Incremental Learning of Affordances using Markov Logic Networks

G.B.G. POTTER, TNO and TU Delft, The Netherlands
G.J. BURGHOUTS, TNO, The Netherlands
J. SIJS, TU Delft, The Netherlands

Affordances enable robots to have a semantic understanding of their surroundings. This allows them to have more acting flexibility when completing a given task. Capturing object affordances in a machine learning model is a difficult task, because of their dependence on contextual information. Markov Logic Networks (MLN) combine probabilistic reasoning with logic that is able to capture such context. Mobile robots operate in partially known environments wherein unseen object affordances can be observed. This new information must be incorporated into the existing knowledge, without having to retrain the MLN from scratch. We introduce the MLN Cumulative Learning Algorithm (MLN-CLA). MLN-CLA learns new relations in various knowledge domains by retaining knowledge and only updating the changed knowledge, for which the MLN is retrained. We show that MLN-CLA is effective for accumulative learning and zero-shot affordance inference, outperforming strong baselines.

CCS Concepts: • **Computing methodologies** → **Probabilistic reasoning**.

Additional Key Words and Phrases: affordances, incremental learning, Markov Logic Network

ACM Reference Format:
G.B.G. Potter, G.J. Burghouts, and J. Sijs. 2024. Incremental Learning of Affordances using Markov Logic Networks. 1, 1 (July 2024), 14 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

Affordances play an important role in semantic understanding of scenes in robotics. These affordances, first introduced by Gibson [1], are the potential actions that an object affords to an agent depending on object properties and state, action effects, situational context and agent capabilities. In a robotics context, affordances model the relation between the robot, an object, and the possible interactions between the two [2]. These affordances allow the robot to reason about its beliefs of the world in relation to the tasks and actions it may execute within the environment. Particularly in partially known environments, these affordances, in combination with reasoning about them, may result in more options for the robot to choose from. As a result affordances increase the possibility of the robot successfully completing its task [3, 4]. Gibson defined affordance as *what the environment offers, provides or makes available to an animal* [1]. His 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 can fuse this sensor data to reason about action possibilities present in the environment. Including a component in the perception-action system to reason about affordances, also called

Authors' addresses: G.B.G. Potter, george.potter@tno.nl, TNO and TU Delft, The Hague, The Netherlands; G.J. Burghouts, gertjan.burghouts@tno.nl, TNO, The Hague, The Netherlands; J. Sijs, j.sijs@tudelft.nl, TU Delft, Delft, The Netherlands.

affordance inference, would result in a robot that is more effective in planning its task given a current belief of the surrounding world.

Reasoning about affordances is strongly related to symbolic reasoning as an affordance is a symbolic concept. 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 is 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. An example of a (mathematical) language that embeds symbolic knowledge in logic is first-order logic (FOL) [5, 6]. 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, such as *all swans are white*, in the world has two major drawbacks: contradicting formulas and the Boolean interpretation of logic. Markov Logic Networks can solve these problems [7, 8].

A Markov Logic Network (MLN) is a knowledge base of first-order logic formulas with a weight attached to each formula [7, 8]. 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. Table 1 provides an example MLN that consists of three formulas. The formulas do not conflict logically, but semantically seem incorrect when taking into account that each formula is $\forall x, y$. This is reflected in the formula weights that soften the $\forall$ constraints. 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 inferring the object's affordance based on object properties. In section 3 we provide a more complete explanation of MLNs.

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

Weight	Formula	Meaning
1.3	`SharpEdge(x)` $\Rightarrow$ `Affordance(x, Cutting)`	If x has a sharp edge, it has the cutting affordance
2.8	`Heavy(x)` $\Rightarrow$ `!Affordance(x, Lifting)`	If x is heavy, it cannot be lifted
-4.5	`Affordance(x)` $\Leftrightarrow$ `Affordance(y)`	The affordance x and y are the same

In most applications MLN models are trained once before deployment [7, 9] in a process called *batch* or *offline* learning [10]. Updating these models afterwards with newly obtained information from the real world requires retraining the MLN from scratch. It is thus not very efficient from a computational point of view as old data needs to be processed again. This is illustrated in Figure 1 where our method stores knowledge for future reuse. Updating only parts of the model relevant to the newly obtained information saves on compute resources and time [11]. The Markov properties of MLNs enable partially updating the network [7]. Newly obtained information can thus be integrated into the existing knowledge over time in a manner called *cumulative*, *online*, *incremental* or *lifelong* learning [12]. The task of affordance inference is suitable for incremental learning as object affordances are difficult to define a priori without additional context from a given situation.

The ideal cumulative learner can learn many things at any time in a continuous manner. Our approach is a first step to this ideal situation as we introduce a discrete cumulative learner, i.e. it learns consecutively in batches. In discrete cumulative learning new batches of data are presented to the learner to extract knowledge from. An important feature for such a cumulative learners is to mitigate *catastrophic forgetting*. Catastrophic forgetting refers to the overwriting of

(a) Batch learning (b) MLN-CLA (ours)

Fig. 1. In batch learning two MLN-database pairs are combined into one before training. In MLN-CLA the pairs are only combined after training. Before training each MLN-database pair is trained separately in parallel.

