning rate reduces the chance of overshooting, but will slow down convergence time.

Singla and Domingos proposed an alternative weight learning algorithm called *discriminative* weight learning. In discriminative learning the conditional log-likelihood (eq. 3-11) of the query predicates y given the evidence x is maximised. In contrast to generative learning, discriminative learning does not optimise the join likelihood of all predicates but only of the query predicates. A drawback of this method is the requirement that the query predicate must be *a priori* known. As discussed in chapter 4, this requirement is troublesome for the cumulative learning extension of MLNs. However, for batch learning methods the discriminative weight learning method has been shown to be more effective than the generative method, because the formula weights are optimised for a given query instead of generically optimising weight gradients to zero [35, 38].

$$\frac{\partial}{\partial w_i} - \log P(Y = y \mid X = x) = E_w[n_i] - n_i \tag{3-11}$$

3-4-2 Structure learning

An MLN is created from expert knowledge about a specific knowledge domain. This knowledge can be incomplete or even incorrect. The formulas of an MLN can revised to correct mistakes and extended to add new formulas. This process changes the *structure* of the Markov network by adding or removing vertices and edges. The structure of an MLN can be learned from evidence data. Structure learning can be performed from scratch, i.e. add learned formulas to an empty MLN.

The basic structure learning algorithm is akin to inductive logic programming, except that it directly optimise the likelihood of the data. It starts with the ground MLN, then it greedily combines both ground and non-grounded predicates to improve the joint distribution fit of the evidence based on a score. This is done for all formulas and possible predicate combinations of the MLN. Despite its simplicity, a significant problem with this method is the necessity of calculating the new score after each combination step. Another drawback of structure learning is the poor duration scaling as the number of possible predicate combinations grows exponentially with the addition of more predicates to an MLN.

Nevertheless, learning new formulas and updating existing formulas is a powerful tool. In cumulative learning it allows for error correction and mining new relations between previously independent domains. Multiple MLN structure learning algorithms exist [36, 76, 37, 77]. In this work the structure learning algorithm of Kok and Domingos is used [36].

3-4-3 Evidence

To learn new concepts a cumulative learning algorithm requires new input data. An ideal cumulative learner can continuously learn from a stream of input data with an infinite learning window size [18]. However, no such algorithm exists yet. Our cumulative learning algorithm learns incrementally from new databases one at a time, where the learning window size is as large as the size of the database. This means that a whole new database is ingested at once for every learning step.

To learn from any database, they must conform to several constraints.

- The data must be written in FOL.
- The FOL notation for predicates, variables and constants follows Alchemy grammar[5].
- Duplicate evidence is possible but has no effect on formula weights.
- Only ground atoms are allowed as evidence, i.e. only predicates of the form `HasSize(Shoe, Small)`.
- With one exception: Domains of predicates must be declared in the database, i.e. `HasSize(obj, size)` must occur once (domains are known assumption).
- All evidence can be either true or false (denoted by an '!' in front of the ground atom), i.e. no fuzzy evidence.
- Evidence not in the database is considered false (closed world assumption).

3-5 Summary

In this chapter the definition and several properties of Markov Logic networks were discussed. Markov Logic Networks are based on Markov networks. MLNs combine first-order logic with probabilistic reasoning. The formulas in an MLN form a graph with pairwise, local and global Markov properties. Each vertex in the graph corresponds to a predicates in a formula. The vertices are connected by an edge if the two predicates appear together in a formula. All edges in the graph denote the OR relation as each formula is converted to its conjunctive normal form. Formula weights in an MLN can be trained on evidence with algorithms such as MC-SAT to optimise a log likelihood to fit the joint probability of the evidence. In addition, the structure of the network can be learned from evidence. The evidence consists of only ground atoms, except for predicate declarations for each evidence predicate.

[5]https://alchemy.cs.washington.edu/user-manual/4Syntax.html

Chapter 4

Cumulative learning with MLNs

This chapter delves into the concepts and mechanisms of the Markov Logic Network Cumulative Learning Algorithm (MLN-CLA). First the fundamental concepts and tools for cumulative learning with Markov Logic Networks are set out. With these tools in hand, our cumulative learning algorithm is explained step-by-step. Figure 4-1 illustrates a simplified overview of MLN-CLA.

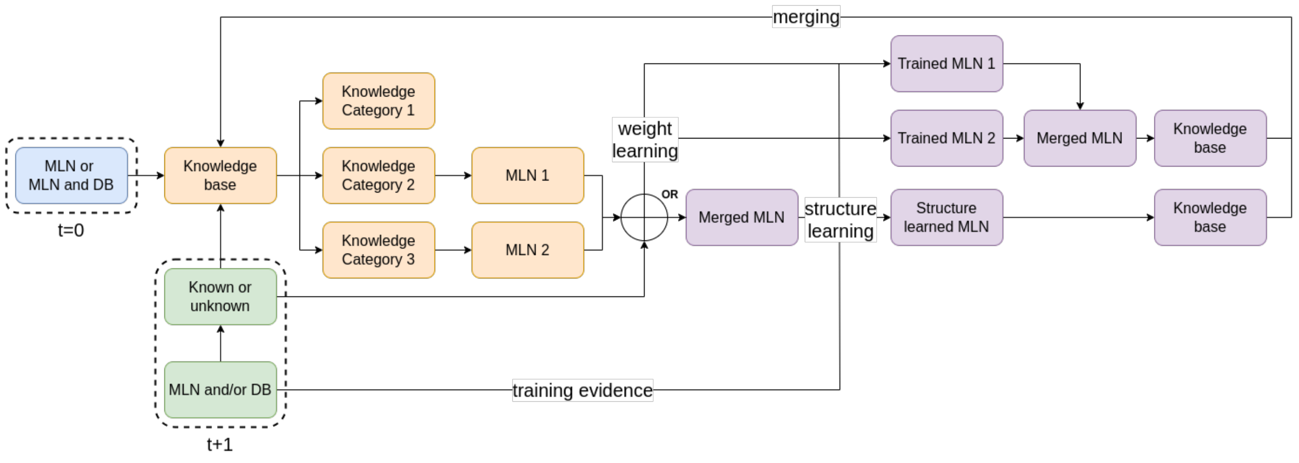


Figure 4-1: Simplified overview of MLN-CLA in terms of MLNs and databases. In the first step an MLN or an MLN and an evidence database are given as initial input to create a knowledge base with. In the next time step a new MLN, evidence database or both are introduced to the knowledge base. The new information is compared against the knowledge base to determine for each piece of information whether its is known or unknown. If all information is known then structure learning is performed, otherwise only formula weight learning is performed. After comparing against the knowledge base, each piece of information is put into a Knowledge Category. Only the Knowledge Categories changed by the new information are converted to independent MLNs. Subsequently, if structure learning is performed then the MLN splits are first merged together before structure learning on the new information. Otherwise, the MLN splits formula weights are trained on the new evidence. Afterwards, the splits are merged together and a new knowledge base is created from the merged or structure learned MLN. Finally, the new knowledge base is merged with the initial knowledge base to incorporate the newly learnt formula weights or structure.

4-1 The ingredients

First of all, to learn cumulatively an existing knowledge base and new evidence in first-order logic are required. The knowledge base is required to make sense of any newly received evidence. Everything new is compared to the knowledge base to determine how to incorporate it. Without a pre-existing knowledge base or any *a priori* knowledge it is impossible to categorise new knowledge as there is nothing to compare new evidence against [65]. Cui *et al.* solved this problem by a priori labelling each piece of evidence with a 'knowledge identifier' label [21]. In their work they categorise incoming evidence into clusters, called Knowledge Categories, based on this knowledge identifier. To manage these clusters they introduce a Knowledge List. In MLN-CLA we extend these concepts of Knowledge Categories and Knowledge Lists. We lift the requirement of labelling data beforehand through automatic categorisation based on domains of constants instead of knowledge identifiers. This includes automatic categorisation and incorporation of unknown predicates and domains from evidence. Furthermore, in contrast to Cui *et al.*, we show how to merge Knowledge Categories together to more efficiently cluster formulas in the Knowledge List.

Within the context of MLN cumulative learning new evidence consists of one or a combination of:

- new formulas (from an MLN)
- new ground atoms (from a database)
- new predicate declarations that define new relations over either existing or new domains of constants (from either an MLN or database)

MLN-CLA allows for two types of sources of new evidence, either from an MLN or an evidence database of ground atoms. At each time step a combination of any type of evidence can be presented to the cumulative learner. The software used for MLN learning[1] only requires predicates used in formulas in any MLN to be declared beforehand. Within MLN-CLA this requirement is extended to evidence databases. Predicates in evidence must also be declared to indicate to which domains the arguments of ground atoms belong. For example: `IsA(object, category)` is a predicate declaration that defines the 'IsA' relation over the domains 'object' and 'category'. The domains are filled with constants by evidence ground atoms such as `IsA(Cat, Animal)`, indicating by argument position that 'Cat' belongs to the 'object' domain and 'Animal' belongs to the 'category' domain. This assumption enables the cumulative learner to incorporate new relations over new or existing domains in the knowledge base. Structure learning can be applied to newly discovered predicate declarations to construct formulas from these and other predicates.

To manage knowledge and compare new evidence against the knowledge base is the basis of a cumulative learner [18]. Before delving into the Markov Cumulative Learning Algorithm, two types of knowledge containers must be defined first. These containers form the basis in managing the knowledge base for cumulative learning. First, the concept of the Knowledge List (KL) is introduced. Subsequently, the second knowledge container, called the Knowledge Category (KC), is introduced.

[1]Described in detail in section 5-1

4-1-1 The Knowledge List

A Knowledge List $\mathcal{L}$ consists of a set of predicate declarations $\mathcal{P}$, a mapping $\mathcal{D}$ of constants to domains, and a set of Knowledge Categories $\mathcal{C}$ (see section 4-1-2). The definition of a Knowledge List is given in definition 4.1.

Definition 4.1 (Knowledge List).

$$
\begin{aligned}
&\mathcal{P} = \{P_1(D_A), P_2(D_B, D_C), P_3(D_D), P_4(D_E, D_F, D_G), \ldots, P_W(D_X)\}\\
&\mathcal{D} = \{\\
&\qquad D_A \mapsto \{k_{a1}, k_{a2}, \ldots, k_{aA}\}\\
&\qquad D_B \mapsto \{k_{b1}, k_{b2}, \ldots, k_{bB}\}\\
&\qquad\ \vdots \qquad\qquad \vdots\\
&\qquad D_X \mapsto \{k_{x1}, k_{x2}, \ldots, k_{xC}\}\\
&\quad\ \}\\
&\mathcal{C} = \{c_1, c_2, c_3, \ldots, c_Y\}
\end{aligned}
$$

where W is the number of predicate declarations, X the number of domains in the predicate declarations, A, B and C the number of constants k in each respective domain and Y the number of Knowledge Categories in the Knowledge List. □

Building and maintaining a Knowledge List is required to keep track of all known knowledge elements. When new evidence is fed into the cumulative learner it can be compared against the Knowledge List to determine whether each piece of new evidence is known or (yet) unknown. During the MLN-CLA process the Knowledge List is updated according to the new evidence. Only the elements in the Knowledge List that are affected by the new evidence will be updated. This method enables selective updating of knowledge and incremental learning of new knowledge. In MLN-CLA the initial Knowledge List is constantly updated by merging it with another Knowledge List created from MLNs trained on the new evidence as illustrated in figure 4-1.

Example 1 shows an example of a Knowledge List with several predicate declarations and domain to constant mappings. In this example the three predicate declarations all share a domain. Each predicate defines a relation between an `object` and another concept, i.e. the size or shape of any object. Intuitively, because these predicates have a domain in common they define knowledge within the same context. This property of the predicate and domains declaration is exploited in the Knowledge Categories to structure the information in the Knowledge List.

Example 1.

$$
\begin{aligned}
&\mathcal{P} = \{\text{Size(object, size), Shape(object, shape), Affordance(object, action)}\}\\
&\mathcal{D} = \{\\
&\qquad \text{object} \mapsto \{\text{Ball, Glass, Shoe, Chair}, \dots\}\\
&\qquad \text{size} \mapsto \{\text{Small, Large}, \dots\}\\
&\qquad \text{shape} \mapsto \{\text{Round, Sphere, Cylinder, Cube, Flat, Convex}, \dots\}\\
&\qquad \text{action} \mapsto \{\text{Push, Throw, Pull, Open, Close, Hit, Pick, Place}, \dots\}\\
&\quad \}\\
&\mathcal{C} = \{C_1, C_2, C_3, \dots, C_N\}
\end{aligned}
$$

4-1-2 Knowledge Categories

A Knowledge Category C consist of an index or identifier i, a set of triplets (F_j, w_j, z_j) of associated formulas F_j, weights w_j and formula evidence counts z_j, and a set of category domains $\mathcal{D}_i$. A Knowledge Category contains a finite number of knowledge triplets and domains. All knowledge triplets in a Knowledge Category contain formulas on the same concept(s). The definition of a Knowledge Category is given in definition 4.2. Knowledge Categories are always part of a Knowledge List as they share the same context. The index i is used to keep track of the Knowledge Category within a Knowledge List over time.

Definition 4.2 (Knowledge Category).

$$
\begin{aligned}
\text{index} &= i\\
(F, w, z) &= \{(F_1, w_1, z_1), (F_2, w_2, z_2), \dots, (F_N, w_N, z_N)\}\\
\mathcal{D}_i &= \{D_A, D_B, \dots, D_X\}
\end{aligned}
$$

where i is an integer index, N is the number of knowledge triplets in the category, and X is the number of domains in the formulas of the category. □

Example 2 shows a possible Knowledge Category within the Knowledge List from example 1. It contains two formulas relating the size of an object to possible actions and two unit clauses `Size(o,s)` and `Affordance(o,a)` that model the prior probability of the predicate. The constants `Large`, `Push`, `Small` and `Throw` are mapped to their respective domains `object`, `size` and `action` via the Knowledge List. Each formula in the knowledge triplets is accompanied by its weight and evidence count. The evidence counts suggest these formulas were learned from different evidence databases as their evidence counts vary for each formula. In evidence counting only the predicates of a formula are counted in the evidence, not the constants (see algorithm 4. If these formulas were learned from the same evidence database, the counts would be identical for the first two triplets as well as for the unit clause triplets. The unit clause triplet evidence counts would sum up to the evidence count of the first two triplets.

Example 2.

$$\begin{aligned}\text{index} &= 3\\ (F, w, z) =&\{(\text{Size(o, Large)} \implies \text{Affordance(o, Push)}, 0.563, 8),\\ &(\text{Size(o, Small)} \implies \text{!Affordance(o, Throw)}, -1.27, 4),\\ &(\text{Size(o, s)}, 0.2, 1), (\text{Affordance(o, a)}, 0.3, 1)\}\\ \mathcal{D}_3 =&\{\text{object}, \text{size}, \text{action}\}\end{aligned}$$

In Markov logic, all predicates have at least one argument. A predicate without an argument is a proposition and is part of Propositional logic. A predicate can range over a finite number of domains in Markov logic. Predicate names are unique: `Affordance(object, action, effect)` and `Affordance(object, action)` cannot be both defined in one MLN. Predicate arguments are not unique. A predicate can range over one domain but have two arguments, such as `NextTo(object, object)`. Instead of relating two domains together, these kind of predicates define relations within one domain. Similarly, a formula can be formed by any number of predicates and the predicates of a formula are not unique within the formula: `Affordance(o, a)` $\Leftrightarrow$ (`NextTo(o1, o2)` $\wedge$ `NextTo(o2, o1)`). The same predicates can occur in other formulas as well.

A Knowledge Category in a Knowledge List is unique in a Knowledge List. It contains all formulas in a Knowledge List that relate to the same concept, forming a cluster. Knowledge Categories are initially created from one formula. A Knowledge Category can contain multiple different formulas. Determining which formula to add to which Knowledge Category is the most important mechanism of MLN-CLA. Any formula in an MLN belongs to only one Knowledge Category in the Knowledge List.

To determine to which Knowledge Category a formula belongs, the algorithm can look at either the predicates or domains of a formula. All predicates and the domains they range over are known through the MLN predicate declarations. A Knowledge Category is made unique by either the set of predicates or domains of the formulas in the category. In practice, there are more relations between domains than there are domains defined in an MLN. We made the design choice for our algorithm to make Knowledge Categories unique by their domain set, where the set is formed by all domains of each predicate in a formula as described. In MLN-CLA the set of domains $\mathcal{D}_i$ of a Knowledge Category fully determine the category as described by definition 4.3. Figure 4-2 illustrates example Knowledge Categories. In the figure several categories contain the same set of domains.

Definition 4.3. $\mathcal{C}_i$ is a Knowledge Category with index i formed by the set of domains $\mathcal{D}_i$ of predicates $\mathcal{P}_i$ that occur together in a formula. □

Figure 4-2 visualises the uniqueness of Knowledge Categories based on their domains. Some examples of predicates ranging over the same sets of domains are: `Large(obj)`, `Small(obj)`, `Heavy(obj)` or `Manages(person, person)` and `Family(person, person)`.

As a consequence of theorem 4.3, one predicate can belong to multiple Knowledge Categories. How do Knowledge Categories differ from each other? Two Knowledge Categories can be compared against each other simply by comparing their respective sets of domains $\mathcal{D}_i$ (fig. 4-2). Definition 4.4 describes in what circumstance two independent Knowledge Categories contain information on the same knowledge concept.

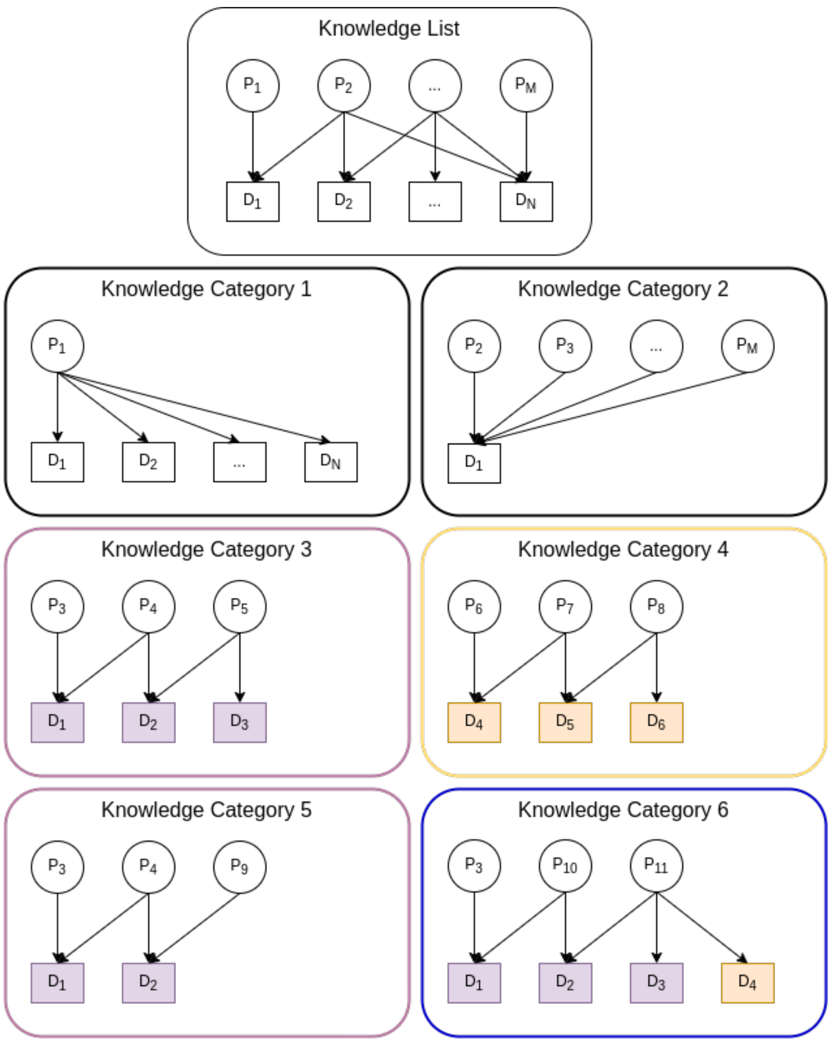


Figure 4-2: Illustration of the differences between predicates and domains of several example Knowledge Categories of a Knowledge List. KC 1 consists of a formula of one predicate ranging over N domains. KC 2 consists of a formula of M predicates ranging over the same domain D_1. The domains of KC 2 form a subset of KC 1, whereas the predicates of KC 1 form a subset of KC 2. Either set can make a Knowledge Category unique. KCs 5 consists of a formula of three predicates ranging over two domains that are part of KC 3. The domains of KC 5 are a subset of the domains of KC 3. The domains of KC 6 intersect with both KC 4 and 5. The domains of KCs 2-6 are all subsets of the domains of KC 1. Knowledge Categories with subset domains contain formulas on the same concepts.