(parts of) old knowledge during the learning of new knowledge, thus losing potentially critical information necessary to complete a task [13].

To our knowledge, only two methods exist in the literature that extend Markov Logic Networks with cumulative learning capabilities: OSL and MCLA. The Online Structure Learning (OSL) [14] method learns new formulas and new weights from incoming evidence at each time step. This method is prone to catastrophic forgetting due to the lack of a knowledge manager that prevents overwriting of old knowledge. Our method mitigates catastrophic forgetting by introducing knowledge updating strategies that define conditions for which formula weight overwriting is preferred. The MLN Cumulative Learning Algorithm (MCLA) [11] method employs a knowledge manager to cluster the knowledge in the knowledge base. The clustering helps in efficiently updating the formula weights of only relevant formulas in relation to newly obtained evidence. However, in MCLA all information about which formulas belong to which cluster is known a priori. Our approach lifts this requirement by introducing a knowledge clustering method based on co-occurrence of predicate domains in formulas.

The cumulative learner introduced in this article, called MLN-Cumulative Learning Algorithm (MLN-CLA), will resolve these limitations of catastrophic forgetting and a-priori clustering of formulas. MLN-CLA enables a robot to reason over existing object affordances and learn new ones. An overview of our algorithm is given in Figure 2. We demonstrate a cumulative learner capable of learning new affordances, objects, object properties, and relations between these concepts. Our contributions are:

- A new algorithm for incremental learning of MLNs called MLN-CLA.
- Three knowledge updating strategies to mitigate catastrophic forgetting.
- A new knowledge clustering method for knowledge base management.
- Zero-shot affordance inference experiments with MLN-CLA.

In section 2 we describe different approaches to cumulative learning of MLNs and their limitations. The mechanisms of MLN-CLA and the workings of the knowledge updating strategies are explained in section 3. In section 4 MLN-CLA is compared to batch learning methods in two scenarios. In addition, the performances of the updating strategies are compared against each other. section 5 provides a summary of our findings.

2 RELATED WORK

Previous approaches to affordance modelling used Bayesian networks (BN) to represent relations between objects and affordances [15–18]. Montesano et al. used a Bayesian network to model the interaction between objects, actions and effects [15–17]. Their model learns many of these types of relations 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 objects outside the set of objects the BN was trained on. Additionally, it can update its beliefs, i.e. the strength of a relation in the network, based on observations of interactions with objects. As a result, it

can adjust its predictions over time as it learns from new object interactions. A major drawback of BNs is their graph structure and dependency representation. BNs cannot handle cyclic dependencies without additional extensions. Cyclic dependencies occur when two variables have a direct or an indirect (via other variables) influence 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. For example, a BN cannot represent that an action has an effect and that effect influences an object's state, where the object state is a pre-condition for the action. This type of relation is circular and cannot be represented in an acyclic graph. Furthermore, 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 [19–23].

Large Language Models (LLM) [4] and neuro-symbolic AI [24] have been used to model affordances or otherwise encode logical relations in neural networks. The vast amount of literature available on the topic Explainable AI shows that neural networks are in fact black boxes [25, 26]. The outputs of these models are difficult to explain, which is a property not desirable for our use-case. Therefore, our approach models object properties and affordances in Markov Logic Networks, because they are based on the concept of Markov networks [27] and combine logical rules with probabilistic reasoning [7, 28].

Markov networks are similar to Bayesian networks but with the benefit that they do allow modelling of cyclic dependencies in an undirected graph. Moreover, Markov logic is a first-order logic that balances human interpretability with model expressiveness [8]. A typical MLN cannot be updated over time, i.e. it is not ready for cumulative learning. To enable this property one must transform the MLN into a knowledge base wherein the formulas, their weights, variables and evidence are managed. Cui et al. introduced a 'knowledge identifier' that is assigned a priori to manage the knowledge base [11]. Their knowledge manager assigns a unique identifier to different clusters of knowledge. Incoming evidence is categorised into these clusters, called Knowledge Categories, based on this knowledge identifier. To manage these clusters they introduce a Knowledge List. This Knowledge List is essentially a container for all Knowledge Categories. In MLN-CLA we adopt these concepts of Knowledge Categories and Knowledge Lists. We lift the requirement of labelling data beforehand through automatic categorisation. In contrast to [11], we show how to merge Knowledge Categories together to more efficiently cluster formulas in the Knowledge List. Cui et al. do not manage overwriting essential old knowledge by newly learned knowledge. OSL has no knowledge manager at all. We introduce an explicit knowledge merging step that handles formula weight overwriting conflicts. These conflicts arising from a newer version of a formula overwriting the old formula. Depending on the chosen knowledge updating strategy the conflict is resolved and the user is in control of how catastrophic forgetting is handled. The strategies can be as simple as 'only overwrite if the new formula is based on more evidence than the old one' or as complex as the user wants. A new strategy can easily be defined and used within our framework.