Definition 4.4. Two Knowledge Categories contain formulas on the same knowledge concept iff the domains of either Knowledge Category is a subset of the other, i.e. $\mathcal{D}_{c_1} \subset \mathcal{D}_{c_2}$ or vice versa. $\square$

There are three possibilities for comparing Knowledge Categories by their domains. The set of domains can either be:

- $D_1 \subseteq D_2$, one set fully overlaps with the other set, i.e. one set is a subset, superset of the other or they are equal. It is irrelevant which of the three options occurs.
- $D_1 \cap D_2 \neq \emptyset$, there is some overlap between the sets but also some differences, i.e. the domain sets intersect.
- $D_1 \cap D_2 \to \emptyset$, there is no overlap between the sets, i.e. the domain sets are disjoint.

For the disjoint domain sets case, the two Knowledge Categories are independent of each other and share no common knowledge. In other words, none of the formulas in the Knowledge Category relate to knowledge common to both categories. Example 3 describes this situation.

Example 3.

$$
\begin{aligned}
D_1 &= \{\text{object}, \text{size}, \text{shape}, \text{affordance}\} \\
D_2 &= \{\text{person}, \text{child}, \text{mother}, \text{father}, \text{parent}\} \\
D_1 &\cap D_2 \to \emptyset
\end{aligned}
$$

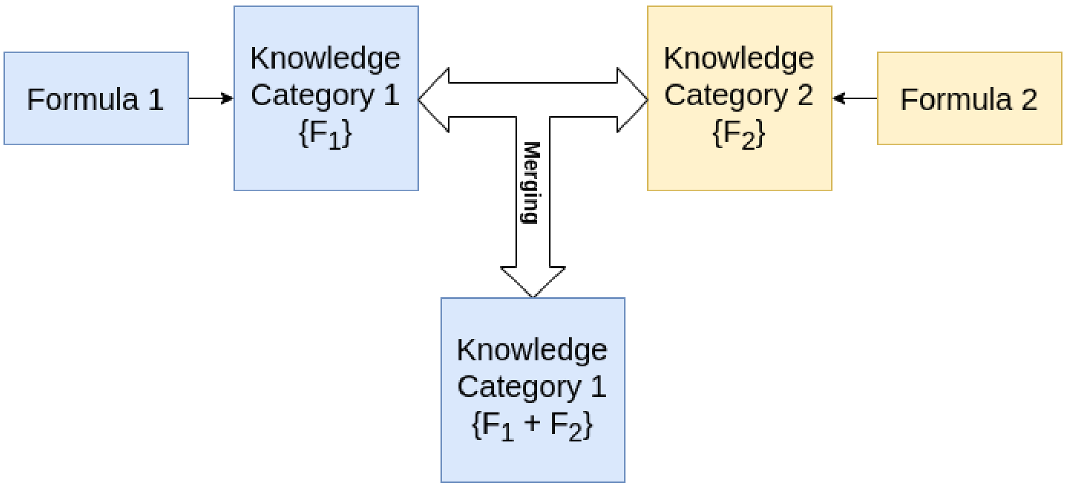
Knowledge updating

If the domain set of a Knowledge Category C_1 is a subset or is equal to the domain set of another Knowledge Category C_2, the two Knowledge Categories share common knowledge. All formulas in C_1 define rules over the same knowledge domains as C_2. The formulas and constants of the categories can be merged together into one Knowledge Category, i.e. $C_1(F_1, D_1, K_1) \cup C_2(F_2, D_2, K_2) \to C_2(F_1 \cup F_2, D_2, K_1 \cup K_2)$. Example 4 describes this situation.

Example 4.

$$
\begin{aligned}
D_1 &= \{\text{object}, \text{size}, \text{shape}, \text{affordance}\} \\
D_2 &= \{\text{object}, \text{affordance}\} \\
D_3 &= \{\text{object}, \text{size}, \text{shape}, \text{affordance}\} \\
D_2 &\subset D_1 \subseteq D_3 \\
D_1 &\cup D_2 \cup D_3 \to \{\text{object}, \text{size}, \text{shape}, \text{affordance}\}
\end{aligned}
$$

In the case that the domain sets of two Knowledge Categories intersect but neither is a subset of the other, then both categories describe different knowledge concepts using similar domains. These Knowledge Categories are not merged together as it is not clear how the difference set of the domains has any relation to either Knowledge Category. Example 5 describes three Knowledge Categories with intersecting domain sets with small differences.



Formula 1
Knowledge Category 1 {F1}
Knowledge Category 2 {F2}
Formula 2
Merging
Knowledge Category 1 {F1 + F2}

Figure 4-3: Two formulas each form their own independent Knowledge Categories. If the domain set of category 2 is a subset of that of category 1 then the categories are merged together into Knowledge Category 1. Knowledge Category 2 is subsumed in Knowledge Category 1 and is subsequently discarded.

Example 5.

$$
\begin{aligned}
D_1 &= \{\text{object}, \text{size}, \text{shape}, \text{affordance}\}\\
D_2 &= \{\text{object}, \text{mass}, \text{volume}\}\\
D_3 &= \{\text{object}, \text{size}, \text{colour}, \text{affordance}\}
\end{aligned}
$$

$$
\begin{aligned}
D_1 \cap D_1 &\rightarrow \{\text{object}\}\\
D_1 \cap D_3 &\rightarrow \{\text{object, size, affordance}\}
\end{aligned}
$$

As a consequence of definition 4.4, two categories on the same concept can be merged into one category by domain comparison as shown in figure 4-3. In practice, during the initialisation of a Knowledge List from an MLN, the first formula encountered in the MLN defines its own Knowledge Category with its set of domains. Each subsequent formula in the MLN is compared against this category. They are either put into this category if the domain sets match, or they form their own Knowledge Category for other formulas to compare with. Afterwards, each Knowledge Category in the Knowledge List is then compared to all other categories to merge categories on the same knowledge concepts together. After the merging step, the minimal set of Knowledge Categories is left over. During merging the information contained in one category is copied over to the other category. The category that information was copied from is discarded after merging as its information has been subsumed. The merging of Knowledge Categories is essentially a form of clustering. Newly learned formulas are put into an existing Knowledge Category or form their own new Knowledge Categories based on comparison of domain sets.

Knowledge updating strategies

The Knowledge Category merging operation consists of three operations: a domain set union, the constants union and merging the knowledge triplets from one category into the other.

Because the domains of two categories that should be merged are a sub/super set of each other, the union of the two domain sets results in the superset. In the constants union the constants of both categories are assigned to the corresponding domains in the Knowledge List.

During the knowledge triplet merging operation, each formula in the triplets is checked whether it is new to the category or it is the same as an existing formula in the category. If a formula is new, i.e. it is not the exactly same as an existing one, its triplet is simply added to the category. If a formula is the same, a conflict situation occurs.

Two triplets from two different categories each with the exact same formula cannot simply be merged together. The weights of the formulas are learned from different sets of evidence. Even when the weights are exactly the same, the evidence they were based on could differ and thus the knowledge encoded in the weights is different. One of the formula weights must be accepted to merge the two categories.

To solve the conflict it is assumed that formula weights in conflicts always represent different sets of evidence. This assumption allows the formulation of knowledge updating strategies to solve merging conflicts: the *CL-Naive*, *CL-Conservative* and *CL-Balanced* strategies. Other strategies are possible but are left for future work to explore.

The simplest possible strategy is to prefer any new knowledge over the old knowledge. This is represented in the CL-Naive strategy, described in algorithm 1. In this strategy conflicts in merging knowledge triplets (F, w, z) with the same formula $(F_1 = F_2)$ are always resolved by copying the new triplet over the old triplet, regardless of how much evidence either of the formula's weights was trained on.

The weights being overwritten after each learning step using this strategy does not mean all previously learned information is lost. During conversion of a Knowledge Category to an MLN, the formula weights are set to their respective current weights prior to learning. Thus each weight is adjusted based on the new evidence from the starting point of the weight from the previous learning step, i.e. $w_{t+1} = w_t + \Delta w$ where Δw is the newly learned weight difference. This way all learned information is carried over between learning steps.

Algorithm 1: CL-Naive knowledge updating strategy

```
Input: Knowledge triplets $(F_1, w_1, z_1)$ and $(F_2, w_2, z_2)$, where $F_1 = F_2$
Output: $(F_1, w_1, z_1)$
$w_1 \leftarrow w_2$
$z_1 \leftarrow z_2$
```

A converse strategy to simply accepting all new knowledge as better than the old is to only accept new knowledge as better if it is based on more evidence than the old knowledge. This is the CL-Conservative strategy, described in algorithm 2. In this strategy the evidence count portion of the knowledge triplets is important.

The evidence count of a formula represents how many ground atoms were seen during the training of the weight of the formula. The method for counting evidence is described in algorithm 4. Only the ground atoms corresponding to the predicates in the formula are counted as only they contribute to the weight during training. If the evidence count of the new knowledge is larger than that of the old then the new knowledge triplet overwrites the old. In subsequent learning steps the new evidence count is then the barrier to overcome. The old knowledge triplet is not overwritten if the evidence count for the new knowledge is equal or smaller than the old count.

In contrast to the CL-Naive strategy, the CL-Conservative strategy only updates weights of a formula if it is a new formula or there is more evidence for the weight of the formula than in the previous step. Because new evidence counts overwrite old evidence counts in this strategy, in the next step the new evidence is counted from zero. Another approach is to start counting the new evidence from the previous evidence count.

Algorithm 2: CL-Conservative knowledge updating strategy

```
Input: Knowledge triplets $(F_1, w_1, z_1)$ and $(F_2, w_2, z_2)$, where $F_1 = F_2$
Output: $(F_1, w_1, z_1)$
if $z_2 > z_1$ then
    $w_1 \leftarrow w_2$
    $z_1 \leftarrow z_2$
end
```

The CL-Balanced strategy, described in algorithm 3, applies this continuous counting approach. To solve same formula merging conflicts the balanced strategy takes the weighted average of the old and new formula weights based on their respective evidence counts. The resulting averaged formula weight contains partial information of each weight in proportion to their respective evidence. After calculating the formula weight average, the evidence counts of both knowledge triplets are summed. The sum of the evidence counts represents that the formula weights were combined and not overwritten.

If both evidence counts are zero the arithmetic average of the two weights is taken to prevent division by zero. This situation only occurs if two Knowledge Categories are merged with untrained formulas or formulas that have not yet seen any evidence. New evidence can contain only predicate declarations and formulas of yet unknown predicates and domains without any further supporting evidence. In this case Knowledge Categories, with knowledge triplets with weight zero and evidence count zero, are created for each predicate declaration. If any of these categories contain domain subsets of other newly created categories, they are subsequently merged together with the arithmetic average of the zero weights. The zero weight of the original formula is thus conserved. In normal operation this scenario is unlikely to happen.

As the number of learning steps grows, the evidence count of the CL-Balanced strategy will grow too. Except for the case where there is no new evidence for a formula, then the evidence count is zero and the corresponding weight will not contribute anything to the weighted average. Due to the evidence count growth, the influence of new evidence will have less impact on the merged formula weight if its evidence count remains similar to the previous step. In the CL-Balanced strategy the number of evidence must grow in proportion to the accumulated evidence of previous steps to prevent formula weight from converging.

Making a distinction in the Knowledge List for Knowledge Categories is important for determining which pieces of knowledge to update in the Knowledge List when new evidence is given. Only the categories that are relevant for the new knowledge have to be updated, saving computation resources and time. Furthermore, it ensures that any knowledge not relevant to the new evidence is not changed in any way.

Algorithm 3: CL-Balanced knowledge updating strategy

```
Input: Knowledge triplets $(F_1, w_1, z_1)$ and $(F_2, w_2, z_2)$, where $F_1 = F_2$
Output: $(F_1, w_1, z_1)$
if $z_1 = z_2 = 0$ then
    $w1 \leftarrow \frac{w_1+w_2}{2}$
end
else
    $w_1 \leftarrow \frac{z_1 w_1 + z_2 w_2}{z_1+z_2}$
    $z_1 \leftarrow z_1 + z_2$
end
```

4-2 Ingredients in practice

With the two main ingredients and knowledge updating strategies for MLN-CLA in hand, a Knowledge List L can be built from an MLN and an evidence database as shown in algorithm 9. Subsequently, L becomes a living knowledge base open for new knowledge to be incorporated. How this works is explained in the following sections.

4-2-1 Knowledge List creation

First the initial Knowledge List L must be created. From the input MLN its predicate declarations and formulas are parsed. The predicate declarations are put in the set of known predicates $\mathcal{P}$ of L. Each domain in the predicate declarations is added to the set of domains $\mathcal{D}$ of L. Then the first formula in the MLN is put into its own category. Each subsequent formula in the MLN is either merged into this category based on its domains or put into its own category. As a result, an MLN containing N formulas will results in a Knowledge List L containing 1 to N categories. The constants K of the evidence database are then added to their respective domains $D \in \mathcal{D}_L$. This process is shown in algorithm 7.

After these initial operations, the resulting Knowledge List L contains a set of predicate declarations, a domain to constants mapping and the unique Knowledge Categories that cluster the formulas in the input MLN. It is now ready to incorporate new knowledge.

4-2-2 Knowledge List operations

Several operations can be carried out on a Knowledge List. These are comparing new evidence to the List, splitting, merging with another Knowledge List, converting to MLN and merging Knowledge Categories.

New evidence comparison

Each piece of new evidence can be compared to a Knowledge List to determine whether that piece is known or unknown. If all pieces of new evidence are known or no unknown evidence is found, no special operations have to occur. In this case the weights of the formulas in the

Knowledge List have to be updated based on the new evidence. Here structure learning can be applied to find a more optimal structure of the formulas and thus also the Knowledge Categories in the Knowledge List. This is illustrated in figure 4-1 in the structure learning branch. In structure learning the weights of the formulas are optimised too.

If any new evidence is found, it must be incorporated into the Knowledge List. Depending on the symbol type of the evidence (predicate, variable or constant), the corresponding operations to incorporate the knowledge must be taken. In the case of an unknown predicate a matching Knowledge Category must be found based on the domains the predicate defines a relation over. If no match is found the predicate defines a relation over an as of yet unknown concept. Thus a new Knowledge Category is made for the predicate and its domains. This new Knowledge Category is added to the Knowledge List. Otherwise, the predicate is added to a matching Knowledge Category. In either case the predicate is converted to a knowledge triplet with a unit clause formula with weight zero and the evidence count.

In the case that the new evidence is a variable (only found in MLN formulas) or constant, the corresponding predicate declaration is fetched. From this declaration the domain of the variable or constant is determined. With the domain now known, the matching Knowledge Category can be obtained. This category is then flagged for further training on the evidence. In addition, the constant is added to the Knowledge List domain to constants mapping. Variables are not added to the Knowledge List as they only indicate that any constant of the domain can be put in that argument position in the formula. The assumption that all predicates and domains must be declared ensures that no constant or variable has no matching Knowledge Category.

This process is iterated over for each new piece of unknown evidence. Afterwards, the Knowledge Category merging operation is performed to ensure the set of Knowledge Categories is the minimal set.

Category merging

The Knowledge Category merging operation, as described in algorithm 8 and figure 4-3, can be applied in two situations: in Knowledge List initialisation or when two Knowledge Lists are merged. When initialising a Knowledge List categories are merged to ensure the minimal set of Knowledge Categories is kept. When merging two Knowledge Lists, as described in section 4-2-2, each Knowledge Category combination between the lists is compared against each other for matches to merge.

In the example of table 4-1, two KCs are compared against each other. Both contain the same formula with a different weight and evidence count. Example 6 demonstrates the application of the CL-Balanced strategy to solve the merging conflict.

	Knowledge Category 1	Knowledge Category 2
Index	22	8
Triplets	(Size(obj, Huge) ⇒ !Affordance(obj, Throw), 0.88, 122)	(Size(obj, Huge) ⇒ !Affordance(obj, Throw), -0.41, 47)
Domains	object, size, action	object, size, action

Table 4-1: Example of two Knowledge Categories (possibly of different Knowledge Lists) containing the same formula. The domains of Knowledge Category 2 are a subset of the domains of Knowledge Category 1. The weights and evidence counts of the formulas are different. The two categories must be merged together with a knowledge updating strategy.

Example 6.

$$\begin{aligned}
w_{C_1} &= \frac{z_1 w_1 + z_2 w_2}{z_1 + z_2} \\
&= \frac{122 \cdot 0.88 + 47 \cdot -0.41}{122 + 47} \\
&= 0.521 \\
z_1 &= z_1 + z_2 \\
&= 122 + 47 = 169 \\
\hline
\text{index} &= 22 \\
(F, w, z) &= \{(\text{Size(obj, Huge)} \Rightarrow \text{!Affordance(obj, Throw)}, 0.521, 169)\} \\
\mathcal{D} &= \{\text{object, size, action}\}
\end{aligned}$$

Splitting Knowledge List

A Knowledge List can be split up into independent MLNs. Each Knowledge Category can be converted back into an MLN as shown in figure 4-1. The MLN's predicate declarations, formulas and their weights are derived from the Knowledge Category. The formula weights act as a prior to for the weight learning algorithm. Because no two categories share formulas, they are independent when grounding the MLN. As a result, the independent MLNs can be changed without influencing the others. The same splitting strategy can be applied to the evidence database. On each of the MLN splits the weight and structure learning algorithms can be applied given the new evidence splits.

Each MLN split is trained on a corresponding evidence split, containing only those ground atoms of predicates declared in the MLN split. A major advantage of splitting up the Knowledge List is the ability to selectively update knowledge in the List. Depending on the evidence only a portion of the Knowledge List has to be updated, saving computation time and cost. In addition, the training of MLN splits can be parallelised to speed up the learning process.

Merging the splits

After training the MLN splits on the new evidence, they can be merged together. Because the MLNs are independent of each other, no special treatment of the formulas and weights is required to merge the MLN splits into one. The merged MLN then contains all formulas and predicate declarations of each of the splits. From this merged MLN a new Knowledge List is created following algorithm 7.

Knowledge Lists merging

To make sure of a fair and complete comparison with the original Knowledge List $L_{t=0}$, the merged MLN, formed from the MLN splits trained on the new evidence, is converted to a Knowledge List L_{new} first. Afterwards L_{new} is merged into $L_{t=0}$ to form L_{t+1} as described in algorithm 5. This process is similar to the Knowledge Category merging process.

The predicate declarations and domains both Lists are combined. Each KC of L_{new} is compared to every Knowledge Category in $L_{t=0}$ to find matches. Once a KC match is found based on their domains, the KC of L_{new} is merged into the matching KC of $L_{t=0}$. If any KC from L_{new} does not have a matching KC in $L_{t=0}$ it is simply added to the list of KCs of $L_{t=0}$.

After the end of this merging step the original Knowledge List $L_{t=0}$ has been updated with new knowledge to L_{t+1}. It is again ready to receive new evidence to incorporate in a never ending cycle.

4-3 What can be learned?

Cumulative learning with Markov Logic Networks can be applied in three ways:

- Learning new constants for known domains from new evidence
- Learning new predicates and formulas from new MLNs and new evidence
- Adjusting the knowledge base given new known evidence

New constants can be learned from new evidence. These constants extend existing domains in the knowledge base. For example, the domain `Size:(Tiny, Small)` can be extended with newly learned sizes `Medium, Large` and `Huge`. The weights of formulas can be adjusted to reflect the changes in the domains. If the formula templating ('+') symbol is used in the MLN formulas then new formulas can be learned for each new constant. The symbol causes Alchemy to learn a formula instance for each constant in the domain of the variable preceded by the + symbol. Extra constants can provide more context for formulas or better define the distribution of a domain.

New predicates can be learned from either a new MLN or from new evidence if an unknown predicate is declared in the new evidence. New formulas can only be learned from a new MLN as evidence databases in Alchemy only consist of ground atoms. In both cases the unknown predicates must be declared as part of the prior knowledge. The new predicates and formulas can define new relations for existing knowledge if the predicates range over any known domain. Those formulas are merged into existing Knowledge Categories. The existing knowledge can also be extended with new relations over previously unknown domains. These two cases can be combined; new predicates can define a relation between both known and unknown domains. Thus relations can be learned between previously independent knowledge domains.

New evidence can be additionally be used to update existing knowledge. If all predicates and constants in a new dataset are already incorporated in a Knowledge List, then the weights of the formulas in the Knowledge Categories can be adjusted by training on the new evidence. The formulas themselves can additionally be refined by performing structure learning on the new evidence.

In the case that a new MLN is fed into the cumulative learner consisting of only new formulas without any supporting ground atoms evidence, the new formulas are added to the Knowledge List, but no formula weight or structure learning is performed. Formulas without any support evidence (i.e. with an evidence count of zero) have zero weight and cannot be learned without supporting evidence. The formulas can, however, be learned in future learning steps and are therefore added to Knowledge List instead of being discarded.

4-4 Cumulative learning aspects of MLN-CLA

In chapter 1 the aspects of an ideal cumulative learner as described by Thórisson *et al.* were discussed. The MLN-CLA extension contains components that uniquely deal with each aspect of the ideal cumulative learner. In MLN-CLA the *memory and knowledge management* of a cumulative learner is represented by the Knowledge List and its Knowledge Categories, and the MLN structure learning of Alchemy.

The Knowledge List is the knowledge base of the cumulative learner. It contains all previously seen predicates, domains, constants and Knowledge Categories. The knowledge in the knowledge base is managed by the Knowledge Categories and the merging mechanics of MLN-CLA. The unique Knowledge Categories ensure that formulas and constants are clustered, and that new evidence is put in either the correct cluster or into a new one. The merging mechanics, including the knowledge updating strategies, manage the Knowledge List and update it for new evidence. Structure learning of MLNs is a form of memory and knowledge management as well. In structure learning formulas are changed. Incorrect or incomplete formulas are fixed and new formulas are added to an MLN under structure learning. Afterwards, the updated formulas are incorporated into the Knowledge List.

The *temporal capacity* of MLN-CLA is varied. The algorithm is passive and discrete, meaning that any time new evidence is fed into the algorithm it will initiate a new learning step, otherwise it will do nothing. The amount of new evidence in a step is not limited and can vary per step. However, a new learning step can only be initiated after the previous one. MLN-CLA is thus a sequential cumulative learner. The duration of a learning step is dependent on the amount of new evidence.

The *generality* of MLN-CLA is substantial. Due to their Markov properties MLNs can handle any number of knowledge domains. The structure learning algorithm enables MLN-CLA to find relations between previously independent knowledge domains. A limitation of MLN-CLA are the strict data input constraints. The algorithm only accepts input data in the Alchemy Markov Logic syntax. In addition, MLN-CLA only accepts evidence for which the predicates are declared, i.e. new evidence domains are labelled. These constraints can be lifted with the addition of an input translation step. For example, an LLM such as [78] can be used to translate natural language descriptions to first-order logic. Fine-tuning such an LLM on

Alchemy syntax allows it translate large image annotations datasets, text documents and more to fit the MLN-CLA input modality.

No human input is required for any aspect of MLN-CLA, except for the initial knowledge base. Of course, MLN-CLA is not an ideal cumulative learner. In terms of temporal capacity it suffers from its sequential learning steps. However, in terms of generality MLN-CLA shines as it can generalise knowledge from any domain.

4-5 Summary

In this chapter the concepts of a Knowledge List and Knowledge Category were introduced. The Knowledge List acts as a knowledge base for all known evidence and formulas. Knowledge Categories encapsulate a set of formulas that share a knowledge concept. These categories are unique based on the domains they encompass. In cumulative learning several operations can be performed on the Knowledge List, including splitting, merging and comparison. These operations enable the Knowledge List to correctly incorporate new evidence such as constants, predicates and formulas, into the knowledge base. The Knowledge List and Category together with the operations form MLN-CLA. In the next chapter the capabilities of MLN-CLA are put to the test.

Chapter 5

Experiments

To demonstrate the capabilities of MLN-CLA two experiments are carried out. As shown in the previous chapter, MLN-CLA is capable of cumulatively learning new constants, formulas or a combination of new constants and formulas. First the MLN-CLA ability of learning new constants from independent datasets is tested. Second the ability to learn new formulas from MLNs and corresponding evidence is tested. Together these experiments demonstrate all three cumulative learning capabilities of MLN-CLA. In addition each experiment tests the different knowledge updating strategies: CL-Naive, CL-Conservative and CL-Balanced. These are compared against two baseline batch learned MLNs.

First the software tools used to learn MLN weights and structure are discussed. Secondly, the dataset used in the experiments is showcased. Thirdly the experiments and their setup are explained. Finally, the measure used to evaluate the performance of the cumulatively learned knowledge bases is introduced.

5-1 MLN software tools

Few software tools for the creation, parameter learning, structure learning and inference of MLNs exist. One tool that incorporates all these features is ProbCog[1] by Technical University of Munich [79]. ProbCog has system integration with for instance Robot Operating System (ROS), but lacks advanced learning and inference algorithms such as lifted inference.

The Python software package pracMLN[2] is an easy to use way to learn weights of and performance inference with MLNs [80]. pracMLN is a fork of ProbCog and thus can also be integrated with ROS based robotics applications. pracMLN lacks support for structure learning of formulas in MLNs.

Alchemy[3] (version 2) by the University of Washington has been under development longer than ProbCog and pracMLN [81]. Alchemy 2 contains more advanced learning and inference

[1]`https://github.com/opcode81/ProbCog`, last updated January 2021.
[2]`https://www.pracmln.org/`, last updated May 2019.
[3]`https://alchemy.cs.washington.edu/`, last updated May 2020.

algorithms than ProbCog and pracMLN. It does have support for structure learning, albeit with a very slow algorithm [36]. We adopt the Alchemy systems and notation[4] for our experiments. Our MLN-CLA method is compatible with Alchemy.

5-2 Dataset

The experiments are conducted on the dataset described in the paper on knowledge based affordance inference from object properties by Zhu *et al.* [54]. Their dataset is based on the Stanford 40 Actions dataset by Yao *et al.* [82]. The Zhu *et al.* dataset contains 4 predicates describing objects: `IsA`, `HasVisualAttribute`, `HasWeight` and `HasSize` and one predicate ascribing an action affordance to an object: `HasAffordance`. Each predicate has two arguments, i.e. `HasWeight(object, weight)`: one labelling the object and the other describing the object property. In total the 5 predicates define relations over 6 domains: `object`, `affordance`, `category`[5], `visual_attribute`[6], `size`[7] and `weight`[8].

Only the portion of the dataset describing objects and their affordances is available[9]. The object properties `visual_attribute`, `size` and `weight` were derived from the paper. The `category` property of each object is taken from WordNet [83]. Any gaps in the dataset were filled in by hand based on a best guess. The dataset used in these experiments is thus an approximated reproduction of the original dataset.

The dataset is split in a training and a test set. The training set contains 40 objects and their corresponding object property ground atoms. The test set contains 22 objects not seen in the training set. The objects are similar but not the same as those in the training set. The 22 objects are described by their object property ground atoms. The object properties in the test set are all present in the training set. The `HasAffordance` predicate is the target query of the training and test sets. The ground truth for the query predicate in the training set is described in Table 1 and for the test set is described in Table 2. Table 5-1 gives an example for the evidence of an object in the dataset.

5-2-1 MLN

Next to data the MLN of Zhu *et al.* as described in MLN 1 is used in these experiments. The Zhu *et al.* represents a simple and compact model for highly correlated relations. It consists of 5 predicate declarations for the same 6 domains of the dataset. Four of the five formulas directly predict the affordance of an object via an implication. The only formula that does not directly predict the affordance of an object, `IsA(o,+c1)` $\Rightarrow$ `IsA(o,+c2)` is a

[4]See Appendix 8

[5]22 categories: animal, instrumentality, implement, device, container, tool, equipment, vehicle, machine, wheeled vehicle, vessel, electronic equipment, edge tool, handcart, seat, musical instrument, cooking utensil, computer, scientific instrument, knife, telephone, writing implement.

[6]33 visual attributes: boxy_2D, boxy_3D, clear, cloth, feather, furn_arm, furn_back, furn_leg, furn_seat, furry, glass, handlebars, head, horiz_cyl, label, leather, metal, pedal, plastic, pot, rein, round, saddle, screen, shiny, skin, tail, text, vegetation, vert_cyl, wheel, wood, wool.

[7]3 sizes: S1 (<10in), S2 (10-100in) and S3 >100inches.

[8]4 weights: W1 (<1 kg), W2 (1-10 kg, W3 (10-100 kg) and W4 (>100 kg)

[9]Available at https://ai.stanford.edu/~yukez/eccv2014.html.

Ground atom	Description
`IsA(Banjo, Musical_instrument)`	Object category is musical instrument
`IsA(Banjo, Device)`	Object category is also a device
`IsA(Banjo, Instrumentality)`	Object top level category is 'instrumentality'
`HasAffordance(Banjo, Grasp)`	Object can be grasped
`HasAffordance(Banjo, Lift)`	Object can be lifted
`HasAffordance(Banjo, Throw)`	Object can be thrown
`HasAffordance(Banjo, Push)`	Object can be grasped
`HasAffordance(Banjo, Fix)`	Object can be fixed
`HasAffordance(Banjo, Play)`	Object can be played
`HasVisualAttribute(Banjo, Boxy_3D)`	Object has a box shape
`HasVisualAttribute(Banjo, Wood)`	Object looks to be (partly) made of wood
`HasVisualAttribute(Banjo, Leather)`	Object looks to be (partly) made of leather
`HasWeight(Banjo, W2)`	Object has weight class W2 (1-10 kg)
`HasSize(Banjo, S2)`	Object has size class S2 (10-100in)

Table 5-1: Example of the evidence for the 'Banjo' object in the test dataset. Note that the object categories are taken from WordNet [83]. The `HasAffordance` evidence is part of the ground truth.

support formula. Zhu *et al.* note that this formula helps the model better predict affordances by establishing an ontological relation between objects.

The use of the special Alchemy + symbol in each formula signifies that, during training, a separate formula for each constant in the domain of that variable is learned. This allows the model to capture the differences between constants in the formula weights instead of learning a robust generalised weight for a single formula spanning the whole domain.

5-3 Evaluation

The ground truth of the Zhu *et al.* dataset is a binary 22×14 array of object-affordance pairs in the test set corresponding to the target query. Each entry in the ground truth array has a value of either 0 or 1. If an object-affordance pair does not occur in the test set, the corresponding value in the array is 0. Conversely, if an object-affordance pair does occur in the test set, the corresponding value in the array is 1. The ground truth array is shown in Table 2.

The inference output of an MLN on the test data consists of a marginal probability for each ground atom instance of the query predicate. This output is parsed and put into a predictions array of the same shape as the ground truth array. In contrast to the ground truth, the predictions array consists of non-binary values between 0 and 1. The predictions can be interpreted as follows: a marginal probability of 1 means the MLN is certain of the ground atom being true, a marginal probability of 0.5 means the MLN is uncertain whether the ground atom is true or not. Finally, a marginal probability of 0 means the MLN is certain that the corresponding ground atom is false. Marginal probabilities larger than 0.5 indicate that the ground atom is likely to be true. Marginal probabilities smaller than 0.5 indicate that the ground atom is likely to be false.

In the cumulative learning setting a problem arises with the parsing of the inference output and filling out of the predictions array. Some constants have not yet been learned by the cumulative learner during test time. For these constants the MLN cannot predict a marginal probability. For example, if the affordance constant `Play`, part of the test data, has not been seen in the training data of any cumulative learning step then the resulting MLN does not know about the constant and does not output a prediction for it.

The solution to this problem is to initialise the predictions array with all marginal probabilities of 0.5, i.e. each prediction is at base fully uncertain or unknown. Subsequently the inference output is parsed and the 0.5 entries in the predictions array are overwritten with the actual predictions. Afterwards, if any entry in the predictions is not overwritten due to missing predictions as a consequence of unseen constants, then the prediction for that ground atom is unknown and thus has marginal probability of 0.5. Thus missing predictions has no unintended positive or negative influence on the performance.

In order to properly compare the performance of different models a standard performance measure must be adopted. We follow Richardson and Domingos by adopting the Area Under the Receiver Operating Characteristic (ROC) Curve (AUC) performance measure of MLN inference [12]. The AUC measure is well suited for evaluation of classification problems. The followings experiments are formulated as multi-class classification problems.

5-3-1 Evaluation measure

The AUC measure is commonly applied as a performance measure in classification tasks. The ROC curve or receiver operating characteristic curve, is the graph of the True Positive Rate (TPR) plotted against the False Positive Rate (FPR) for different classification thresholds. A high classification thresholds means fewer positive classifications, resulting in fewer true positives and false positives and vice versa. Figure 5-1 shows an example of an ROC curve. Equations 5-1 and 5-2 describe how to calculate the TPR and FPR from the MLN inference predictions. Here, TP stands for the number of true positives, FP for false positives, TN for true negatives and FN for false negatives.

$$TPR = \frac{TP}{TP + FN} \tag{5-1}$$

$$FPR = \frac{FP}{FP + TN} \tag{5-2}$$

The AUC score is an aggregation of the classification performance for all possible classification thresholds. AUC is scale-invariant, meaning that it measures how well predictions are ranked against each other instead of the absolute values of predictions. Furthermore, it is classification-threshold invariant, meaning that the chosen threshold has no influence on the AUC and the quality of the model's predictive power in terms of AUC.

In the experiments the weighted average of the scores for each class is calculated. The weighted setting is used to counteract class imbalance as some affordances occur more frequently than others and skew the data. The frequency of the ground truth data is show in Table 2. In addition, the 'one-vs-rest' setting is used to compute the AUC for each class. In this setting

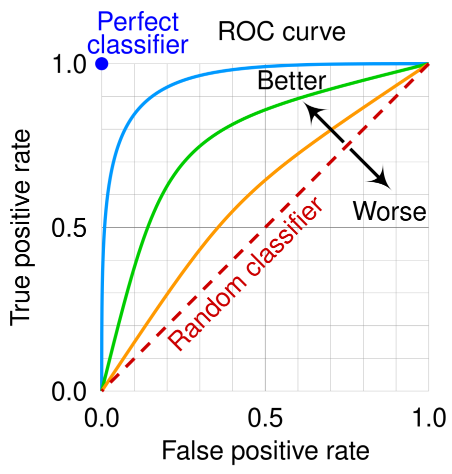


Figure 5-1: An example of an ROC curve: true positive rate against false positive rate for different classification thresholds. The diagonal line represents a random classifier that guesses correctly half the time. Better than random classifiers have a curve leaning towards the top left corner in the ROC space. The perfect classifier has 0% false positives and 100% true positives [84].

each class is compared against all others to calculate its score. The inference output of the MLN for a query predicate is multi-class. In our experiment setup the goal is to determine which affordances best fit an unknown object based on its properties. There are multiple answers correct for each object. The 'one-vs-rest' setting compares each affordance prediction against all others to determine if it is a better fit than the rest for the object. This setting is sensitive to class imbalance[10].

5-4 Cumulative learning of constants

In this experiment the learning of new constants is tested. The experiment is set up as a zero shot benchmark to compare a cumulative learner against a batch learner. The batch learned MLN is trained in one step on all training data. The experiment is repeated for each knowledge updating strategy.

The cumulative learner is trained in small steps on a new portion of the training data each time. Each learning step the cumulative learner is fed a new, independent database to learn from. Each database may contain evidence for constants that were not previously seen in other databases. In addition, each database contains evidence not present in any other database. The evidence is in the form of the ground atoms of predicates described in section 5-2. The Zhu *et al.* training set of 40 objects is split into 8 independent datasets of 5 objects and their descriptions.

The Zhu *et al.* MLN (see appendix C) consisting of 5 formulas is used as the initial knowledge base. Two batch learned MLNs are trained with the same initial MLN as the cumulative

[10]See the Sklearn documentation for more information `https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_auc_score.html`.

learner. The batch learned *et al.* MLNs are trained on the generative and discriminative setting respectively (see section 3-4-1).

Learning cycle

The experiment is started with an empty Knowledge List and an initial untrained MLN. The formula weights of the untrained MLN are all 0. In the first step before processing the new evidence of the step, a Knowledge List is automatically created based on the initial untrained MLN. Then the evidence for the step is incorporated into the Knowledge List. The updated Knowledge List contains formulas in the Knowledge Categories trained on the evidence. The next step builds further upon this Knowledge List with new evidence. In each step the formula weights of the previous step are used as initial weights for the next step. This process is repeated until all 8 datasets have been learned by the cumulative learner.

During cumulative learning, each MLN is trained with the generative setting on its corresponding dataset. The generative training setting is used as the query predicate for each Knowledge Category is not known *a priori* by the cumulative learner and thus the discriminative training setting cannot be used.

Inference and predictions

At the end of each step in cumulative learning, i.e. when the new evidence is fully incorporated in the Knowledge List, the current Knowledge List is saved. Then the Knowledge List is converted to an MLN for evaluation. This allows for evaluation of the learned knowledge on the test dataset (see appendix B) after each learning step.

Expected results

The expected result of learning new constants is an increase of performance over the learning steps for each knowledge updating method that approaches the performance of the batch learned MLNs. The batch learned MLNs should outperform the cumulative learner at 100% of the training data seen, because they can better learn the distribution of the training data domains than the cumulative learner can. In each step the cumulative learner only sees a portion of the distribution.

The knowledge strategy CL-Naive is expected to perform the worst as it causes formula weights to bounce around at each step. This strategy is the most sensitive to outliers. The CL-Conservative strategy is expected to perform better than CL-Naive because it only adopts formula weights based on stronger evidence, causing it to be more robust against outliers. Conversely, this can also have negative performance impact as slightly weaker, but still valuable evidence is disregarded. The CL-Balanced strategy is expected to perform the best of the three strategies as it values strong evidence, does not disregard any evidence and better takes previously learned weights into account.

Index	Formula
1	`IsA(obj,+category)` ⇒ `HasAffordance(obj,+affordance)`
2	`HasVisualAttribute(obj,+attribute)` ⇒ `HasAffordance(obj,+affordance)`
3	`HasWeight(obj,+weight)` ⇒ `HasAffordance(obj,+affordance)`
4	`HasSize(obj,+size)` ⇒ `HasAffordance(obj,+affordance)`
5	`IsA(obj,+category)` ⇒ `IsA(obj,+category)`

Table 5-2: Mapping of formulas to MLN split indices. Each formula is forms an independent MLN.

5-5 Cumulative learning of formulas

In this experiment the learning of new formulas is tested. This experiment focuses on the role of Knowledge Categories in clustering new evidence. New formulas can be about existing knowledge concepts or introduce new ones. If a formula is about new knowledge concepts, i.e. the formula predicates define relations over unknown domains, it will form a new Knowledge Category in the Knowledge List. Otherwise the formula is incorporated into an existing category.

Learning new formulas allows a cumulative learner to better categorise incoming data. Furthermore, learning new formulas with new predicates increases the number of questions that can be asked to the learner. New formulas can also help increase performance of certain queries if these formulas add more information about the query predicate. In the Zhu *et al.* MLN example, the formula `IsA(o,+c)` ⇒ `IsA(o,+c)` helps the model better distinguish objects by providing a class ontology to each object. This ontological relation is exploited in the formula `IsA(o,+c)` ⇒ `HasAffordance(o,+a)` to answer the query predicate.

The setup for the cumulative learning of formulas experiment is similar to that of learning constants. The main difference is that instead of feeding new evidence to the cumulative learner at each step, a new MLN containing one new formula is fed into the cumulative learner. The original MLN is split into five independent MLNs, each containing one formula and the corresponding predicate declarations. Table 5-2 shows the MLN split index and corresponding formulas.

Because the Alchemy system does not allow undeclared predicates to be used in training data, the training dataset is split up too. The training dataset is split into 5 databases, each containing only evidence for one formula. Thus each MLN split from table 5-2 has a corresponding database containing all ground atoms for each predicate in the associated formula. Each database contains evidence for all 40 objects in the training dataset.

Similar to the constants experiment, two MLNs are trained on all formulas and evidence in a batch learning setting. These are the original Zhu *et al.* MLNs trained on the generative and discriminative setting respectively.

Learning cycle

In the first learning step of the experiment a cumulative learning instance is created with an empty Knowledge List. The cumulative learner is fed a new MLN and its corresponding evidence to incorporate. At the end of the step the cumulative learner has incorporated the new formula into its Knowledge Categories and trained the weights of the formulas in each category.

A different new formula and corresponding evidence is fed to the cumulative learner until all five formulas from table 5-2 are learned. During the cumulative learning process, if the evidence for a new formula is already known to the cumulative learner, then structure learning is performed on the Knowledge Categories relevant to the new evidence. Structure learning updates the structure and weights of all formulas in a Knowledge Category based on the new evidence, instead of just updating the weights. The experiment is repeated for each different knowledge updating strategy.

In the cumulative learning process, each MLN is trained with the generative setting on its corresponding dataset. The generative training setting is used as the query predicate for each MLN is not known *a priori* by the cumulative learner and thus the discriminative training setting cannot be used.