3 METHOD

Markov logic is a form of first-order logic (FOL). In FOL, formulas consist of four types of symbols: constants, variables, functions and predicates. Constants represent things within a domain, such as objects (domain): `Ball` (constant), `Chair` or actions: `Throw`, `Kick`. Variables range over constants in a domain, e.g. `y`. Predicates represent relations over their arguments, e.g. `HasWeight(Ball, Light)` where `Ball` and `Light` are of the domain object and weight respectively. Functions are not considered in MLNs. With the constant, variable and predicate symbols any FOL formula can be

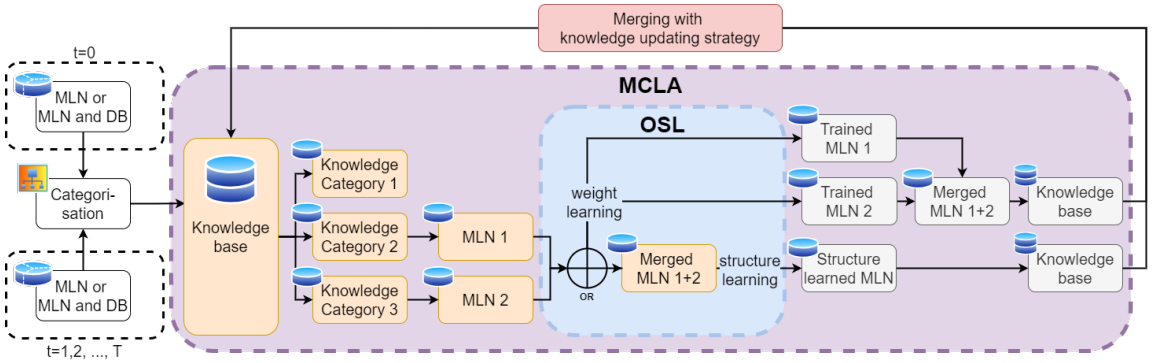


Fig. 2. MLN-CLA: We build our knowledge categorisation and knowledge updating strategies on top of the same concepts as MCLA [11], outlined in purple. With the blue outlined box we indicate the overlap with OSL [14].

constructed: $\forall y$ `HasWeigth(y, Heavy)` $\Rightarrow \neg$`Affordance(y, Throw)`, where `y` is of the domain `object`. A world is defined by the truth values of all possible predicates and constants combinations. Markov logic adds probabilistic weights to first-order logic formulas. These weights acts as soft constraints on the universal quantifier $\forall$ in FOL formulas. The weights indicate that a world that violates a formula is less probable but not impossible. A weight determines how strong the soft constraint is. An infinite weight is equal to first-order logic. A Markov Logic Network is a set of weighted first-order logic formulas. These formulas in combination with a set of constants form a Markov network [29]. A Markov network is a graphical representation of the probability distribution of the possible worlds. MLNs are templates for the construction of Markov networks. In MLN-CLA, illustrated in Figure 2, knowledge exists in the forms of Knowledge Lists, Knowledge Categories, MLN models and evidence. MLN models are simply the formulas with associated weights, such as in Table 1, that capture knowledge about the world. Evidence is the actual data consisting of solely predicates of which all arguments are constants. This evidence is used to determine the weights of the formulas in the MLN model during training. The Knowledge List and Categories are further examined as these concepts form the core of knowledge management in MLN-CLA. Both concepts are adapted from Cui et al. [11] and further refined.

3.1 Updating Knowledge Lists

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 C. The definition of a Knowledge List is given in Definition 3.1.

Definition 3.1 (Knowledge List).

$$\mathcal{P} = \{P_1(D_A), P_2(D_B, D_C), P_3(D_D), P_4(D_E, D_F, D_G), \ldots, P_W(D_X)\}$$
$$\mathcal{D} = \{D_A \mapsto \{k_{a1}, k_{a2}, \ldots, k_{aA}\}$$
$$\ldots$$
$$D_X \mapsto \{k_{x1}, k_{x2}, \ldots, k_{xC}\}\}$$
$$C = \{c_1, c_2, c_3, \ldots, c_Y\}$$

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. 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 2 on the right. When new evidence is fed into MLN-CLA, first all elements, i.e. predicates, domains and constants, are compared against the Knowledge List. Elements not in the list are marked as unknown and are added to the list after training. Elements already in the list are marked as known and used for training and evidence counting. 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.