Inference and predictions

At the end of each learning step the current Knowledge List is saved. Then the Knowledge List is converted to an MLN for evaluation. This allows for evaluation of the learned knowledge on the test dataset (see appendix B) after each learning step.

However, similar to the training dataset, the test dataset must be split up for each step, because the resulting MLN of each learning step may not contain formulas for one or more predicates in the test dataset. The result of each step is thus tested on a subset of the complete test dataset. As a result the performance at each learning step cannot be directly related to other steps.

Expected results

The expected result of learning new formulas and predicates is an increase of performance over the learning steps for each knowledge updating method. In each step, the cumulative learner incorporates more knowledge relevant to the test set. It learns more formulas and can better approximate the joint distribution of the test data, thus increasing performance with each learning step.

At the end of the learning cycle, the performance of each method is expected to be on par or better than the performance of the generative and discriminative batch learnedMLNs trained on 100% of the formulas and data. In cumulative learning the formula weights are initialised at the beginning of each step to the learned weights of the previous step. Based on the previous weights, the cumulative learner should improve its representation of the joint distribution if more information is gained after each step.

The expectation is that there are no differences in performance between the knowledge updating strategies CL-Naive, CL-Conservative and CL-Balanced for the formulas that form independent Knowledge Categories. From definition 4.4, the independent formulas are 1 to 4 (see table 5-2. The introduction of formula 5 is expected to result in a difference between the strategies as this formula belongs to the same Knowledge Category as formula 1. The performance difference between the strategies is expected to be similar to the learning of constants experiment.

5-6 Summary

To demonstrate the capabilities of the MLN-CLA algorithm two experiments are performed. The capabilities of learning constants and formulas from the dataset by Zhu *et al.* are tested. At the same time, the performance difference between the three knowledge updating strategies is evaluated. Several cumulative learning steps are performed in the experiments, each incorporating new knowledge whilst retaining the old. Two MLNs are trained in the batch learning setting with the generative and discriminative weight learning algorithms. To evaluate the performance of the resulting knowledge bases, they are converted to MLNs. These MLNs are used for inference on the test dataset of Zhu *et al.* [54] to predict marginal probabilities of the query predicate. The performance is measured with the AUC score for comparison. The results of these experiments are discussed in the next chapter.

Chapter 6

Results

In this chapter the results of experiments on the capabilities of MLN-CLA are shown. The results of each experiment are analysed in depth to demonstrate the differences between the knowledge updating strategies of MLN-CLA and to compare the differences between MLN-CLA and batch learning of MLNs.

6-1 Cumulative learning of new constants

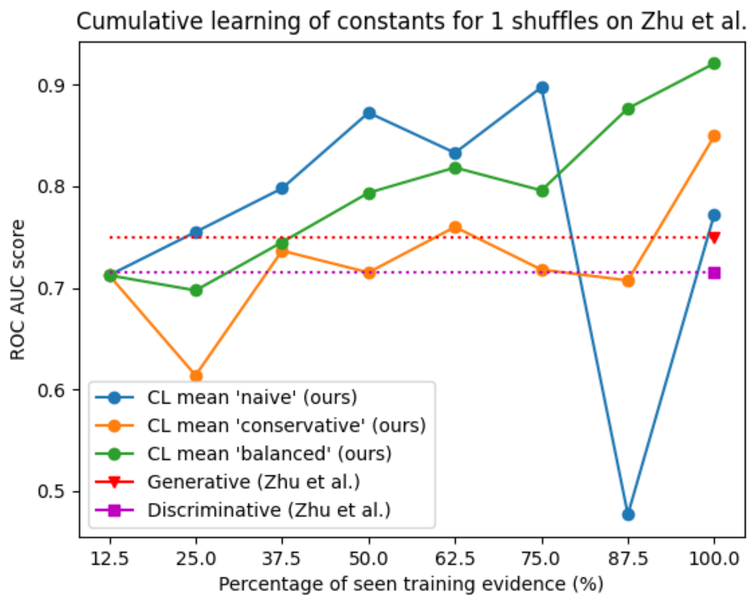
In this experiment the goal is to investigate the performance of MLN-CLA when learning new constants. In each learning step the cumulative learner is given a new, small batch of evidence that is a portion of all training data to incorporate into the knowledge base. The sum of all evidence batches amounts to 100% of all training data. MLN-CLA is compared against the two batch learned *generative* and *discriminative* MLNs that are trained on 100% of the training data. In addition, the performance of the three knowledge updating strategies is tested. A 100% of the test evidence is used to evaluate MLN-CLA on after each step.

The two comparisons are combined in one diagram shown in figure 6-1a. This figure shows the performances of different the different strategies against the batch learned MLNs on the test dataset after each learning step. Figure 6-1b shows the performance of the same methods but then for a different order of the evidence fed to the cumulative learner.

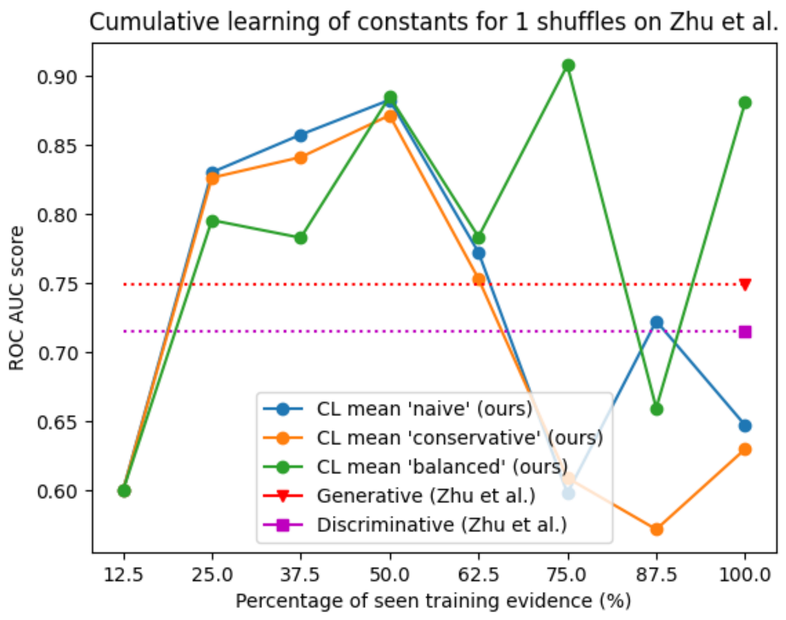
6-1-1 Analysis of new constants experiment

The results in figure 6-1 give rise to several questions:

1. What is the cause of the difference in performance of cumulative learning strategies over the learning steps?
   (a) Why do the three strategies all start with the same performance?
   (b) What causes the difference in performance in the final step between the three strategies?



(a) Evidence order: 4, 8, 6, 2, 1, 3, 5, 7.



(b) Evidence order: 8, 5, 1, 2, 7, 3, 4, 6.

Figure 6-1: ROC AUC performance of cumulative learning of new `object` constants for the MLN of Zhu *et al.* for the three knowledge updating strategies on the reproduced dataset of Zhu *et al.* [54]. A higher score is better. Each learning step a new (training) evidence database of 5 objects is fed into the cumulative learning algorithm. The MLN output for each step is evaluated on the test evidence database. After 8 steps, the cumulative learning algorithm has seen 100% of the training evidence. The performances of the batch learned MLNs for the two settings ‘generative’ and ‘discriminative’, are included for comparison. The difference between 6-1a and 6-1b is the order of evidence fed to the cumulative learner.

   (c) What is the cause of the large jumps in performance between some steps for each strategy?

2. What causes the difference in performance for different orders of evidence?
   (a) Does iterating over a number of different evidence orders result in a trend?
3. Why is the performance of the batch learned MLN with the *generative* setting better than that of the MLN trained with the the *discriminative* setting?
4. What causes the difference in performance between the cumulative learning methods and the batch learned methods?

Each question is answered in the following paragraphs. First, the inference output of the cumulative learning methods is analysed against the groundtruth. Subsequently, the marginal probabilities of the inference output are connected to formula weights for the test evaluation query predicate. Finally, the difference between the knowledge updating strategies is derived from the evidence counts for the weights of one formula for each learning step.

Inference output: marginal probability predictions

The first step in analysing the results of the cumulative learning of new constants is visualising the inference output of each learning step for each knowledge updating strategy. The figures 6-2 and 6-3 in show the marginal probability of each object-affordance pair in the test evidence. Figure 6-4 zooms in on the results of nine object-affordance pairs. These nine pairs showcase the sequential learning property of MLN-CLA, as well as the differences between the knowledge updating strategies.

In the figures 6-2 and 6-3 the affordances `Sit on`, `Ride`, `Feed`, `Play`, `Type on`, `Write with` and `Row` were not yet seen in the training data for some learning steps and thus have a marginal probability of 0.5 at those steps. The groundtruth for each pair can be found in appendix 8. In figure 6-4 the groundtruth of each object-affordance pair is indicated. In the figure the `Feed` affordance is first introduced in the fifth learning step. In the steps 1-4 the marginal probability for all object-'Feed' pairs is 0.5. After the introduction of the affordance constant, the cumulative learner is able to make predictions for this constant for all objects. This experiment result demonstrates the sequential temporal capacity of MLN-CLA.

Difficult and easy to predict cases

The difficult to predict affordances (columns) in figures 6-2 and 6-3 are `Grasp`, `Throw`, `Sit on`, `Fix` and `Pour from`. The `Grasp` affordance is difficult to predict, because there is no clear distinction in the training data between which objects can be grasped and which cannot. Of the 40 objects in the training evidence, 27 have the `Grasp` affordance. For the objects that do not have the `Grasp` affordance there is no defining property that makes clear that an object is not graspable. Intuitively, size is the defining property for an object having the `Grasp` affordance. However, in the training data the size ranges are not well defined, an axe has the same size class as a bicycle, resulting in size being a poor predictor for the `Grasp` affordance. The other difficult affordances have the same issues causing them to be hard to predict.

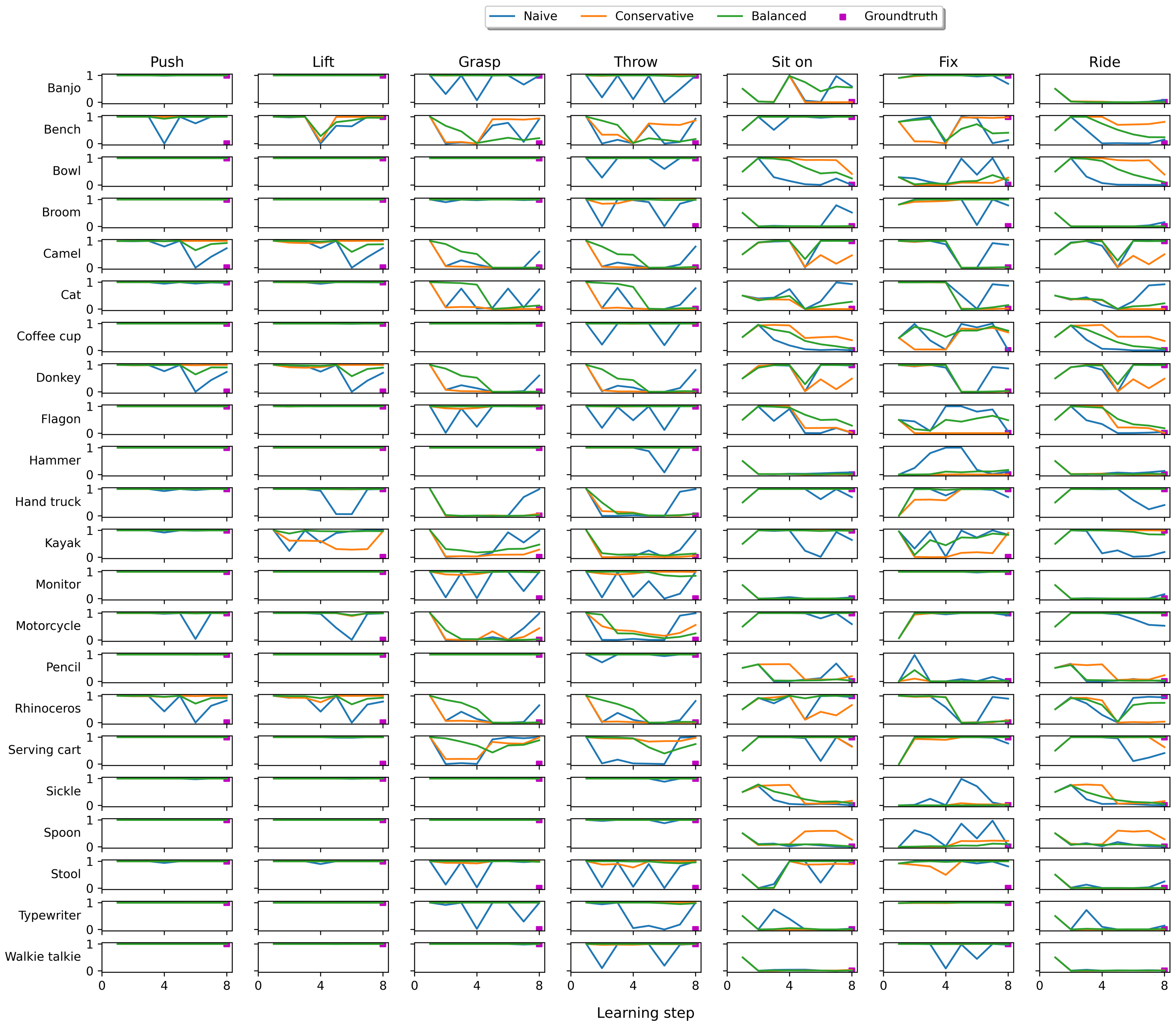


Figure 6-2: Learning new constants experiment: inference marginal probability outputs for the first 7 object-affordance pairs over the cumulative learning steps for each knowledge updating strategy. The affordances are sorted from left to right by descending occurrence frequency in the test evidence. Evidence database order: 4, 8, 6, 2, 1, 3, 5, 7 (same order as fig. 6-1b).

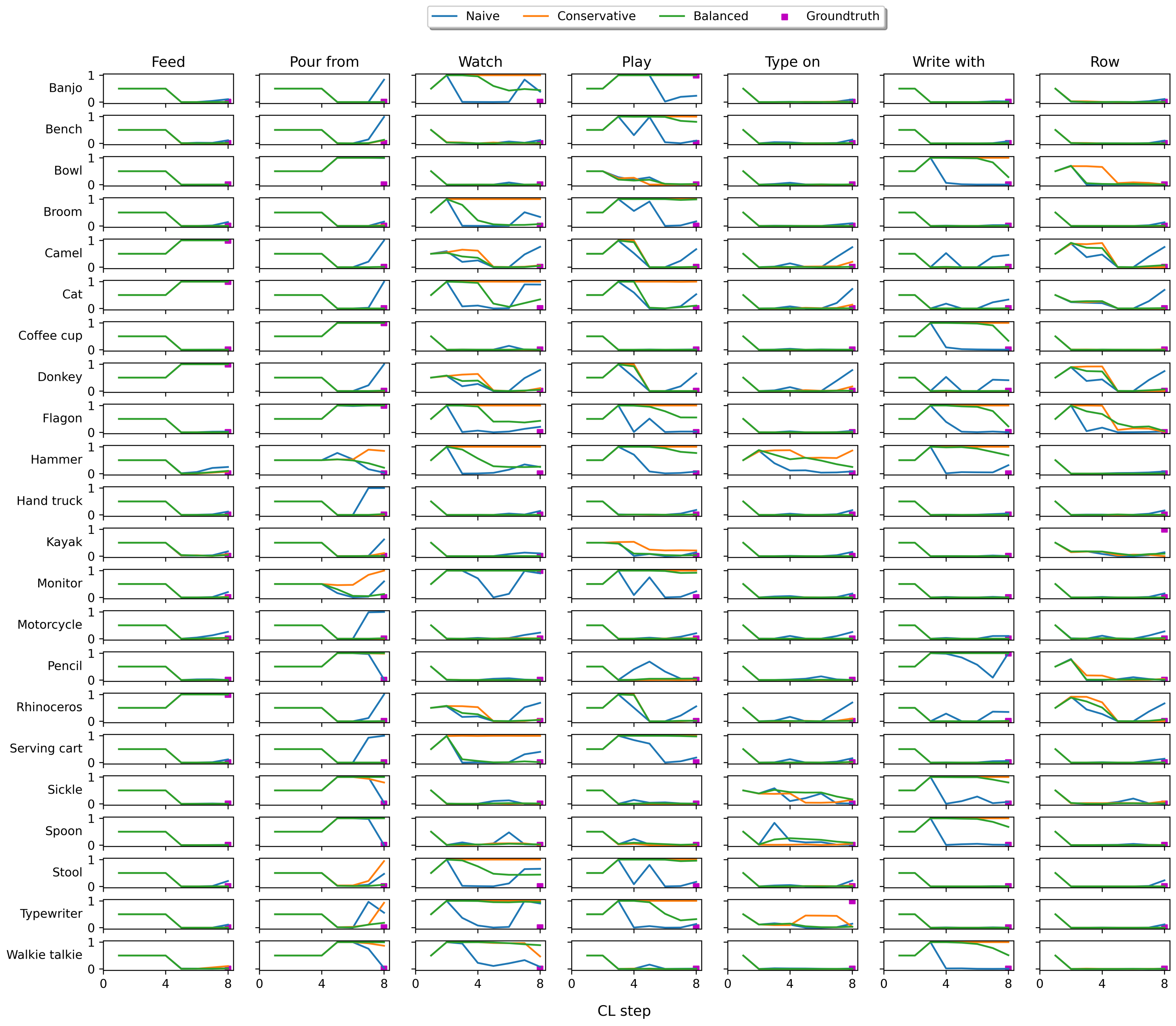


Figure 6-3: Learning new constants experiment: inference marginal probability outputs for the last 7 object-affordance pairs over the cumulative learning steps for each knowledge updating strategy. The affordances are sorted from left to right by descending occurrence frequency in the test evidence. Evidence database order: 4, 8, 6, 2, 1, 3, 5, 7 (same order as fig. 6-1b).

In contrast, the affordances `Push` and `Lift` are relatively easy to predict[1]. Here `Push` and `Lift` are interesting, because in the training evidence 37/40 objects have the `Push` affordance and 35/40 objects have the `Lift` affordance. The test evidence has similar frequencies for these affordances. Thus the MLN is certain in its predictions for these affordances.

The objects (rows) that are difficult to predict affordances for in figures 6-2 and 6-3 are `Camel`, `Cat`, `Donkey` and `Rhinoceros`. It is immediately clear that these objects are all animals, they each belong to the `Animal` category. Animals are a distinct category from the rest of the object categories in the training data. In addition, animals have visual attributes in common, such as `Skin`, `Tail` and `Head`, that occur infrequent among other objects. The training data only contains two examples of animals: a `Dog` in step 5 and `Horse` in step 6 (for this evidence order). In figure 6-4 the effect of the introduction of the `Animal` category is visible in step 5 of the `Camel-Grasp` pair.

The effect of learning about animals is immediately visible in the predictions for the animal objects. From step 4 to 5 for the first time evidence for an animal is given to the cumulative learner. As a result the `Push` and `Lift` affordance marginal probabilities drop from 1 to 0, where 0 is the groundtruth. The `Horse` object is large and heavy, whereas the `Cat` object is light and small. The affordance predictions of the `Cat` follow a similar but different trend than the other animals in the test set. The effect of learning of the `Dog` object in the final step is clearly visible too. The `Dog` is similar to a `Cat` in size and weight. As a result of learning the `Dog` object, the formula that relates object mass to an affordance has its weight lowered by CL-Naive and CL-Conservative. This causes predictions for the larger and heavier test animals to be negatively impacted. In contrast, the CL-Balanced strategy does not adjust the formula weights as drastically as the other strategies. This strategy adjusts the formula weight only slightly such that its predictions for the larger and heavier animals are more correct than the other strategies.

Some objects are relatively easy to predict, such as `Walkie talkie`, `Spoon`, `Bowl`, `Flagon` and `Sickle`. These objects have in common that they are small, lightweight and have a distinct visual attribute or category. In the MLNs these properties have large weights, causing the MLN to be confident in its predictions. In figure 6-4 the `Sickle` object has constant correct predictions by every strategy for the `Grasp` affordance. From the first learning step, no new evidence is introduced that influences the predictions of the `Sickle-Grasp` pair.