Example 1 illustrates 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. its `size` or `weight`. Intuitively, because these predicates share domains, 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.

$$\mathcal{P} = \{\texttt{Size(object, size)}, \texttt{Shape(object, shape)}, \texttt{Affordance(object, action)}\}$$

$$\begin{aligned}\mathcal{D} = \{ \quad & \texttt{object} \mapsto \{\texttt{Ball, Glass, Shoe, Chair, ...}\} \\ & \texttt{size} \mapsto \{\texttt{Small, Large, ...}\} \\ & \texttt{shape} \mapsto \{\texttt{Round, Sphere, Cylinder, Cube, Flat, Convex, ...}\} \\ & \texttt{action} \mapsto \{\texttt{Push, Throw, Pull, Open, Close, Hit, Pick, Place, ...}\}\}\end{aligned}$$

$$C = \{C_1, C_2, C_3, \ldots, C_N\}$$

3.2 Updating Knowledge Categories

The second form of knowledge container is the Knowledge Category. 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 concepts as specified by the domains. The definition of a Knowledge Category is given in Definition 3.2. 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 3.2 (Knowledge Category).

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

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. 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

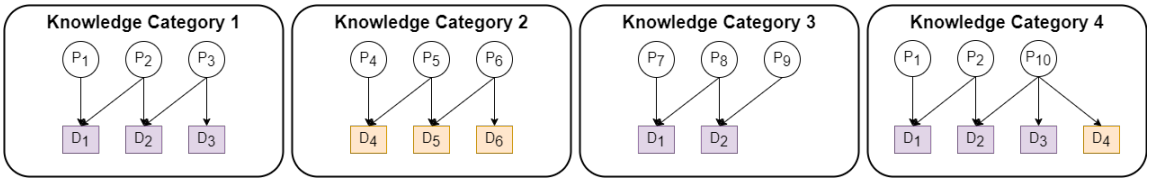


Fig. 3. Illustration of different Knowledge Categories.

accompanied by its weight and evidence count. The evidence counts suggest these formulas were learned from different evidence databases as the count is only based on predicate occurrence in the data.

Example 2.

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

A Knowledge Category of a Knowledge List is unique within the Knowledge List. It contains all formulas in a Knowledge List that relate to the same concept, essentially forming a knowledge 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, as defined by the predicates, than there are domains defined in an MLN. We made the design choice 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. This definition enables the merging of Knowledge Categories, resulting in clusters in the knowledge base that can be converted into an MLN for querying or further learning. Figure 3 illustrates example Knowledge Categories. In the Figure, the category 3 contains a subset of domains of category 1 and thus should be merged, irrespective of the difference in predicates. In the same figure, category 4 shares domains with category 1 and 2. The domains of category 4 intersect with those of category 1 and 2. Category 4 cannot be merged into category 1 as it is unclear how D_4 relates to the domains D_1, D_2 and D_3 in category 1. The same reasoning hold for merging category 4 into 2.

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. $C_1(\{F_1, w_1, z_1\}, D_1) \cup C_2(\{F_2, w_2, z_2\}, D_2), D1 \subseteq D2 \rightarrow C_2(\{F_1, w_1, z_1\} \cup \{F_2, w_2, z_2\}, D_2)$.

The Knowledge Category merging operation consists of two operations: a domain set union and merging the knowledge triplets from one category into the other. During the knowledge triplet merging operation, each formula in the triplets is checked for whether it is new to the category or it is the same as an existing formula in the category. In the case that a formula is new, i.e. it is not the exactly same as an existing one, its triplet is simply added to the category.

In the case that 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 are 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.

Step 1 – 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. The constants the domains of both categories range over, are assigned to the corresponding domains in the Knowledge List.

Step 2 – To solve formula weight conflicts, 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 formula weight merging conflicts: the *CL-Naive*, *CL-Conservative* and *CL-Balanced* strategies. The pseudocode for each strategy is shown in Table 2.

Table 2. Pseudocode of three knowledge updating strategies. w is the weight of a formula, z is the evidence count for the formula.

CL-Naive	CL-Conservative	CL-Balanced
$w_1 \leftarrow w_2$	$if(z_2 > z_1):$	$if(z_1 = z_2 = 0):$
$z_1 \leftarrow z_2$	$w_1 \leftarrow w_2$	$w1 \leftarrow \frac{w_1+w_2}{2}$
	$z_1 \leftarrow z_2$	$else: w_1 \leftarrow \frac{z_1 w_1 + z_2 w_2}{z_1+z_2},\ z_1 \leftarrow z_1 + z_2$

CL Naive – The simplest possible strategy is to prefer any new knowledge over the old knowledge. This is represented in the CL-Naive strategy, described in Table 2. 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 weights was trained on. That the weights are 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.

CL Conservative – A more conservative strategy 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 Table 2. In this strategy the evidence count portion of the knowledge triplets is important. The evidence count of a formula simply represents the amount of evidence seen during the training of the weight of the formula. Only the evidence 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 to force new formula weights to contain more information than the previous weights.

CL Balanced – The CL-Balanced strategy, described in Table 2, 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.

3.3 Learning process

In the first step an MLN and an evidence database are given as initial input to create a knowledge base. 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 using the standard Alchemy algorithm [9]. If any evidence is unknown only formula weight learning [30] is performed. The structure learning threshold can be adjusted according to user preference.