Knowledge updating strategies

The differences between the knowledge updating strategies in figures 6-2 and 6-3 are not visible from the data. Performance jumps are for each strategy are caused by two factors: the introduction of new evidence that is in contrast to previously seen evidence and the strategy for knowledge updating. The differences between the strategies become visible in the weights and evidence counts of formulas over the cumulative learning steps. Figure 6-5a shows the evolution of the weights of the formula `!HasVisualAttribute(obj,Shiny)` $\Rightarrow$ `HasAffordance(obj,Grasp)` and the effect of each strategy based on the evidence count at each step. Figure 6-5b shows the evolution of the weights of the formula `IsA(obj,Animal)` $\Rightarrow$ `HasAffordance(obj,Sit on)`.

[1]Leaving out affordances that were not seen in each cumulative learning step such as `Feed`.

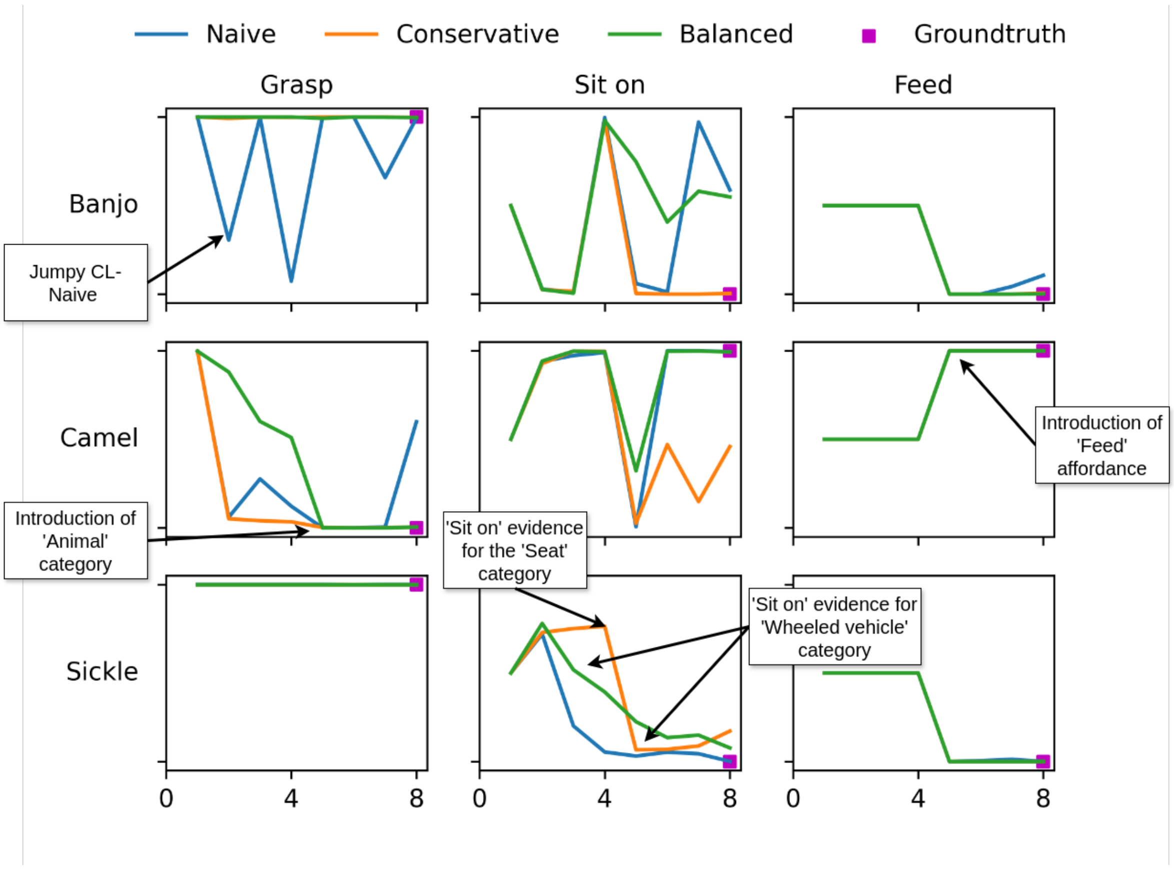


Figure 6-4: Marginal probability (y-axis) results of inference for each knowledge updating strategy of several object affordance pairs over the cumulative learning steps. Marginal probabilities range from 0 (certainly false) to 1 (certainly true). The binary groundtruth marginal probability for each pair is marked by the square symbol in the final step. Evidence database order: 8, 5, 1, 2, 7, 3, 4, 6 (same as fig. 6-1b).

Figure 6-5a and figure 6-5b demonstrate clearly the differences between the knowledge updating strategies and their resulting effects on formula weights. The formula weights differ between the strategies in each learning step, except for the first step where the initial weight is learned from the initial weight value 0. Each strategy reacts differently to the introduction of new evidence.

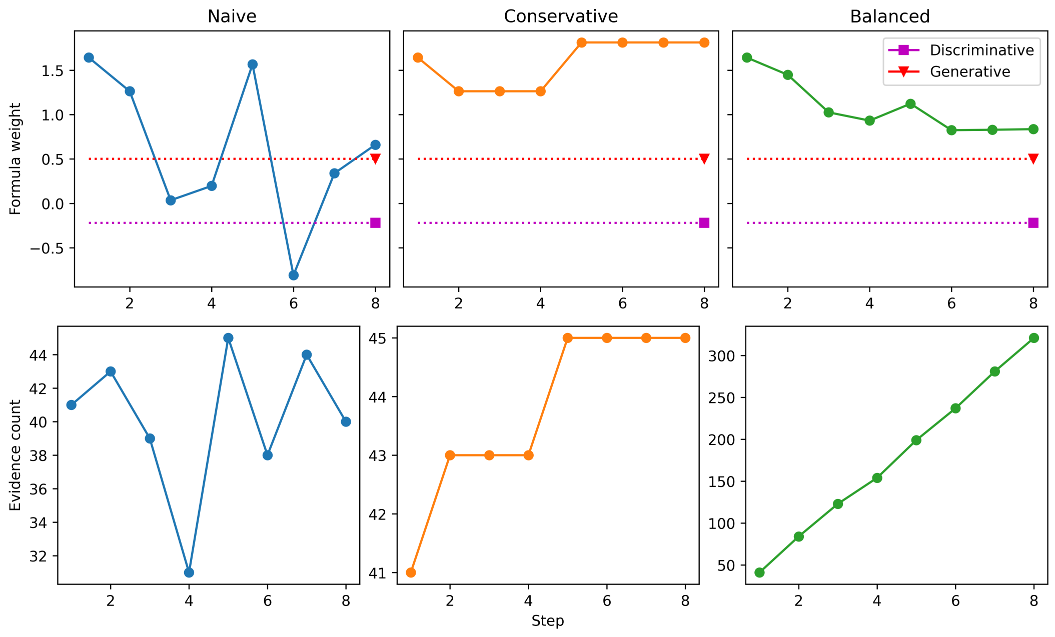
The CL-Naive strategy accepts all new weights, regardless of evidence counts. The formula weight for CL-Naive shows both large increases and decreases over the learning steps. As a results, the CL-Naive strategy shows significant jumps in its predictions for some object-affordance pairs such as `Banjo-Grasp` as indicated in figure 6-4. There seems to be no correlation between the rate of change of the evidence and the changes in the formula weights. A significant increase or decrease in evidence count does not necessarily correspond to a significant change in formula weight between two steps. CL-Naive demonstrates this in step 3 to 4 of figure 6-5a. In addition, the difference in evidence count between steps 4 and 5 is larger than between steps 5 and 6, whereas the difference in weight is larger between steps 5 and 6 than 4 and 5. It is not intuitive that weight changes can be larger for less evidence than in other steps. However, some evidence has more impact than other evidence. For instance, if new evidence suddenly makes a formula false instead of true, the new formula weight will decrease or even turn negative. The weight difference between learning steps is a measure of the impact of that evidence batch on formula weights.

The CL-Conservative strategy demonstrates the effect of not updating the formula weights for any change in evidence. This strategy only updates formula weights if the evidence count of the current step is larger than that of the previous step. In the middle plot of figure 6-5a the effect of this rule is demonstrated clearly between steps 1 and 2, and 4 and 5. The CL-Conservative strategy functions as a high-pass filter on the evidence count.

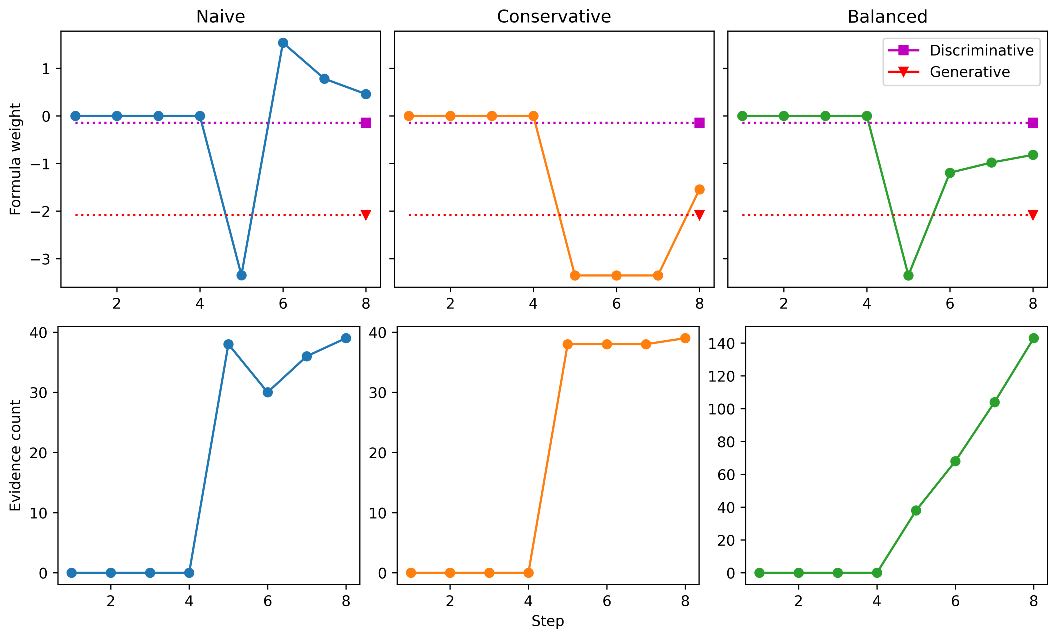
The CL-Balanced strategy directly adjusts formula weights based on the weighted average of the evidence count of the previous step and the current step. After each step, the evidence count is updated to the sum of the previous and current evidence counts. This strategy thus has growing counts. If the evidence count stays more or less equal over the learning steps, the effect of new evidence on the formula weight tapers off. In this experiment the evidence counts for each step differ at maximum by 10.

In figure 6-4 the CL-Balanced strategy clearly changes its predictions slower than CL-Naive and CL-Conservative. This delay is exemplified in the figure in the pairs `Camel-Grasp`, `Banjo-Sit on` and `Sickle-Sit on`. In each result, the balanced strategy follows the predictions of the other strategies, but with a less steep gradient.

Each updating strategy starts with the same formula weight and evidence count in step 1, because the knowledge updating strategies are not yet relevant in the first step. From step 1 to 2 the strategies do show their effect. From the evidence counts in figure 6-5a it is clear that a different order of the evidence will result in a different formula weight for each strategy. This is due to the formula weight of each learning step being initialised to the weight of the previous step before being trained on the new evidence, regardless of knowledge updating strategy. Between evidence orders the evidence in each step is (possibly) different.



(a) Formula: `HasVisualAttribute(obj,Shiny)` ⇒ `HasAffordance(obj,Grasp)`.



(b) Formula: `IsA(obj,Animal)` ⇒ `HasAffordance(obj,Sit on)`.The category constant `Animal` is first learned in step 5.

Figure 6-5: Tracking weights (top) and evidence counts (bottom) for two formulas. The three knowledge updating strategies CL-Naive (left), CL-Conservative (middle) and CL-Balanced (right) demonstrate different methods for updating the formula weights based on the evidence counts. All plots are based on the same data. The formula weights of the MLNs trained on 100% of the training data are included for reference. Evidence database order: 8, 5, 1, 2, 7, 3, 4, 6 (same as fig 6-1b).

Learning step	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
CL-Naive	0.059	0.103	0.108	0.127	0.133	0.142	0.147	<u>0.156</u>
CL-Conservative	0.059	0.069	0.081	0.091	0.099	0.105	0.108	0.106
CL-Balanced	0.059	0.069	0.080	0.097	0.100	0.104	0.106	0.104

Table 6-1: Standard deviations per step of each cumulative learning knowledge updating strategy. The largest overall standard deviation is underlined.

Evidence order

That evidence order influences the performance of the cumulative learner is evident from figures 6-1a, 6-1b and 6-5. The influence of evidence order on performance is be negated by averaging the experiment performance results of random evidence orderings. In figure 6-6 the mean performance over 300 shuffles of the 8 learning steps in the experiment is shown. Averaging over different evidence orderings better showcases the differences between the strategies.

In this figure, CL-Naive has the worst overall performance. At first CL-Naive outperforms CL-Conservative up to 50% of the training data, after which the strategy dramatically drops off in performance. The conservative strategy overtakes the naive strategy from 62.5% of the data seen and onwards, indicating that not updating weights can be an effective strategy. CL-Conservative has on par performance with the batch learned discriminative MLN.

In contrast to CL-Naive and CL-Conservative, CL-Balanced outperforms all other methods from the second step. This result suggests that the strategy of combining weights by ratio is effective in representing the joint probability of the data over time. Taking the weighted average of formula weights at each step based on evidence counts is effective in strengthening the belief of the cumulative learner in its knowledge. The influence of new evidence is weakened in comparison to CL-Naive and CL-Conservative. For those strategies new evidence seems to be detrimental for their performance.

The standard deviation for each strategy over the 300 shuffles grows larger after each learning step as shown in table 6-1. Intuitively, the cause of this growth is the increase in the number of possible seen evidence databases at each step. From the first step to the second, there are only 8 possible evidence orderings. From the second to the third step, there are already 64 possible orderings. Each combination of evidence orderings has a different resulting performance. The evidence ordering possibilities grow according to n^2_{steps}.

CL-Naive has the largest standard deviation owing to this strategy accepting all new knowledge without concern for the amount of evidence for that knowledge. This causes the naive approach to erratically change formula weights and 'jump to conclusions' that do not represent the overall data distribution well. The strategies CL-Conservative and CL-Balanced have similar standard deviations. The standard deviation curve of each strategy shows similarities to a radical function. Overall, the deviations are significant for each strategy, indicating that the large difference between the performance results of fig. 6-1a and 6-1b are to be expected.

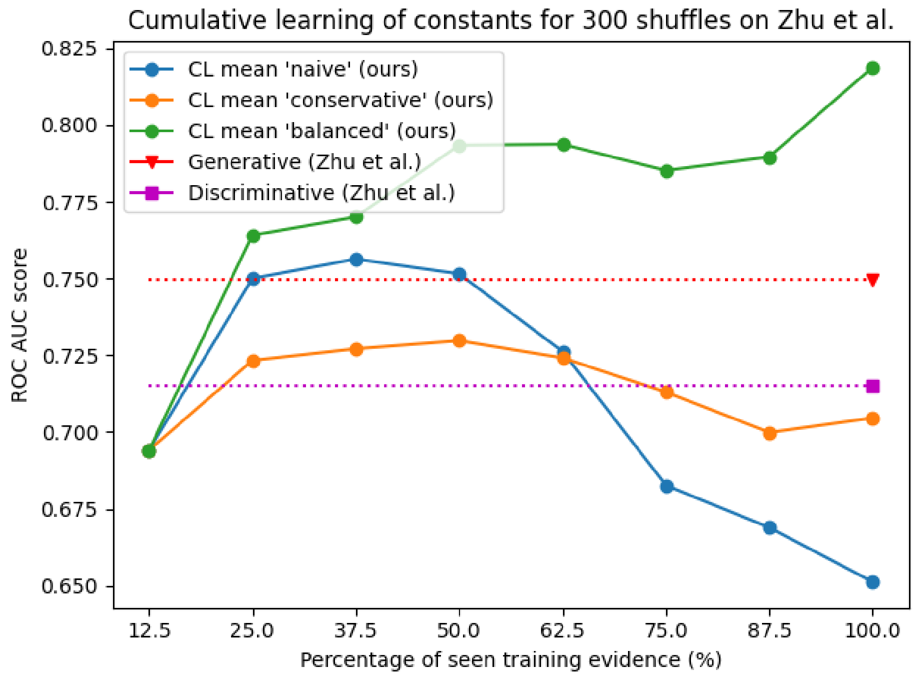


Figure 6-6: Area Under the Receiver Operating Characteristic (ROC) Curve (AUC) performance score of cumulative learning of new `object` constants for the MLN of Zhu *et al.* [54] for the three modes of knowledge merging. Each learning step a new (training) evidence database is fed into the cumulative learning algorithm. The resulting MLN is evaluated on the test evidence database. The order of databases is shuffled 300 times. Each graph represents the average performance over the shuffles. After 8 steps, the cumulative learning algorithm has seen 100% of the training evidence. The test performance of the MLNs trained once on all data, generative and discriminative batch learned MLN, is included for comparison.

Cumulative learning versus batch learning

The MLNs that were *generatively* and *discriminatively* trained on 100% of the training data outperform both the CL-Naive and CL-Conservative methods. CL-Conservative seems to approach the batch learned Markov Logic Network (MLN)s performance. The initial weights after the first step for each method already approach the batch learned performance. The CL-Conservative strategy updates formula weights the least during cumulative learning. Thus the conservative approach will not deviate much from the initial performance if there is no significant number of evidence to affect formula weights in another direction.

CL-Balanced outperforms the batch learned models from 25% and onwards of the training data seen already. The balanced approach is able to leverage the maximisation of the likelihood of the data during training, and the adjustment of formula weights after training based on evidence counts. The latter is beneficial as it tends to converge formula weights to a more optimal value than the initial formula weight given the fact that the training data in each step is similar in this experiment. If the data in each learning step were more diverse, i.e. each step containing a varying number of objects instead of a constant five objects, the performance impact of the balanced approach should be different. More investigation is required to determine whether this performance boost of the CL-Balanced strategy generalises to other databases and batch sizes than the dataset by Zhu *et al.* [54] and the batch sizes used in this experiment.

The *generative* training setting performs better on the Zhu *et al.* [54] than the *discriminative* training setting. Singla and Domingos showed that *discriminative* weight learning leverages prior knowledge of the query predicate to better approximate the data joint distribution. However, discriminative weight learning is only better than generative weight learning for MLNs with complex formulas such that inference paths are longer [35]. This result shows that for datasets with a small number of predicates optimising the joint likelihood of all predicates instead of the conditional likelihood of the query predicates given the evidence predicates results in better performance. MLN-CLA employs *generative* training of MLNs in the cumulative learning process as *a priori* the query predicate is not known.

6-1-2 Experiment conclusion

Learning new constants and incorporating them into the knowledge base extends the domains of knowledge of a cumulative learner. This experiment proves that MLN-CLA can learn new constants cumulatively from evidence. Within MLN-CLA the three knowledge updating strategies give different performance results due to their different rules for formula weight overwriting. In the first learning step there is not yet any difference between the strategies, because there are no merging conflicts yet. In each learning step the MLN formula weights are initialised to the weights of the previous step for each strategy. The differences between the strategies thus accumulate over the learning step, culminating in the performance difference in the final step. The impact that some evidence batches have on formula weights cause large jumps in performance on a step by step basis.

The sequential processing nature of MLN-CLA implies that the order that the evidence batches are seen in has a large impact on performance. Some evidence batches have more impact than others. Averaging over 300 different shuffles of evidence batch order permutations results in performance trends. On average, the CL-Balanced strategy outperforms all other methods when learning new constants. The CL-Naive and CL-Conservative strategy do not perform better than the baseline batch learned MLNs. The CL-Naive strategy suffers from discarding too much information by overwriting all old weights with newly learned weights. The CL-Conservative strategy performance tends to neither improve nor worsen when learning new constants.

The performance difference between the batch learned MLNs and the MLN-CLA strategies is caused by prior weights. The batch learned MLNs start learning weights from a prior weight of 0. In contrast, the MLN-CLA strategies only start learning weights from 0 in the first step. Afterwards, they use the weight of the previous step to start learning from in each step. Only for the CL-Balanced strategy does this approach result in better than baseline performance when learning new constants. This is unexpected as normally incremental learning should approach the performance of batch learning methods. CL-Balanced adjusts learned weights in each step by averaging the previous weight with the newly learned one. This results in a smaller difference in weights between steps. The accumulated difference results in weights that better fit the test evidence distribution.