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. In this final merging process the knowledge updating strategies are applied to solve conflicts between old and new formula weights for all duplicate formulas. For the cumulative learning of an MLN, new evidence consists of one or a combination of: new formulas (from an MLN), new evidence (from a database), or new predicate declarations that define new relations over either existing or new domains of constants (from either an MLN or database).

3.4 Cumulative Learning Algorithm

As indicated in Figure 2, MLN-CLA allows for two types of sources of new evidence: either from an MLN or an evidence database. At each time step a combination of any type of evidence can be presented to the cumulative learner to incorporate. The Alchemy software [31] used for MLN learning 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 evidence predicates 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 from the evidence databases such as `IsA(Cat, Animal)`, indicating by argument index 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. MLN-CLA employs the Knowledge List concept to manage a knowledge base. First-order logic formulas are clustered together within the Knowledge List in Knowledge Categories. These categories can be converted to standalone MLNs to update weights, add new formulas or modify the network structure. After the update, the MLN is converted back to a standalone Knowledge List. Each standalone Knowledge List is then merged into the original Knowledge List using a knowledge updating strategy to solve conflicts. Subsequently, the algorithm is ready to accept new information.

4 EXPERIMENTS

We conducted two experiments with MLN-CLA to test the learning capacities in different scenarios. In the first experiment the MLN model is shown new constants (objects such as `Horse` or `Small_boat`) in each learning step whilst keeping the model formulas unchanged. In the second experiment a new formula is introduced to the model in each learning step. In both experiments the three knowledge updating strategies CL-Naive, CL-Conservative and CL-Balanced are tested against two baseline methods.

The MLN model used in the experiments is based on Zhu et al. [32]. The MLN consists of five formulas as shown in Table 3. The '+' symbol indicates that a separate formula weight is learned for each constant that the variable ranges over. The resulting trained model contains a formula with learned weight for each combination of possible constants of the variables preceded by '+' in the untrained MLN. The formulas 1-4 relate an object property to an affordance. These formulas represent a zero-shot approach to object affordance inference as objects are not learned by their label but from their properties [33]. Formula 5 relates object categories (such as `Writing_implement` and `Animal`) to each other. This formula acts as supporting information for formula 1. It is a Markov logic representation of an ontology for object categories. The constants and formulas used for training each model are based on the training set of the FOL affordance dataset by Zhu et al. [32].

Table 3. MLN model for object affordance inference based on Zhu et al [32].

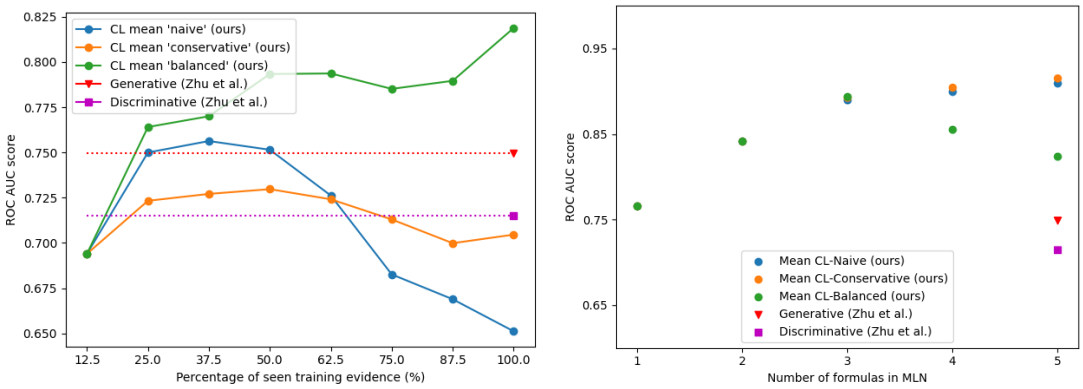
Index	Formula
1	`IsA(obj,+category)` ⇒ `HasAffordance(obj,+affordance)`
2	`HasVisualAttribute(obj,+attribute)` ⇒ `HasAffordance(obj,+affordance)`
3	`HasWeight(obj,+weight)` ⇒ `HasAffordance(obj,+affordance)`
4	`HasSize(obj,+size)` ⇒ `HasAffordance(obj,+affordance)`
5	`IsA(obj,+category)` ⇒ `IsA(obj,+category)`

The training set evidence consists of evidence for 5 predicates and 40 different objects, such as `HasAffordance(Banjo, Play)`, `HasVisualAttribute(Banjo, Wood)`, and `HasWeight(Cat, W1)` where `W1` is a constant representing the weight class < 1 kg. The model performance of each learning step of each experiment is evaluated on the test dataset by Zhu et al. [32]. The test set consists of the evidence for the same 5 predicates for 22 objects. The test objects are similar to the training set but not the same.

In each experiment, the trained MLN at the end of a learning step is queried for the `HasAffordance` predicate given all test evidence. The model outputs the predicted marginal probability – between 0 (false), 0.5 (unknown) and 1 (true) – that an object has a certain affordance. To evaluate the performance we adopt the Area under the ROC curve (AUC) metric, similar to Richardson and Domingos [7]. In each experiment the baseline performance is set by the standard weight learning algorithms in Alchemy called the *generative* [7] and *discriminatively* [30] learned MLN models, both trained on the whole training set including all formulas.