6-2 Cumulative learning of new formulas

In the learning of new formulas experiment, the goal is to investigate the ability of a cumulative learner to learn new formulas and corresponding evidence. In learning new formulas the Knowledge Categories play an important role. Depending on the domains of a new formula, it is either categorised into an existing Knowledge Category in the Knowledge List or a new Knowledge Category is created for the formula. In the case of a new formula belonging to an existing Knowledge Category, the formula is added to it. Otherwise the a new, independently trained Knowledge Category is created for the formula.

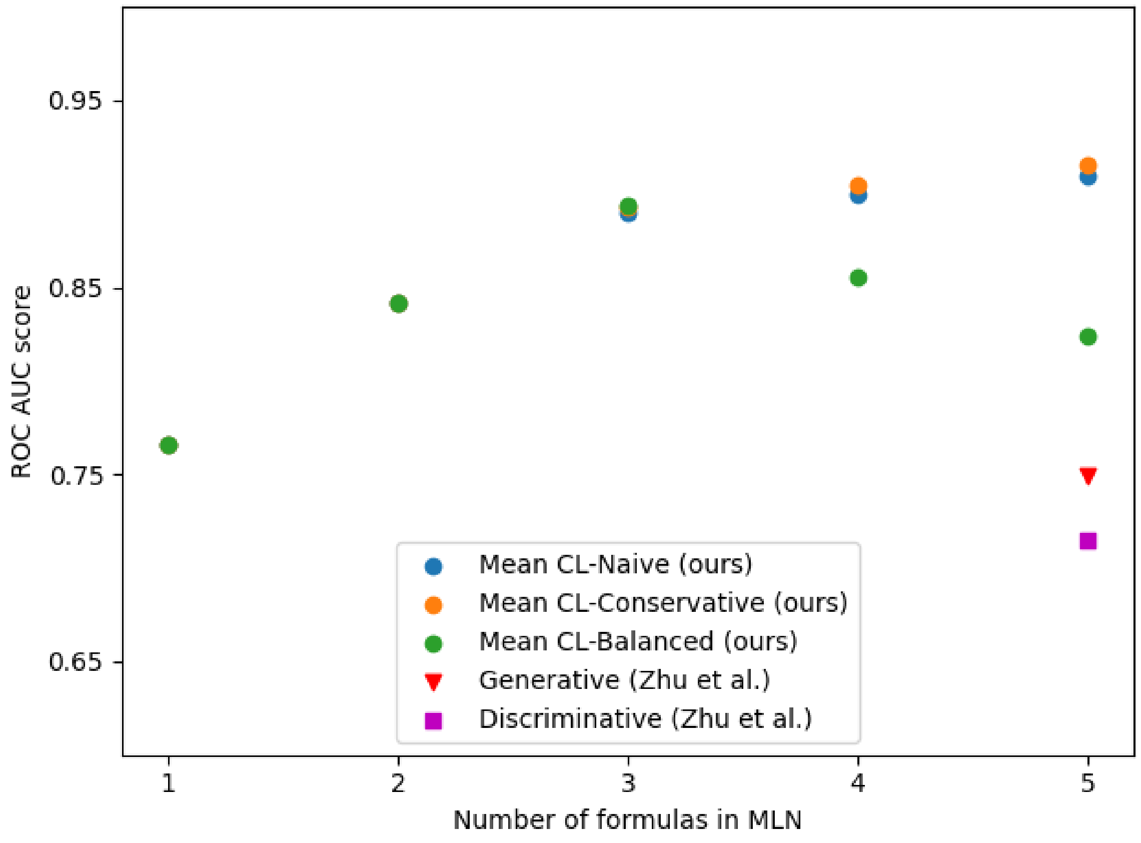


Figure 6-7: AUC performance score of cumulative learning of new formulas of the Zhu *et al.* [54] MLN. At each step a new formula with corresponding evidence is fed to the cumulative learner. After each learning step the resulting MLN is evaluated on the test evidence filtered for predicates seen up to the and including the current step. At each step step the test set is different. After five steps, the Knowledge List resulting from the cumulative learning algorithm has seen 100% of the formulas and evidence. The MLNs *discriminative* and *generative* trained with all five formulas on 100% of the training evidence batch learned are included for comparison.

The results of this experiment are shown in figure 6-7. The figure shows the mean evaluation results of the cumulatively learned MLN of each learning step for five iterations. At each step the MLN is evaluated on a test set containing only ground atoms for the predicates known to the cumulative learner to prevent inference on unknown predicates. In each step the amount of test evidence grows. As a result, the performance scores of each step are not

Step	**1**	**2**	**3**	**4**	**5**
Iteration 1	2	1	3	4	5*
Iteration 2	3	4	1	5*	2
Iteration 3	1	2	5*	3	4
<u>Iteration 4</u>	5	3	4	2	1*
Iteration 5	4	5	2	1*	3

Table 6-2: The formulas learned at each cumulative learning step for the five iterations. Each column and row is unique. Table 5-2 maps formula indices to the corresponding formula. The underlined iteration learns a formula in the first step that does not contain the query predicate `HasAffordance` and thus cannot be evaluated on the test evidence. Asterisks indicate when structure learning is performed.

directly comparable to the scores of the other steps.

To fairly compare each knowledge updating strategy against each other when performing this experiment it is imperative that the order of the formulas over the learning steps is unique for each iteration and the same for every strategy. The table 6-2 shows the order of the learned formulas for five iterations. Each strategy is trained on the same formula orderings from this table. Each row and column in the table is unique to ensure that taking the mean of the results of each step (column) represents the performance of that step for all five different formulas. Each row is unique to ensure that the cumulative learner is not fed any duplicate formulas and has seen all five formulas in total after the final learning step.

In each iteration structure learning is performed once. In MLN-CLA structure learning is performed when all new evidence is known and part of a Knowledge Category as explained in chapter 4. Only Knowledge Categories relevant to the new evidence are used for structure learning. These are converted MLNs and subsequently merged together into one large MLN. Structure learning on the new evidence is applied on that large MLN.

When structure learning is done depends on in which step the formulas `IsA` $\Rightarrow$ `HasAffordance` (1) and `IsA` $\Rightarrow$ `IsA` (5), were learned. Structure learning is performed after both formula 1 and 5 are learned as indicated by the asterisks in table 6-2. The weights and evidence counts of the formula `IsA(obj,Instrumentality)`[2] => `IsA(obj,Implement)`[3] are tracked in figure 6-8.

Table 6-3 shows the performance of each knowledge updating strategy after learning a new formula over the five iterations. The mean of each strategy at each step is plotted in Figure 6-7. The standard deviations at each step for each strategy are shown in table 6-4.

6-2-1 Analysis of new formulas experiment

In figure 6-7 the three knowledge updating strategies all outperform the two batch learned MLNs after having learned the final formula. The difference between the cumulatively learned methods and the batch learned methods can be found in the formula weights of each method.

[2]Constant `Instrumentality` refers to "an artifact that is instrumental in accomplishing some end" according to Wordnet [83]

[3]Constant `Implement` refers to "instrumentation (a piece of equipment or tool) used to effect an end" [83]

10^{-2}	N	C	B	N	C	B	N	C	B	N	C	B	N	C	B
Step		**1**			**2**			**3**			**4**			**5**	
Iter. 1	88	88	88	94	94	94	92	92	92	90	90	90	93	93	91
Iter. 2	80	80	80	81	81	81	93	93	93	92	91	88	93	93	91
Iter. 3	93	93	93	94	94	94	90	91	91	94	93	92	93	93	91
Iter. 4	<u>50</u>	<u>50</u>	<u>50</u>	80	80	80	81	81	81	89	89	89	90	91	*71*
Iter. 5	72	72	72	72	72	72	89	89	89	85	90	*69*	87	89	*68*
Mean	**77**	**77**	**77**	**84**	**84**	**84**	**89**	**89**	**89**	**90**	**90**	**86**	**91**	**92**	**82**

Table 6-3: The AUC score (10^{-2}) of each step of all iterations in the cumulative learning of formulas experiment for each strategy CL-Naive (N), CL-Conservative (C) and CL-Balanced (B). The underlined score indicates the step in which the first formula learned does not contain the query predicate. The scores in italics show outliers. Batch learned MLN AUC scores: *generative* 0.750, *discriminative* 0.715.

Learning step	**1**	**2**	**3**	**4**	**5**
CL-Naive	<u>0.168</u>	0.0.94	0.048	0.034	0.027
CL-Conservative	<u>0.168</u>	0.094	0.049	0.016	**0.014**
CL-Balanced	<u>0.168</u>	0.094	0.049	0.094	0.118

Table 6-4: Standard deviations per step of each cumulative learning knowledge updating strategy. The largest overall standard deviation is underlined. Bold indicates the smallest standard deviation.

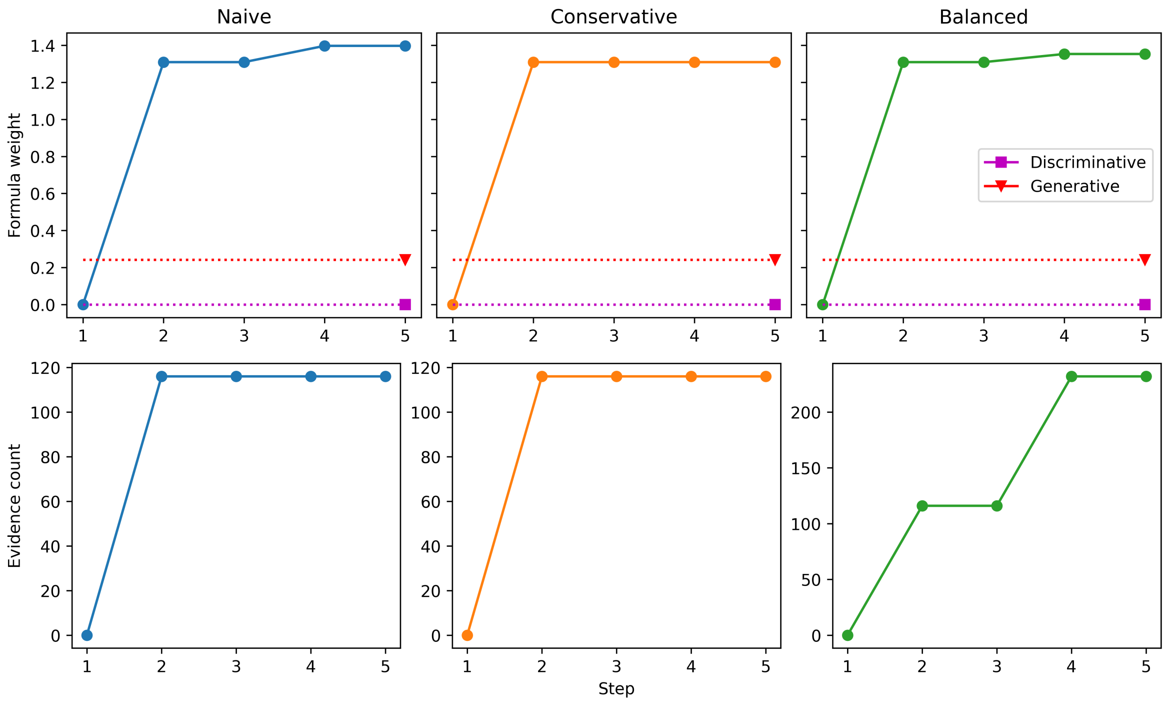


Figure 6-8: Formula weights and evidence counts for the formula `IsA(obj,Instrumentality) => IsA(obj,Implement)`. These are based on iteration 5 of the formula sequences. The formula is first learned in step 2. In step 4 another formula with the `IsA` predicate is learned.

The formula weights of the cumulatively learned methods are overall larger in both the positive and negative weight direction than those of the batch learned methods. This observation explains the performance difference between the batch learned and cumulatively learned methods. The cumulatively learned methods perform better because the weights of their final MLN models are relatively stronger than those of the batch learned methods.

We hypothesise that the stronger weights are caused by the cumulative learner having to optimise fewer weights in each learning step than the batch learned MLNs have to. This could cause the batch learned MLNs to 'soften' their weights in comparison to cumulatively learned MLNs. More experiments on larger datasets are required to test this hypotheses.

The cumulative learning process thus has a positive impact on the final performance of the learned MLN. The question is how the formula weights develop over the learning steps for each strategy. Quantitative analysis of the formula weights after each step of the fifth iteration is shown in 6-8. In the fifth sequence of formulas, the `IsA` $\Rightarrow$ `IsA` formula is learned in the second step. After incorporating this formula, each strategy learned the same initial weight. In the fourth learning step the formula $IsA \Rightarrow HasAffordance$ is learned. The evidence counts for both steps are the same as in each step the evidence for all 40 objects is given. This causes CL-Conservative to not update the formula weight. CL-Naive and CL-Balanced do update the formula weight.

In the same figure the formula weight for the batch learned MLNs are plotted. The discriminative MLN assigned a weight of 0 to all `IsA` $\Rightarrow$ `IsA` formulas, because the formula does not contain the query predicate `HasAffordance`. Adding the predicate `IsA` to the query predicates is possible but not fair for comparison with the generative MLN. The generative MLN did learn a weight for the formula, but only a relatively small weight in comparison to the cumulative learning strategies. These relatively weaker weights cause the generative MLN to make worse predictions than the cumulative learning strategies.

The effect of learning a formula without a query predicate in iteration 4 has no significant impact on the performance in the next steps for each strategy. The reason for this is that the formula weights learned in this first step are carried over as initial weights to the next step. Meanwhile, the resulting MLN from the first step is not evaluated because of the lacking query predicate, causing a dip in performance. In the second step of this iteration, the query predicate is learned for the first time. Now the learned MLN can be queried for inference with the query predicate without loss of performance as illustrated in table 6-3.

The standard deviation (table 6-4) decreases in each learning step for each strategy. This is the reverse of the results in the previous experiment. The increase in number of learned formulas causes the cumulative learner to better predict the query predicate in the test evidence in each step. Each step the test evidence grows too. The extra test evidence helps the MLN to more accurately predict the query predicate during inference.

Knowledge updating strategies

The three strategies have near identical performance for the first three learning steps before CL-Balanced tapers off in performance. Each strategy learns the same new formula in a step. The learned formulas are all categorised in their own domain, except for formula 5 and 1. Formula 5 and 1 belong to the same category as the domains of formula 5 – $\{$`object`, `category`$\}$

– is a subset of the domains of formula 1: $\{\texttt{object}, \texttt{category}, \texttt{affordance}\}$. In each step the learned formula does not conflict with previously learned formulas and is simply added to the Knowledge List. As a result there is no difference between the strategies as no formula merging conflicts have arisen.

In this experiment, the balanced strategy of taking the weighted average of formula weights to solve knowledge merging conflicts has a negative performance effect. The performance of CL-Balanced is significantly worse in iteration 4 step 5, and iteration 5 step 4 than the other two strategies as shown in table 6-3. These two steps correspond to structure learning steps after learning the formula `IsA` $\Rightarrow$ `HasAffordance` as shown in table 6-2. The same drop in performance score does not occur after structure learning, having learned the `IsA` $\Rightarrow$ `IsA` formula, in iterations 1 to 3.

Manual inspection of the learned formula weights of CL-Balanced and CL-Naive for iteration 4 resulted in significant differences in formula weights found. The learned formula weights for each formula in CL-Balanced are different from every formula in CL-Naive, except for the newly learned `IsA` $\Rightarrow$ `HasAffordance` formula. For the latter no differences in formula weights were observed between the two strategies. For iteration 1 only the formulas containing the query predicate are different in weight between the two strategies. The main difference between the weights is caused by structure learning. For iterations 1-3 structure learning does not change the weights of the `IsA` $\Rightarrow$ `IsA` formulas much. For iterations 4 and 5 structure learning does change the weight of these formulas. This causes the difference in performance. The effect of changing the weights of the `IsA` $\Rightarrow$ `IsA` by structure learning is only expressed in the performance of the balanced strategy. The averaging of the structure learned weights with the weights from the previous step has a significant negative impact on the performance for the CL-Balanced strategy. CL-Naive and CL-Conservative benefit from their non-interference with the weights. CL-Balanced has the largest standard deviation due to this effect of structure learning as shown in table 6-4.

The strategies CL-Naive and CL-Conservative perform the best, in contrast to the constants learning experiment. The conservative strategy performs slightly better of the two. As demonstrated in figure 6-8, the conservative strategy does not update any weights for all steps in this experiment. Due to the experiment setup, the evidence counts for each formula are the same in every step. The effect of structure learning on the performance is thus negative as CL-Conservative does not accept the structure learned weights where CL-Naive does.

6-2-2 Experiment conclusion

This experiment demonstrates the ability of MLN-CLA to learn new formulas and corresponding evidence. The learning of new formulas is essential to better model the relations between knowledge domains. In contrast to the previous experiment, all three knowledge strategies of MLN-CLA perform better on the test set than the batch learned MLNs. Structure learning occurs during the learning of new formulas. This has an insignificant performance impact for CL-Naive and CL-Conservative, but has a significant impact on the performance of CL-Balanced. The direct altering of formula weights that was beneficial in the learning of new constants is now a drawback, causing a deterioration in performance after structure learning. The CL-Naive and CL-Conservative strategies have near identical performance due to the experiment setup.

In this experiment the impact of evidence order is again significant. Each different permutation of formula order shows a different performance for each strategy. Although the steps are different between the iterations, the final results are identical. Except for one situation where a specific formula is learned after another. This causes structure learning to trigger but the result of structure learning is worse than the MLN used as input. This require more experiments on other datasets to explore the root cause of this phenomenon.

6-3 Summary

In this chapter the results of the two experiments of learning new constants and learning new formulas with MLN-CLA are analysed. The different updating strategies of MLN-CLA show significant differences in performance when learning new constants. The CL-Balanced strategy outperforms the batch learned MLNs. The order of the evidence in each learning step has a major impact on performance. This phenomenon is inherent to the sequential handling of new evidence of MLN-CLA. All knowledge updating strategies of MLN-CLA outperform the baseline batch learned methods when learning new formulas. In this setting the CL-Balanced strategy performs significantly worse than the CL-Naive and CL-Conservative strategies. The balanced strategy suffers from structure learning affecting some formula weights. In this experiment the order the formulas are learnt in impacts performance as well.

Chapter 7

Discussion

In this chapter the limitations and unexplored possibilities of MLN-CLA are discussed.

7-1 Evidence assumption

Automatic incorporation of new evidence in MLN-CLA is based on the assumption that all predicates are declared a priori in each database. This assumption is necessary for the algorithm to cluster knowledge. Without a concept of how knowledge elements are related to each a cumulative learner cannot place new knowledge into context [65].

That this assumption is feasible is proven by the fact that new predicates can be discovered and automatically declared through the use of predicate invention algorithms such as [85]. Predicate invention algorithms can discover new concepts from evidence. Another approach is to use LLMs such as LogicLLaMA [86] to translate natural language to FOL statements. With this approach large image annotation datasets such as MSCOCO [87] or VisualGenome [88] can be translated to FOL datasets suitable for cumulative learning with MLNs.

7-2 Memory and knowledge management

As part of the memory and knowledge management aspect of cumulative learners, MLN-CLA keeps track of knowledge elements with the Knowledge List. Knowledge Categories are used to cluster knowledge. Only relevant clusters have to be kept in memory to free up memory for other processes.

Super Knowledge Categories

However, a limitation of MLN-CLA is the inability to prevent super Knowledge Categories. Over time evidence can contain large amounts of new knowledge over the same domains. This evidence is clustered into the same Knowledge Category causing it to grow significantly larger than other categories. Such a super Knowledge Category is detrimental to efficient management of knowledge. In the extreme case that MLN-CLA contains only one large Knowledge Category it is no different from a batch learner. A method for preventing or splitting super categories is required.

On the other hand, new evidence fed into MLN-CLA that only contains evidence of new domains in each learning step would result in a Knowledge List full of single formula Knowledge Categories. This is intended behaviour. The micro Knowledge Categories are highly efficient for memory management as only a ground MLN for one formula has to be kept in memory for inference. The micro categories will only merge together once new formulas are learned from evidence that relate domains from different Knowledge Categories to each other.

A possible solution to super categories dominating the Knowledge List is to split up the super category into smaller sub categories and recording the parent-child relation between the super and sub categories into a taxonomy or ontology of Knowledge Categories. Such a taxonomy would allow for efficient look up of knowledge subsets to use for inference. A taxonomy on top of the Knowledge List transforms the knowledge base from a flat structure to a hierarchical tree structure.