4.1 Learning new constants

In Figure 4a the average performance of the constants learning experiment is shown over 300 runs. In each run the order of the evidence fed to MLN-CLA was randomised. We expected MLN-CLA to approach but not outperform the baseline batch trained MLNs, indicated in Figures 4a-4b with *Generative* [7] and *Discriminative* [30]. Initially the MLN-CLA variants do perform similar to the baseline MLNs. However, from 50% of the object constants evidence seen, CL-Naive and CL-Conservative decline in performance or drop below the baselines. Surprisingly, the CL-Balanced variant outperforms the baselines from the second learning step. We hypothesise that the CL-Balanced strategy acts as a regulariser on the weights. A regulariser affects the weights by making formulas that have a larger impact on performance, e.g. `HasWeight(object, Heavy)` ⇒ `HasAffordance(object, Lift)`, have larger weights (either negative or positive). It pushes the weights for less important formulas, e.g. `HasSize(obj, Small)` ⇒ `HasAffordance(obj, Row)`, to zero.



(a) Learning new `object` constants. (b) Learning new formulas.

Fig. 4. Learning of new constants and new formulas.

4.2 Learning new formulas

In Figure 4b the average performance of the formulas learning experiment is shown over 5 runs. In each run the order of formulas learned is different to ensure that the results are invariant to evidence order. In this experiment the cumulative learner learns a new predicate in each step. However, the test set contains evidence of predicates not yet known to the cumulative learner. As a consequence, MLNs trained on only a subset of the predicates present in the evidence cannot be evaluated on the full test set. We circumvented this problem by only evaluating on the test evidence of which the predicates are known to the incremental learning. Consequently, the results in Figure 4b are only comparable within one step and not between steps. This is indicated in the figure by the lack of a connecting line between each data point. In the figure, the MLN-CLA variants have overlapping performance for the first three learning steps. In the fourth

and fifth learning steps the MLN-CLA variants showcase their differences. The CL-Conservative outperforms both the baselines and the other variants after having seen all five formulas. In the same learning step, CL-Balanced performs the worst of the three MLN-CLA variants, although it still outperforms the baselines.

Similar to the constants experiment, we expected MLN-CLA to approach the performance of the baseline batch trained MLNs. Curiously, MLN-CLA outperforms the baselines for each variant after learning all five formulas. Structure learning seems to play a large role in the formula learning scenario. Structure learning is applied each time a new formula consists of a set of domains that form a subset of a previously seen formula. In this experiment the formulas 1 and 5 from Table 3 share domains. Interestingly, in Table 4 the CL-Balanced strategy dips in performance after structure learning when learning the 1st formula after having seen the 5th formula in the previous steps. The same behaviour is not observed for the other two strategies. We hypothesise that this behaviour is caused by the CL-Balanced strategy adjusting formula weights directly where the other strategies do not. The baseline MLNs are not structure learned. Structure learning whilst learning thus improves performance for the CL-Naive and CL-Conservative strategies but not for the CL-Balanced strategy.

Table 4. The AUC score (10^{-2}) of each step of all runs in the cumulative learning of formulas experiment for each strategy CL-Naive (N), CL-Conservative (C) and CL-Balanced (B). In each run the order of formulas learned is different. 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* 75, *Discriminative* 72.

	N	C	B	N	C	B	N	C	B	N	C	B	N	C	B
Step		**1**			**2**			**3**			**4**			**5**	
Run 1	88	88	88	94	94	94	92	92	92	90	90	90	93	93	91
Run 2	80	80	80	81	81	81	93	93	93	92	91	88	93	93	91
Run 3	93	93	93	94	94	94	90	91	91	94	93	92	93	93	91
Run 4	50	50	50	80	80	80	81	81	81	89	89	89	90	91	*71*
Run 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**

4.3 Knowledge updating strategies

In Figures 4a and 4b the knowledge updating strategies show different behaviours under different scenarios. In the constant learning setting the CL-Balanced strategy performs best, whereas in the formula learning setting it performs worst of the three strategies. There seems to be a trade-off between when each strategy can be applied best. The differences between the strategies are highlighted by Figure 5. Comparison of strategies is made difficult by the fact that the effects on formula weights by each strategy in earlier step propagate to subsequent steps. In the figure the predicted marginal probabilities for nine object-affordance pairs over the eight learning steps of the constants learning experiment are shown. Each knowledge updating strategy reacts differently to new evidence. CL-Naive adjusts weights every time new evidence comes in. A good example of this behaviour is shown in the top-left in Figure 5. The CL-Conservative strategy is a less erratic version of CL-Naive. CL-Balanced is even less erratic and introduces a delay to a flip in predictions such as shown in the pair Camel-Grasp. In summary, there is no one strategy that performs best. However, the cumulative learning strategies do outperform the standard MLN batch learning algorithms [7, 30]. This observation invites more research into the knowledge updating strategies. Other strategies than the three presented here are possible.

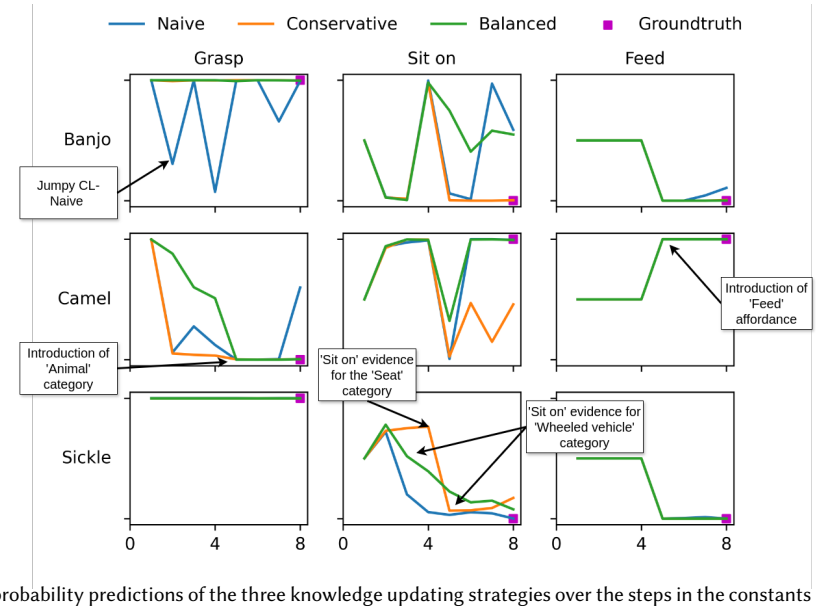


Fig. 5. Marginal probability predictions of the three knowledge updating strategies over the steps in the constants learning experiment.

5 CONCLUSIONS

We introduced MLN-CLA for the incremental learning of new knowledge and applied our algorithm to an affordance learning scenario. To prevent knowledge from being lost during learning we additionally introduced three knowledge updating strategies. Each strategy solves knowledge merging conflicts in a different manner based on evidence counts. Within MLN-CLA the strategies can easily be modified.

Our experiments show the effectiveness of MLN-CLA in adapting to new evidence. In addition, the experiments highlight the differences and trade-offs between the choice of knowledge updating strategy to apply. The CL-Balanced strategy performs best overall when only learning new constants. The same strategy performs worst of the three whilst still outperforming the batch learned baselines.

The knowledge updating strategies we formulated rely heavily on the evidence count (except for CL-Naive). In our experiments, we did not vary the batch sizes of new evidence per time step. The impact of varying batch sizes over time on performance and system stability should be further investigated. Our current implementation of MLN-CLA uses the default Alchemy weight and structure learning algorithms. An interesting exercise would be to replace those by the Online Structure Learning algorithm to improve learning performance.

REFERENCES

[1] James J. Gibson. *The Ecological Approach to Visual Perception.* Boston: Houghton Mifflin, 1979.
[2] Mihai Andries, Ricardo Omar Chavez-Garcia, Raja Chatila, Alessandro Giusti, and Luca Maria Gambardella. Affordance equivalences in robotics: A formalism. *Frontiers in Neurorobotics*, 12, 2018.
[3] Paola Ardón, Èric Pairet, Katrin S Lohan, Subramanian Ramamoorthy, and Ronald Petrick. Affordances in robotic tasks–a survey. *arXiv preprint arXiv:2004.07400*, 2020.
[4] Michael Ahn, Anthony Brohan, Noah Brown, Yevgen Chebotar, Omar Cortes, Byron David, Chelsea Finn, Chuyuan Fu, Keerthana Gopalakrishnan, Karol Hausman, et al. Do as i can, not as i say: Grounding language in robotic affordances. *arXiv preprint arXiv:2204.01691*, 2022.
[5] Harmen van den Berg. First-order logic in knowledge graphs. *Current Issues in Mathematical Linguistics*, 56:319–328, 1993.
[6] Vaishak Belle. Statistical relational learning and neuro-symbolic ai: what does first-order logic offer? *arXiv preprint arXiv:2306.13660*, 2023.
[7] Matthew Richardson and Pedro Domingos. Markov logic networks. *Machine Learning*, 62:107–136, 2006. https://doi.org/10.1007/s10994-006-5833-1.
[8] Pedro Domingos and Daniel Lowd. Unifying logical and statistical ai with markov logic. *Commun. ACM*, 62(7):74–83, jun 2019.

[9] Stanley Kok and Pedro Domingos. Learning the structure of markov logic networks. In *Proceedings of the 22nd international conference on Machine learning*, pages 441–448, 2005.

[10] Shai Ben-David, Eyal Kushilevitz, and Yishay Mansour. Online learning versus offline learning. *Machine Learning*, 29(1):45–63, Oct 1997.