7-2-1 Knowledge Categories and intersecting domains

In the merging operation of two Knowledge Categories only those categories that are subset or supersets of each other are merged together. Often formulas do share domains but the sets are not subsets. These formulas share an intersection of domains. In the ground Markov network the features of these formulas are connected via edges and thus influence each other. However, in MLN-CLA the formulas are put into separate Knowledge Categories. The relations between Knowledge Category formulas is not exploited during training as each category is trained as its own independent MLN. The domains that fall outside the intersection set of the two formulas make the categorisation of the two formulas ambiguous. During inference, all categories in the Knowledge List are converted into one large MLN. In this MLN the features once again overlap. The difference being that the relations between features are not optimized for during training.

The ambiguity problem can be lifted with the previously discussed taxonomy of Knowledge Categories. In such a taxonomy the ambiguous domains can be compared to more abstract categories higher up in the taxonomy hierarchy. The formulas can be both be categorised under the same parent category. This approach does exploit the knowledge captured by formula domain intersections.

7-2-2 Knowledge List pruning

Currently MLN-CLA has no pruning mechanism. Although structure learning is able to result in shortened formulas and weight learning can generate zero weight formulas, no formulas are removed from the Knowledge List. In MLN-CLA all previously seen knowledge is kept. The reason being that any learned knowledge might be relevant again later and knowing that some knowledge is irrelevant is also beneficial. Formulas with weight zero do not influence the inference outcome. The only downside of retaining all gained knowledge is an increased memory footprint.

As Thórisson noted, the knowledge management aspect of cumulative learners must prune the knowledge base to prevent the memory footprint from becoming too large [18]. A possible mechanism for MLN-CLA is to keep track of how long ago a piece of knowledge was last interacted with. Possible interactions are: used in inference, seen in evidence or knowledge merged into another category. Formulas with weight zero can be pruned if the last interaction was past a pre-determined threshold. Pruning the Knowledge List could lead to an unstable cumulative learning system.

7-2-3 Knowledge updating strategies

The knowledge updating strategies, CL-Naive, CL-Conservative and CL-Balanced, are set up to demonstrate the effect of catastrophic forgetting. Catastrophic forgetting is the overwriting of essential old knowledge by new knowledge. The naive approach to merging categories exemplifies this phenomenon. This strategy always discards old knowledge for new knowledge regardless of the quality of the new knowledge or the importance of the old knowledge.

The conservative strategy is more careful and looks at the quality of new knowledge before overwriting the old. However, the quality of new knowledge might be better, the old knowledge could still be based on relevant evidence. The conservative approach can still cause catastrophic forgetting.

The balanced strategy best prevents catastrophic forgetting of the three strategies. It always takes the quality of the old knowledge into account through weighted averaging of formula weights. However, this strategy is not perfect. With enough new evidence the contribution of the old knowledge to the new averaged weight is insignificant. With the balanced strategy it still possible to fully overwrite the old knowledge with the new, but it requires a large amount of evidence to convince MLN-CLA-Balanced its old knowledge was incorrect.

Conversely, the balanced strategy can also suffer from too strong a belief in its current formula weights. After a large number of learning steps the evidence count for a weight can become insurmountable by new evidence. In this situation the new evidence cannot provide a significant contribution to the newly averaged weight anymore. This situation only occurs if the learning steps are more or less equal in amount of new evidence. With varying amounts of new data in each step, a large step can still provide a significant contribution to the newly averaged formula weights.

7-3 Generality

The Markov properties of MLN-CLA enable it to learn formulas about any task, making it a multi-task learner. However, MLN-CLA is restricted in its input modality to first-order logic text. It cannot learn directly from videos, images or robot sensors. A translation step before the algorithm input can improve the generality of MLN-CLA. Furthermore, it does not incorporate fuzzy semantics. In MLN-CLA all evidence is either true or false. In fuzzy logic ground atoms in the evidence have an associated weight between 0 and 1 that indicates how certain it is that the atom is true [50]. Extending MLN-CLA with fuzzy semantics enables it to determine how strong it believes the weight of a formula is correct. This addition expands the reasoning dimension of the algorithm, increasing generality.

Chapter 8

Conclusion

In this research the MLN Cumulative Learning Algorithm was introduced. The algorithm is an extension to the Markov Logic Network framework for probabilistic reasoning over logical statements. MLN-CLA can automatically incorporate new first-order logic formulas into a knowledge base without any data annotations. Furthermore, it can automatically incorporate information about new domain constants and update relevant parts of the knowledge base. MLN-CLA has different knowledge updating strategies. The balanced strategy is most applicable when learning new constants. The conservative strategy works best in new formula learning.

MLN-CLA is a sequential cumulative learner. Within each learning step, the algorithm can incorporate any finite amount of new evidence. The algorithm can accept information on any knowledge domain, but only in one modality: first-order logic. The novel Knowledge Categories and knowledge updating strategies help keep the cumulative learner stable over time by preventing catastrophic forgetting.

In future work new knowledge updating strategies for intersecting domain sets of Knowledge Categories could be investigated. These strategies could be made more effective by the inclusion of a Knowledge Category taxonomy. The pruning of Knowledge Categories to keep the knowledge base compact is another research area. In addition, new studies could investigate the effect of variable learning step sizes and more complex knowledge bases on the stability of MLN-CLA.

Bibliography

[1] T. Lesort, V. Lomonaco, A. Stoian, D. Maltoni, D. Filliat, and N. Díaz-Rodríguez, "Continual learning for robotics: Definition, framework, learning strategies, opportunities and challenges," *Information Fusion*, vol. 58, pp. 52–68, 2020.

[2] A. Farhadi, I. Endres, D. Hoiem, and D. Forsyth, "Describing objects by their attributes," in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1778–1785, 2009.

[3] F. Xu, H. Uszkoreit, Y. Du, W. Fan, D. Zhao, and J. Zhu, "Explainable ai: A brief survey on history, research areas, approaches and challenges," in *Natural Language Processing and Chinese Computing: 8th CCF International Conference, NLPCC 2019, Dunhuang, China, October 9–14, 2019, Proceedings, Part II 8*, pp. 563–574, Springer, 2019.

[4] R. Dwivedi, D. Dave, H. Naik, S. Singhal, R. Omer, P. Patel, B. Qian, Z. Wen, T. Shah, G. Morgan, *et al.*, "Explainable AI (XAI): Core ideas, techniques, and solutions," *ACM Computing Surveys*, vol. 55, no. 9, pp. 1–33, 2023.

[5] A. Sheth, K. Roy, and M. Gaur, "Neurosymbolic ai – why, what, and how," 2023.

[6] A. Sheth, M. Gaur, K. Roy, R. Venkataraman, and V. Khandelwal, "Process knowledge-infused ai: Toward user-level explainability, interpretability, and safety," *IEEE Internet Computing*, vol. 26, no. 5, pp. 76–84, 2022.

[7] M. Ahn, A. Brohan, N. Brown, Y. Chebotar, O. Cortes, B. David, C. Finn, C. Fu, K. Gopalakrishnan, K. Hausman, A. Herzog, D. Ho, J. Hsu, J. Ibarz, B. Ichter, A. Irpan, E. Jang, R. J. Ruano, K. Jeffrey, S. Jesmonth, N. J. Joshi, R. Julian, D. Kalashnikov, Y. Kuang, K.-H. Lee, S. Levine, Y. Lu, L. Luu, C. Parada, P. Pastor, J. Quiambao, K. Rao, J. Rettinghouse, D. Reyes, P. Sermanet, N. Sievers, C. Tan, A. Toshev, V. Vanhoucke, F. Xia, T. Xiao, P. Xu, S. Xu, M. Yan, and A. Zeng, "Do As I Can, Not As I Say: Grounding Language in Robotic Affordances," 2022.

[8] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 779–788, 2016.

[9] A. Kirillov, E. Mintun, N. Ravi, H. Mao, C. Rolland, L. Gustafson, T. Xiao, S. Whitehead, A. C. Berg, W.-Y. Lo, P. Dollár, and R. Girshick, "Segment anything," 2023.

[10] A. Chowdhery, S. Narang, J. Devlin, M. Bosma, G. Mishra, A. Roberts, P. Barham, H. W. Chung, C. Sutton, S. Gehrmann, P. Schuh, K. Shi, S. Tsvyashchenko, J. Maynez, A. Rao, P. Barnes, Y. Tay, N. Shazeer, V. Prabhakaran, E. Reif, N. Du, B. Hutchinson, R. Pope, J. Bradbury, J. Austin, M. Isard, G. Gur-Ari, P. Yin, T. Duke, A. Levskaya, S. Ghemawat, S. Dev, H. Michalewski, X. Garcia, V. Misra, K. Robinson, L. Fedus, D. Zhou, D. Ippolito, D. Luan, H. Lim, B. Zoph, A. Spiridonov, R. Sepassi, D. Dohan, S. Agrawal, M. Omernick, A. M. Dai, T. S. Pillai, M. Pellat, A. Lewkowycz, E. Moreira, R. Child, O. Polozov, K. Lee, Z. Zhou, X. Wang, B. Saeta, M. Diaz, O. Firat, M. Catasta, J. Wei, K. Meier-Hellstern, D. Eck, J. Dean, S. Petrov, and N. Fiedel, "Palm: Scaling language modeling with pathways," 2022.

[11] H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, N. Bashlykov, S. Batra, P. Bhargava, S. Bhosale, D. Bikel, L. Blecher, C. C. Ferrer, M. Chen, G. Cucurull, D. Esiobu, J. Fernandes, J. Fu, W. Fu, B. Fuller, C. Gao, V. Goswami, N. Goyal, A. Hartshorn, S. Hosseini, R. Hou, H. Inan, M. Kardas, V. Kerkez, M. Khabsa, I. Kloumann, A. Korenev, P. S. Koura, M.-A. Lachaux, T. Lavril, J. Lee, D. Liskovich, Y. Lu, Y. Mao, X. Martinet, T. Mihaylov, P. Mishra, I. Molybog, Y. Nie, A. Poulton, J. Reizenstein, R. Rungta, K. Saladi, A. Schelten, R. Silva, E. M. Smith, R. Subramanian, X. E. Tan, B. Tang, R. Taylor, A. Williams, J. X. Kuan, P. Xu, Z. Yan, I. Zarov, Y. Zhang, A. Fan, M. Kambadur, S. Narang, A. Rodriguez, R. Stojnic, S. Edunov, and T. Scialom, "Llama 2: Open Foundation and Fine-Tuned Chat Models," 2023.

[12] M. Richardson and P. Domingos, "Markov logic networks," *Machine Learning*, vol. 62, pp. 107–136, 2006. https://doi.org/10.1007/s10994-006-5833-1.

[13] L. Jamone, E. Ugur, A. Cangelosi, L. Fadiga, A. Bernardino, J. Piater, and J. Santos-Victor, "Affordances in psychology, neuroscience, and robotics: A survey," *IEEE Transactions on Cognitive and Developmental Systems*, vol. 10, no. 1, pp. 4–25, 2018.

[14] J. Gibson and L. Carmichael, *The Senses Considered as Perceptual Systems.* Houghton Mifflin, 1966.

[15] J. Gibson, *The ecological approach to visual perception.* Houghton Mifflin, 1979.

[16] Y. Xian, C. H. Lampert, B. Schiele, and Z. Akata, "Zero-shot learning—a comprehensive evaluation of the good, the bad and the ugly," *IEEE transactions on pattern analysis and machine intelligence*, vol. 41, no. 9, pp. 2251–2265, 2018.

[17] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, ch. 19. Pearson, 3rd ed., 2010.

[18] K. R. Thórisson, J. Bieger, X. Li, and P. Wang, "Cumulative learning," in *Artificial General Intelligence* (P. Hammer, P. Agrawal, B. Goertzel, and M. Iklé, eds.), (Cham), pp. 198–208, Springer International Publishing, 2019.

[19] J. Kirkpatrick, R. Pascanu, N. Rabinowitz, J. Veness, G. Desjardins, A. A. Rusu, K. Milan, J. Quan, T. Ramalho, A. Grabska-Barwinska, D. Hassabis, C. Clopath, D. Kumaran,

and R. Hadsell, “Overcoming catastrophic forgetting in neural networks,” *Proceedings of the National Academy of Sciences*, vol. 114, no. 13, pp. 3521–3526, 2017.

[20] M. De Lange, R. Aljundi, M. Masana, S. Parisot, X. Jia, A. Leonardis, G. Slabaugh, and T. Tuytelaars, “A continual learning survey: Defying forgetting in classification tasks,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 44, pp. 3366–3385, July 2022.

[21] S. Cui, T. Zhu, X. Zhang, and H. Ning, “MCLA: Research on cumulative learning of Markov Logic Network,” *Knowledge-Based Systems*, vol. 242, p. 108352, 2022.

[22] L. Getoor and B. Taskar, *Introduction to Statistical Relational Learning.* Adaptive computation and machine learning, The MIT Press, 08 2007.

[23] D. Koller, N. Friedman, L. Getoor, and B. Taskar, “Graphical Models in a Nutshell,” in *Introduction to Statistical Relational Learning*, The MIT Press, 08 2007.

[24] J. Pearl, *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference.* San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1988.

[25] M. Lopes, F. S. Melo, and L. Montesano, “Affordance-based imitation learning in robots,” in *2007 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 1015–1021, 2007.

[26] L. Montesano, M. Lopes, A. Bernardino, and J. Santos-Victor, “Learning object affordances: From sensory–motor coordination to imitation,” *IEEE Transactions on Robotics*, vol. 24, no. 1, pp. 15–26, 2008.

[27] A. Pfeffer, “A bayesian language for cumulative learning,” in *Proceedings of AAAI 2000 Workshop on Learning Statististical Models from Relational Data*, 2000.

[28] K. Yue, Q. Fang, X. Wang, J. Li, and W. Liu, “A parallel and incremental approach for data-intensive learning of bayesian networks,” *IEEE Transactions on Cybernetics*, vol. 45, no. 12, pp. 2890–2904, 2015.

[29] W. Liu, K. Yue, M. Yue, Z. Yin, and B. Zhang, “A bayesian network-based approach for incremental learning of uncertain knowledge,” *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, vol. 26, no. 01, pp. 87–108, 2018.

[30] H. Liu, Z. SU, Y. Liu, L. Zhang, R. Yin, and Z. Ying, “An improved incremental structure learning algorithm for bayesian networks,” in *2019 6th International Conference on Systems and Informatics (ICSAI)*, pp. 505–510, 2019.

[31] A. Kumar, S. Chatterjee, and P. Rai, “Bayesian structural adaptation for continual learning,” in *Proceedings of the 38th International Conference on Machine Learning* (M. Meila and T. Zhang, eds.), vol. 139 of *Proceedings of Machine Learning Research*, pp. 5850–5860, PMLR, 18–24 Jul 2021.

[32] E. Şahin, M. Cakmak, M. R. Doğar, E. Uğur, and G. Üçoluk, “To afford or not to afford: A new formalization of affordances toward affordance-based robot control,” *Adaptive Behavior*, vol. 15, no. 4, pp. 447–472, 2007.

[33] P. Parag, *Markov logic: theory, algorithms and applications.* University of Washington, 2009.

[34] P. Domingos and D. Lowd, “Unifying logical and statistical ai with markov logic,” *Commun. ACM*, vol. 62, p. 74–83, jun 2019.

[35] P. Singla and P. Domingos, “Discriminative training of markov logic networks,” in *AAAI*, vol. 5, pp. 868–873, 2005.

[36] S. Kok and P. Domingos, “Learning the structure of markov logic networks,” in *Proceedings of the 22nd international conference on Machine learning*, pp. 441–448, 2005.

[37] S. Kok and P. M. Domingos, “Learning markov logic networks using structural motifs.,” in *ICML*, vol. 10, pp. 551–558, 2010.

[38] D. Lowd and P. Domingos, “Efficient weight learning for markov logic networks,” in *European conference on principles of data mining and knowledge discovery*, pp. 200–211, Springer, 2007.

[39] P. Singla, A. Nath, and P. Domingos, “Approximate lifting techniques for belief propagation,” *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 28, Jun. 2014.

[40] J. Van Haaren, G. Van den Broeck, W. Meert, and J. Davis, “Lifted generative learning of markov logic networks,” *Machine Learning*, vol. 103, pp. 27–55, Apr 2016.

[41] G. Van den Broeck, W. Meert, and J. Davis, “Lifted generative parameter learning,” in *AAAI Workshop-Technical Report*, pp. 87–94, AAAI, 2013.

[42] V. Sharma, N. A. Sheikh, H. Mittal, V. Gogate, and P. Singla, “Lifted Marginal MAP Inference,” 2018.

[43] V. Gogate and P. Domingos, “Probabilistic theorem proving,” *Communications of the ACM*, vol. 59, no. 7, pp. 107–115, 2016.

[44] V. Belle, “Open-universe weighted model counting,” Feb. 2017.

[45] D. Jain, A. Barthels, and M. Beetz, “Adaptive markov logic networks: Learning statistical relational models with dynamic parameters,” in *ECAI 2010*, pp. 937–942, IOS Press, 2010.

[46] H. Mittal, A. Bhardwaj, V. Gogate, and P. Singla, “Domain-size aware markov logic networks,” in *Proceedings of the Twenty-Second International Conference on Artificial Intelligence and Statistics* (K. Chaudhuri and M. Sugiyama, eds.), vol. 89 of *Proceedings of Machine Learning Research*, pp. 3216–3224, PMLR, 16–18 Apr 2019.

[47] S. Malhotra and L. Serafini, “On projectivity in markov logic networks,” in *Machine Learning and Knowledge Discovery in Databases* (S. Amini, Massih-Rezavand Canu, A. Fischer, T. Guns, P. Kralj Novak, and G. Tsoumakas, eds.), (Cham), pp. 223–238, Springer Nature Switzerland, 2023.

[48] V. David, R. Fournier-S’niehotta, and N. Travers, “Parameterisation of reasoning on temporal markov logic networks,” 2022.

[49] V. David, R. Fournier-S'niehotta, and N. Travers, "Neomapy: A parametric framework for reasoning with map inference on temporal markov logic networks," in *Proceedings of the 32nd ACM International Conference on Information and Knowledge Management*, CIKM '23, (New York, NY, USA), p. 400–409, Association for Computing Machinery, 2023.

[50] D. Nyga and M. Beetz, "Reasoning about unmodelled concepts - incorporating class taxonomies in probabilistic relational models," 2015.

[51] J. Choi, C. Choi, E. Lee, and P. Kim, *Markov Logic Network Based Social Relation Inference for Personalized Social Search*, pp. 195–202. Cham: Springer International Publishing, 2015.

[52] H. Papadopoulos and G. Tzanetakis, "Exploiting structural relationships in audio music signals using markov logic networks," in *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pp. 1–5, IEEE, 2013.

[53] E. Ha, J. Rowe, B. Mott, and J. Lester, "Goal recognition with markov logic networks for player-adaptive games," in *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, vol. 7, pp. 32–39, 2011.

[54] Y. Zhu, A. Fathi, and L. Fei-Fei, "Reasoning about object affordances in a knowledge base representation," in *Computer Vision – ECCV 2014* (D. Fleet, T. Pajdla, B. Schiele, and T. Tuytelaars, eds.), (Cham), pp. 408–424, Springer International Publishing, 2014.

[55] D. Yu, X. Liu, S. Pan, A. Li, and B. Yang, "A novel neural-symbolic system under statistical relational learning," 2023.

[56] G. Marra and O. Kuželka, "Neural markov logic networks," 2020.

[57] L. Torrey and J. Shavlik, "Policy transfer via markov logic networks," in *International Conference on Inductive Logic Programming*, pp. 234–248, Springer, 2009.

[58] I. Donadello, L. Serafini, and A. d'Avila Garcez, "Logic tensor networks for semantic image interpretation," 2017.

[59] F. Bianchi and P. Hitzler, "On the capabilities of logic tensor networks for deductive reasoning," in *AAAI Spring Symposium Combining Machine Learning with Knowledge Engineering*, 2019.

[60] S. Badreddine, A. d'Avila Garcez, L. Serafini, and M. Spranger, "Logic tensor networks," *Artificial Intelligence*, vol. 303, p. 103649, 2022.

[61] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[62] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, *et al.*, "Language models are few-shot learners," *Advances in neural information processing systems*, vol. 33, pp. 1877–1901, 2020.

[63] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, and I. Polosukhin, “Attention is all you need,” in *Advances in Neural Information Processing Systems* (I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, eds.), vol. 30, Curran Associates, Inc., 2017.

[64] D. Patterson, J. Gonzalez, Q. Le, C. Liang, L.-M. Munguia, D. Rothchild, D. So, M. Texier, and J. Dean, “Carbon emissions and large neural network training,” *arXiv preprint arXiv:2104.10350*, 2021.

[65] S. Harnad, “To cognize is to categorize: Cognition is categorization,” in *Handbook of Categorization in Cognitive Science* (H. Cohen and C. Lefebvre, eds.), ch. 2, pp. 21–54, San Diego: Elsevier, 2nd ed., 2017.

[66] S. Rüping, “Incremental learning with support vector machines,” in *Proceedings of the 2001 IEEE International Conference on Data Mining*, ICDM ’01, (USA), p. 641–642, IEEE Computer Society, 2001.

[67] G. Fei, S. Wang, and B. Liu, “Learning Cumulatively to Become More Knowledgeable,” in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD ’16, (New York, NY, USA), p. 1565–1574, Association for Computing Machinery, 2016.

[68] K. Seddiki, P. Saudemont, F. Precioso, N. Ogrinc, M. Wisztorski, M. Salzet, I. Fournier, and A. Droit, “Cumulative learning enables convolutional neural network representations for small mass spectrometry data classification,” *Nature Communications*, vol. 11, p. 5595, 11 2020.

[69] S. Cui, T. Zhu, X. Zhang, L. Chen, L. Mao, and H. Ning, “Generating markov logic networks rulebase based on probabilistic latent semantics analysis,” *Tsinghua Science and Technology*, vol. 28, no. 5, pp. 952–964, 2023.

[70] T. N. Huynh and R. J. Mooney, “Online structure learning for markov logic networks,” in *Machine Learning and Knowledge Discovery in Databases* (D. Gunopulos, T. Hofmann, D. Malerba, and M. Vazirgiannis, eds.), (Berlin, Heidelberg), pp. 81–96, Springer Berlin Heidelberg, 2011.

[71] B. L. Richards and R. J. Mooney, *Learning relations by pathfinding.* Artificial Intelligence Laboratory, University of Texas at Austin, 1992.

[72] J. Pearl and A. Paz, *GRAPHOIDS: A Graph-based Logic for Reasoning about Relevance Relations Or when Would X Tell You More about Y If You Already Know Z.* University of California, Computer Science Department, Cognitive Systems Laboratory, 1986.

[73] J. A. Robinson, “A machine-oriented logic based on the resolution principle,” *Journal of the ACM (JACM)*, vol. 12, no. 1, pp. 23–41, 1965.

[74] H. Poon and P. Domingos, “Sound and efficient inference with probabilistic and deterministic dependencies,” in *AAAI*, vol. 6, pp. 458–463, 2006.

[75] J. Besag, “Statistical analysis of non-lattice data,” *Journal of the Royal Statistical Society Series D: The Statistician*, vol. 24, no. 3, pp. 179–195, 1975.

[76] Q.-T. Dinh, M. Exbrayat, and C. Vrain, "Generative structure learning for markov logic networks based on graph of predicates," in *Twenty-Second International Joint Conference on Artificial Intelligence*, 2011.

[77] J. Van Haaren and J. Davis, "Markov network structure learning: A randomized feature generation approach," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 26, pp. 1148–1154, 2012.

[78] X. Lu, J. Liu, Z. Gu, H. Tong, C. Xie, J. Huang, Y. Xiao, and W. Wang, "Parsing natural language into propositional and first-order logic with dual reinforcement learning," in *Proceedings of the 29th International Conference on Computational Linguistics*, pp. 5419–5431, 2022.

[79] D. Jain, S. Waldherr, K. von Gleissenthall, A. Barthels, R. Wernicke, G. Wylezich, M. Schuster, P. Meyer, and D. Nyga, "ProbCog: A Toolbox for Statistical Relational Learning and Reasoning," 2021.

[80] D. Nyga, M. Picklum, M. Beetz, *et al.*, "pracmln – Markov logic networks in Python," 2013. https://www.pracmln.org/ accessed on April 4, 2023.

[81] S. Kok, P. Singla, M. Richardson, and P. Domingos, "The Alchemy system for statistical relational AI," tech. rep., Department of Computer Science and Engineering, University of Washington, Seattle, WA, 2005.

[82] B. Yao, X. Jiang, A. Khosla, A. L. Lin, L. Guibas, and L. Fei-Fei, "Human action recognition by learning bases of action attributes and parts," in *2011 International conference on computer vision*, pp. 1331–1338, IEEE, 2011.

[83] C. Fellbaum, *WordNet: An Electronic Lexical Database.* MIT Press, 1998.

[84] CMG Lee, "The ROC space for a "better" and "worse" classifier," 2018. [Online; accessed October 19, 2023].

[85] S. Kok and P. Domingos, "Statistical predicate invention," in *Proceedings of the 24th International Conference on Machine Learning*, ICML '07, (New York, NY, USA), p. 433–440, Association for Computing Machinery, 2007.

[86] Y. Yang, S. Xiong, A. Payani, E. Shareghi, and F. Fekri, "Harnessing the power of large language models for natural language to first-order logic translation," 2023.

[87] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft coco: Common objects in context," in *European conference on computer vision*, pp. 740–755, Springer, 2014.

[88] R. Krishna, Y. Zhu, O. Groth, J. Johnson, K. Hata, J. Kravitz, S. Chen, Y. Kalantidis, L.-J. Li, D. A. Shamma, *et al.*, "Visual genome: Connecting language and vision using crowdsourced dense image annotations," *International journal of computer vision*, vol. 123, pp. 32–73, 2017.

A - Alchemy syntax

The Alchemy software packages requires MLNs and evidence databases of ground atoms to be written in a specific format. The Alchemy grammar rules are set out in the list below.

- Predicates must start with upper case
- Predicates must contain at least one argument
- Constants must start with upper case
- Variables must start with lower case
- A full stop '.' at end of formula indicates hard clause that has infinite weight
- The '!'-symbol before a predicate indicates the negation ($\neg$) of that predicate
- The '+'-symbol before a variable indicates that a separate formula is created for each constant of the domain of that variable
- List of allowed MLN symbols:
  - Quantifiers: $\forall, \exists$
  - Logical operators: $\wedge$, $\vee$, $\Leftrightarrow$, $\Rightarrow$
  - Alchemy specific: $+$, !, .

B - Datasets

Table 1: Objects and their corresponding actions of the Zhu *et al.* training dataset [54].

	Grasp	**Lift**	**Throw**	**Push**	**Fix**	**Ride**	**Play**	**Watch**	**Sit on**	**Feed**	**Row**	**Pour from**	**Write with**	**Type on**	**Total**
Automobile engine	0	0	0	0	1	0	0	0	0	0	0	0	0	0	**1**
Axe	1	1	1	1	0	0	0	0	0	0	0	0	0	0	**4**
Bicycle	0	1	0	1	1	1	0	0	1	0	0	0	0	0	**5**
Bottle	1	1	1	1	0	0	0	0	0	0	0	1	0	0	**5**
Camera	1	1	1	1	1	0	0	0	0	0	0	0	0	0	**6**
Can	1	1	1	1	0	0	0	0	0	0	0	1	0	0	**5**
Car tire	0	1	0	1	0	0	0	0	0	0	0	0	0	0	**2**
Carving knife	1	1	1	1	0	0	0	0	0	0	0	0	0	0	**4**
Chair	0	1	0	1	0	0	0	0	1	0	0	0	0	0	**3**
Chalk	1	1	1	1	0	0	0	0	0	0	0	0	1	0	**5**
Cleaver	1	1	1	1	0	0	0	0	0	0	0	0	0	0	**4**
Desktop computer	0	1	0	1	1	0	0	1	0	0	0	0	0	1	**5**
Dish	1	1	1	1	0	0	0	0	0	0	0	0	0	0	**4**
Dog	0	1	0	1	0	0	0	0	0	1	0	0	0	0	**3**
Dustcloth	1	1	1	1	0	0	0	0	0	0	0	0	0	0	**4**
Fishing pole	1	1	0	1	1	0	0	0	0	0	0	0	0	0	**4**
Food turner	1	1	1	1	0	0	0	0	0	0	0	0	0	0	**4**
Frisbee	1	1	1	1	0	0	0	0	0	0	0	0	0	0	**4**
Guitar	1	1	1	1	1	0	1	0	0	0	0	0	0	0	**6**
Hand saw	1	1	1	1	0	0	0	0	0	0	0	0	0	0	**4**
Handset	1	1	1	1	1	0	0	0	0	0	0	0	0	0	**5**
Horse	0	0	0	0	0	1	0	0	1	1	0	0	0	0	**3**
Laptop	1	1	1	1	1	0	0	1	0	0	0	0	0	1	**7**
Microscope	1	1	0	1	1	0	0	0	0	0	0	0	0	0	**5**
Mobile phone	1	1	1	1	1	0	0	0	0	0	0	0	0	0	**5**
Mop	1	1	0	1	0	0	0	0	0	0	0	0	0	0	**3**
Pen	1	1	1	1	0	0	0	0	0	0	0	0	1	0	**5**
Pitcher	1	1	1	1	0	0	0	0	0	0	0	1	0	0	**5**
Power saw	1	1	0	1	1	0	0	0	0	0	0	0	0	0	**4**
Shopping cart	0	0	0	1	1	1	0	0	1	0	0	0	0	0	**4**
Small boat	0	0	0	1	0	1	0	0	1	0	1	0	0	0	**4**
Sofa	0	0	0	0	0	0	0	0	1	0	0	0	0	0	**1**
Teapot	1	1	1	1	0	0	0	0	0	0	0	1	0	0	**5**
Telescope	1	1	0	1	1	0	0	0	0	0	0	0	0	0	**5**
Television	0	1	0	1	1	0	0	1	0	0	0	0	0	0	**4**
Toothbrush	1	1	1	1	0	0	0	0	0	0	0	0	0	0	**4**
Umbrella	1	1	1	1	1	0	0	0	0	0	0	0	0	0	**5**
Vacuum cleaner	0	1	0	1	1	0	0	0	0	0	0	0	0	0	**3**
Violin	1	1	1	1	1	0	1	0	0	0	0	0	0	0	**6**
Wheelbarrow	0	1	0	1	1	1	0	0	1	0	0	0	0	0	**5**
Frequency	**27**	**35**	**22**	**37**	**18**	**5**	**2**	**3**	**7**	**2**	**1**	**4**	**2**	**2**	**170**

Table 2: Objects and their corresponding actions of the Zhu *et al.* test dataset [54]. This dataset forms the groundtruth of the experiments in chapter 5.

	Grasp	**Lift**	**Throw**	**Push**	**Fix**	**Ride**	**Play**	**Watch**	**Sit on**	**Feed**	**Row**	**Pour from**	**Write with**	**Type on**	**Total**
Banjo	1	1	1	1	1	0	1	0	0	0	0	0	0	0	**6**
Bench	0	1	0	1	0	0	0	0	1	0	0	0	0	0	**3**
Bowl	1	1	1	1	0	0	0	0	0	0	0	0	0	0	**4**
Broom	1	1	0	1	0	0	0	0	0	0	0	0	0	0	**3**
Camel	0	0	0	0	0	1	0	0	1	1	0	0	0	0	**3**
Cat	0	1	0	1	0	0	0	0	0	1	0	0	0	0	**3**
Coffee cup	1	1	1	1	0	0	0	0	0	0	0	1	0	0	**5**
Donkey	0	0	0	0	0	1	0	0	1	1	0	0	0	0	**3**
Flagon	1	1	1	1	0	0	0	0	0	0	0	1	0	0	**5**
Hammer	1	1	1	1	0	0	0	0	0	0	0	0	0	0	**4**
Hand truck	0	1	0	1	1	1	0	0	1	0	0	0	0	0	**5**
Kayak	0	0	0	1	0	1	0	0	1	0	1	0	0	0	**4**
Monitor	0	1	0	1	1	0	0	1	0	0	0	0	0	0	**4**
Motorcycle	0	0	0	1	1	1	0	0	1	0	0	0	0	0	**4**
Pencil	1	1	1	1	0	0	0	0	0	0	0	0	1	0	**5**
Rhinoceros	0	0	0	0	0	1	0	0	1	1	0	0	0	0	**3**
Serving cart	0	0	0	1	1	1	0	0	1	0	0	0	0	0	**4**
Sickle	1	1	1	1	0	0	0	0	0	0	0	0	0	0	**4**
Spoon	1	1	1	1	0	0	0	0	0	0	0	0	0	0	**4**
Stool	0	1	0	1	0	0	0	0	1	0	0	0	0	0	**3**
Typewriter	0	1	0	1	1	0	0	0	0	0	0	0	0	1	**4**
Walkie-talkie	1	1	1	1	1	0	0	0	0	0	0	0	0	0	**5**
Frequency	**10**	**16**	**9**	**19**	**7**	**7**	**1**	**1**	**9**	**4**	**1**	**2**	**1**	**1**	**88**

C - Experiments MLN

The experiments consisting of five predicate declarations, six domains and five formulas by Zhu *et al.* is shown in Listing 1. The `HasAffordance` predicate is the query predicate in each of the experiments described in Chapter 5.

Listing 1: Zhu *et al.* MLN with a formula for all combinations of instantiations of all variables except for objects.

```
// Predicate declarations
HasAffordance(object, affordance)
IsA(object, category)
HasVisualAttribute(object, attribute)
HasWeight(object, weight)
HasSize(object, size)

// Formulas
0 IsA(x, +c1) => IsA(x, +c2)
0 IsA(x, +c) => HasAffordance(x, +aff)
0 HasVisualAttribute(x, +attr) => HasAffordance(x, +aff)
0 HasWeight(x, +w) => HasAffordance(x, +aff)
0 HasSize(x, +s) => HasAffordance(x, +aff)
```

D - Algorithms

Algorithm 4: Evidence counting algorithm

```
Input: Formula $F$, evidence $DB$
Output: Evidence count $z$
$z \leftarrow 0$
$A \leftarrow \emptyset$                                   // list of ground atoms
for $p \in F$                                              // where $p$ is predicate
 do
    $A \leftarrow \text{filter}(DB, p)$          // get list of ground atoms corresponding to $p$
    $z = z + \text{length}(A)$
end
```

Algorithm 5: Knowledge Lists merging algorithm

```
Input: Knowledge Lists $L_1, L_2$, merging method
Output: $L_1 : \{(F_1, w_1, z_1), D_1, K_1\}$
$\mathcal{D}_{L_1} \leftarrow \mathcal{D}_{L_1} \cup \mathcal{D}_{L_2}$
$\mathcal{P}_{L_1} \leftarrow \mathcal{P}_{L_1} \cup \mathcal{P}_{L_2}$
$C_m \leftarrow C_{L_2}$                                   // set of all KC in $L_2$
for $C_{L_2} \in L_2$ do
    $\mathcal{D}_{C_{L_2}} \leftarrow \text{domains} \in C_{L_2}$          // KC domains set
    for $C_{L_1} \in L_1$ do
        $\mathcal{D}_{C_{L_1}} \leftarrow \text{domains} \in C_{L_1}$
        if $\mathcal{D}_{C_{\mathcal{L}_\infty}} \subseteq \mathcal{D}_{C_{L_2}} \vee \mathcal{D}_{C_{L_1}} \supset \mathcal{D}_{C_{L_2}}$ then
            $C_{L_1} \leftarrow \text{mergeKC}(C_{L_1}, C_{L_2})$
            $C_m \setminus C_{L_2}$                        // discard the merged KC
        end
    end
end
for $C_{L_2} \in C_m$ do
    $\{C_{L_1}\} \cup C_{L_2}$                            // add non-merged KCs
end
return $L_1$
```

Algorithm 6: Knowledge Category creation algorithm

```
Input: Knowledge triplets $\{(F, w, z)\}$, formula domains $\{F_D\}$
Output: $C$
$C \leftarrow \emptyset$
$C_{triplets} \leftarrow \{(F, w, z)\}$
$C_{domains} \leftarrow \{F_D\}$
return $C$
```

Algorithm 7: Knowledge List initialisation algorithm

```
Input: $MLN : \{\mathcal{P}, (F, w), K_{MLN}\}$, merging method $M$, (optional) evidence
       $DB : \{K_{DB}\}$
Output: Knowledge List $L : \{P, D_{d \mapsto k}, C\}$
$L \leftarrow \emptyset$                          // initialise Knowledge List
$L_{\mathcal{P}} \leftarrow MLN_{\mathcal{P}}$                     // predicate declarations
$L_D \leftarrow domains(MLN_P)$     // parse arguments of predicate declarations
if $DB$ then
    // Parse evidence constants and map them to corresponding domains
    $L_D \leftarrow K_{DB}$
end
for $F, w \in MLN$ do
    // Gather domains in formula
    for $Predicate \in F$ do
        $F_D \leftarrow domains(Predicate)$
    end
    if $DB$ then
        $F_D \mapsto K_{DB}$
        // Count predicate evidence
        $z \leftarrow$ count_evidence($Predicate$)
    end
    else
        $z \leftarrow 0$
    end
    $C_F \leftarrow$ create_KC($(F, w, z), F_D$)
    // Check for existing KCs to merge $C_F$ into
    $match \leftarrow False$
    for $c_i \in L_C$ do
        if $C_{F_D} \subseteq c_i \vee c_i \subseteq C_{F_D}$ then
            $c_i \leftarrow$ mergeKC($c_i, C_F, M$)     // can merge with multiple KCs
        end
        $match \leftarrow True$
    end
    if $match = False$ then
        $L_C \leftarrow C_F$
    end
end
return $L$
```

Algorithm 8: Algorithm for merging two Knowledge Categories with three updating strategies

```
Input: Knowledge Categories $C_1, C_2$, merging method $M$
Output: $C_1 : \{(F_1, w_1, z_1), D_1\}$
if $\mathcal{D}_{C_2} \subseteq \mathcal{D}_{C_1} \vee \mathcal{D}_{C_1} \subseteq \mathcal{D}_{C_2}$ then
    // Same knowledge concept
    for $(F_2, w_2, z_2) \in \mathcal{T}_{C_2}$ do
        // check all formulas in $\mathcal{T}_{C_2}$ for conflicts
        for $(F_1, w_1, z_1) \in \mathcal{T}_{C_1}$ do
            if $F_2 \equiv F_1$ then // same formula conflict
                if $M = naive$ then
                    // overwrite weight and evidence count
                    $w_1 \leftarrow w_2$
                    $z_1 \leftarrow z_2$
                end
                if $M = conservative$ then
                    // overwrite weight and evidence count if more evidence for
                       $F_2$ than $F_1$
                    if $z_2 \geq z_1$ then
                        $w_1 \leftarrow w_2$
                        $z_1 \leftarrow z_2$
                    end
                end
                if $M = balanced$ then
                    // combine weights with weighted average (prevent division
                       by zero), sum evidence count
                    if $z1 = z2 = 0$ then
                        $w1 \leftarrow \frac{w_1 + w_2}{2}$
                    end
                    else
                        $w_1 \leftarrow \frac{z_1 w_1 + z_2 w_2}{z_1 + z_2}$
                        $z_1 \leftarrow z_1 + z_2$
                    end
                end
            end
        end
    end
end
```

Algorithm 9: Markov Logic Network Cumulative Learning Algorithm

```
Input: Knowledge List $L_{t=0}$, evidence $DB$
Output: Knowledge List $L_{t+1}$
$U \leftarrow \emptyset$                                          // unknown evidence
$E \leftarrow$ parse_evidence($DB$)                  // list of evidence by symbol type
foreach $e \in E$ do
    // Compile all unknown evidence
    if $e \notin L_{t=0}$ then
        $U \leftarrow e$
    end
end
if $U = \emptyset$ then      // all new evidence is known, perform structure learning
    $MLN \leftarrow$ createMLNfromKL($L_{t=0}$)
    $MLN_{new} \leftarrow$ structlearn($MLN, DB$)
    $L_{new} \leftarrow$ createKLfromMLN($MLN_{new}, DB$)
    $L_{t+1} \leftarrow$ mergeKL($L_{t=0}, L_{new}$)
else                            // put each element of $U$ in the Knowledge List
    foreach $u \in U$ do
        $match \leftarrow$ match_to_KC($u$)    // match each element in $U$ to a KC ink KL
        if $not\ match$ then
            addCategory($u$)
        end
    end
    foreach $(C_1, C_2) \in L$ do
        mergeKC($C_1, C_2$)                               // merge similar categories
    end
end
$MLN_{trained} \leftarrow \emptyset$
foreach $C \in L_{t=0}$ do
    $MLN_C \leftarrow$ createMLNfromKC($C$)
    $DB_C \leftarrow$ split_DB($DB, MLN_C$)
    $MLN_{trained} \leftarrow$ train_MLN($MLN_C, DB_C$)        // generative weight training
end
$MLN_{merged} \leftarrow$ mergeMLNs($MLN_{trained}$)
$L_{new} \leftarrow$ createKLfromMLN($MLN_{merged}$)
$L_{t+1} \leftarrow$ mergeKL($L_{t=0}, L_{new}$)
return $L_{t+1}$
```