[11] Shan Cui, Tao Zhu, Xiao Zhang, and Huansheng Ning. MCLA: Research on cumulative learning of Markov Logic Network. *Knowledge-Based Systems*, 242:108352, 2022.

[12] Kristinn R. Thórisson, Jordi Bieger, Xiang Li, and Pei Wang. Cumulative learning. In Patrick Hammer, Pulin Agrawal, Ben Goertzel, and Matthew Iklé, editors, *Artificial General Intelligence*, pages 198–208, Cham, 2019. Springer International Publishing.

[13] Robert M. French. Catastrophic forgetting in connectionist networks. *Trends in Cognitive Sciences*, 3(4):128–135, 1999.

[14] Tuyen N. Huynh and Raymond J. Mooney. Online structure learning for markov logic networks. In Dimitrios Gunopulos, Thomas Hofmann, Donato Malerba, and Michalis Vazirgiannis, editors, *Machine Learning and Knowledge Discovery in Databases*, pages 81–96, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[15] Alexandre Bernardino Luis Montesano, Manuel Lopes and Jose Santos-Victor. Modeling affordances using bayesian networks. *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4102–4107, 2007.

[16] Luis Montesano, Manuel Lopes, Alexandre Bernardino, and JosÉ Santos-Victor. Learning object affordances: From sensory–motor coordination to imitation. *IEEE Transactions on Robotics*, 24(1):15–26, 2008.

[17] Manuel Lopes, Francisco S. Melo, and Luis Montesano. Affordance-based imitation learning in robots. In *2007 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1015–1021, 2007.

[18] Esteban Jaramillo-Cabrera, Eduardo F Morales, and Jose Martinez-Carranza. Enhancing object, action, and effect recognition using probabilistic affordances. *Adaptive Behavior*, 27(5):295–306, 2019.

[19] Avi Pfeffer. A bayesian language for cumulative learning. In *Proceedings of AAAI 2000 Workshop on Learning Statististical Models from Relational Data*, 2000.

[20] Kun Yue, Qiyu Fang, Xiaoling Wang, Jin Li, and Weiyi Liu. A parallel and incremental approach for data-intensive learning of bayesian networks. *IEEE Transactions on Cybernetics*, 45(12):2890–2904, 2015.

[21] Weiyi Liu, Kun Yue, Mingliang Yue, Zidu Yin, and Binbin Zhang. A bayesian network-based approach for incremental learning of uncertain knowledge. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 26(01):87–108, 2018.

[22] Haoran Liu, Zhaoyu SU, Yongii Liu, Livue Zhang, Rongrong Yin, and Zhang Ying. An improved incremental structure learning algorithm for bayesian networks. In *2019 6th International Conference on Systems and Informatics (ICSAI)*, pages 505–510, 2019.

[23] Abhishek Kumar, Sunabha Chatterjee, and Piyush Rai. Bayesian structural adaptation for continual learning. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 5850–5860. PMLR, 18–24 Jul 2021.

[24] Artur d'Avila Garcez and Luis C Lamb. Neurosymbolic AI: The 3rd wave. *Artificial Intelligence Review*, pages 1–20, 2023.

[25] Feiyu Xu, Hans Uszkoreit, Yangzhou Du, Wei Fan, Dongyan Zhao, and Jun 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*, pages 563–574. Springer, 2019.

[26] Rudresh Dwivedi, Devam Dave, Het Naik, Smiti Singhal, Rana Omer, Pankesh Patel, Bin Qian, Zhenyu Wen, Tejal Shah, Graham Morgan, et al. Explainable AI (XAI): Core ideas, techniques, and solutions. *ACM Computing Surveys*, 55(9):1–33, 2023.

[27] Daphne Koller, Nir Friedman, Lise Getoor, and Ben Taskar. Graphical Models in a Nutshell. In *Introduction to Statistical Relational Learning*. The MIT Press, 08 2007.

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

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

[30] Parag Singla and Pedro Domingos. Discriminative training of markov logic networks. In *Proceedings of the 20th National Conference on Artificial Intelligence - Volume 2*, AAAI'05, page 868–873. AAAI Press, 2005.

[31] Stanley Kok, Parag Singla, M. Richardson, and Pedro Domingos. The Alchemy system for statistical relational AI. Technical report, Department of Computer Science and Engineering, University of Washington, Seattle, WA, 2005.

[32] Yuke Zhu, Alireza Fathi, and Li Fei-Fei. Reasoning about object affordances in a knowledge base representation. In David Fleet, Tomas Pajdla, Bernt Schiele, and Tinne Tuytelaars, editors, *Computer Vision – ECCV 2014*, pages 408–424, Cham, 2014. Springer International Publishing.

[33] Yongqin Xian, Christoph H Lampert, Bernt Schiele, and Zeynep Akata. Zero-shot learning—a comprehensive evaluation of the good, the bad and the ugly. *IEEE transactions on pattern analysis and machine intelligence*, 41(9):2251–2265, 2018.