Parallel Neurosymbolic Integration with Concordia

Jonathan Feldstein [*1,2] **Modestas Jurčius** [*1,3] **Efthymia Tsamoura** [4]

Abstract

Parallel neurosymbolic architectures have been applied effectively in NLP by distilling knowledge from a logic theory into a deep model. However, prior art faces several limitations including supporting restricted forms of logic theories and relying on the assumption of independence between the logic and the deep network. We present Concordia, a framework overcoming the limitations of prior art. Concordia is agnostic both to the deep network and the logic theory offering support for a wide range of probabilistic theories. Our framework can support supervised training of both components and unsupervised training of the neural component. Concordia has been successfully applied to tasks beyond NLP and data classification, improving the accuracy of state-of-the-art on collective activity detection, entity linking and recommendation tasks.

1. Introduction

Motivation. To overcome the limitations of deep networks, such as dependence on significant amount of labelled training data, researchers proposed to integrate logical theories, a computational paradigm known as *neurosymbolic AI* (d'Avila Garcez et al., 2002). One way to integrate the components is in a staged or *stratified* fashion. Stratified neurosymbolic frameworks find applications in problems admitting well-separable symbolic and subsymbolic tasks – to name a toy example, performing mathematical operations using symbolic models over MNIST digits identified by a neural model. One of the first stratified architectures was DeepProbLog (Manhaeve et al., 2018). More followed: NeurASP (Yang et al., 2020), ABL (Dai et al., 2019), RNMs (Marra et al., 2020) and NeuroLog (Tsamoura et al., 2021).

[*]Equal contribution [1]University of Edinburgh, Edinburgh, United Kingdom [2]BENNU.AI, Edinburgh, United Kingdom [3]Mintis AI, Kaunas, Lithuania [4]Samsung AI, Cambridge, United Kingdom. Correspondence to: Jonathan Feldstein <jonathan.feldstein@bennu.ai>.

Proceedings of the 40^{th} International Conference on Machine Learning, Honolulu, Hawaii, USA. PMLR 202, 2023.

An alternative to stratified is *parallel* integration. In contrast to stratified frameworks, parallel integration applies in settings in which the same task can be solved both symbolically and sub-symbolically and the aim is to increase the accuracy of the end task by distilling knowledge from the logic component into the neural one and vice versa. Two parallel neurosymbolic frameworks have been proposed recently: Teacher-Student (T-S) by Hu et al. (Hu et al., 2016a;b) and Deep Probabilistic Logic (DPL) by Wang and Poon (Wang & Poon, 2018). T-S is based on posterior regularization (Ganchev et al., 2010) to build a teacher network which is later used to train the neural module. DPL defines a joint distribution after making the assumption of independence between the logical theory and the deep network and uses this distribution to regularize the deep model.

Problem. Stratified approaches focus on separating the pattern recognition and the reasoning to allow for more complex reasoning and querying. However, the approaches named above tend to suffer from high computational complexity. In this work, we focus in particular on how neurosymbolic AI can help to reduce the need of (labelled) data. To this end, we focus on parallel approaches which are more tailored to this problem as they enhance the neural model. However, both T-S and DPL come with several limitations. Firstly, the adopted formulations do not support settings in which the inputs and the targets abide by more complex relations. In particular, they only support rules expressing constraints directly relating the input with the output data and do not support recursive formulas. However, in practical scenarios, the relationships between the inputs and the outputs may be modelled only via richer logical formulas that reference *latent* information, i.e., information that is not available either in the input or output data. Additionally, with regards to DPL, the integration is based on the assumption of independence between the two components, which generally does not hold as they both depend upon the same input data. Finally, DPL is bound to Markov Logic Networks (MLNs) (Richardson & Domingos, 2006) limiting its applicability to classification only.

Contribution. We present Concordia[1], a parallel neurosymbolic framework that overcomes the above limitations. Concordia adopts probabilistic logics due to their flexibility to

[1]Available on https://github.com/jonathanfeldstein/Concordia

reason in a formal fashion over uncertain data (De Raedt & Kimmig, 2015). Concordia relies on the theoretically sound inference and training techniques of probabilistic logics to train the two components in a supervised or unsupervised fashion. Its interface supports theories expressed as weighted formulas in first-order logic, including lifted graphical models like MLNs and Probabilistic Soft Logic (PSL) (Bach et al., 2017) allowing for easy integration of domain expertise. Offering a plug-and-play interface is crucial as each language comes with its own semantics and inference/learning footprint. Furthermore, our framework employs a mixture of experts technique (Jacobs et al., 1991) to integrate the two components without relying on the independence assumption, and it supports using the deep network predictions as priors. Our empirical findings support the hypothesis that integrating symbolic models with neural ones does indeed reduce the need for (labelled) data. Finally, as we empirically demonstrate, learning the weights of the formulas at training time provides the means to learn the formulas of the theories themselves and not just their weights. In fact, we show that if we throw arbitrary formulas at training time, then the weights of the non-useful ones will drop to zero.

Results Concordia is the first framework of its kind to be applied beyond NLP and, in particular, on formulas that prior art in parallel approaches are incapable of supporting. Our evaluation shows that even simple commonsense formulas can have a significant impact on the accuracy of the end-task. In particular, Concordia leads to NLP models with up to 5.8% higher accuracy than those of DPL and to activity recognition models with up to 6.75% and 3.87% higher accuracy than the neurosymbolic techniques from (London et al., 2013) and (Kuang & Tie, 2020), respectively. It also improves the root mean squared error on recommendation (Kouki et al., 2015) by up to 2.10%. These improvements increase even further with less data.

2. Related work

Stratified integration As reported in (Tsamoura et al., 2021), most existing stratified frameworks suffer from limited scalablity especially in the presence of recursion. Furthermore, the ones that employ heuristics to reduce the reasoning overhead, such as ABL (Dai et al., 2019), suffer from limited accuracy. The above limitations restrict the applicability of frameworks such as (Manhaeve et al., 2018; Dai et al., 2019; Yang et al., 2020) to real-world settings. Concordia supports theories that relevant prior art fails to support, see Section 5 for further details.

Parallel integration Above, we focused on several limitations of T-S and DPL, including inability to express complex relationships that govern the input and the output data. Hence, their applicability is restricted to simple tasks. With regards to T-S, there is an additional limitation: the optimization objective that is used to build the teacher model does not abide by the semantics of Probabilistic Soft Logic (Bach et al., 2017). In particular, in the optimal solution found, when building a teacher, the slack variables (the ξ's in (Hu et al., 2016a)) should approach 0 so that the expectation of the rule satisfaction becomes 1. However, in that case, the weight of each constraint is not taken into account.

Knowledge distillation Our work is relevant to the broader area of knowledge distillation (Mobahi et al., 2020; Dao et al., 2021; Hinton et al., 2015). There, the objective is to distill knowledge from a complex deep model into a simpler one. Our work substantially differs from the above line of research. **(i)** Our teacher is a (probabilistic) logical theory and not a neural model. Therefore, Concordia allows integrating into a deep model prior knowledge in symbolic form, making introduction of domain expertise very simple. **(ii)** At a higher level, prior art on purely neural teacher-student techniques aims to simplify a complex deep model. In contrast, we aim to improve the accuracy of a neural model using prior knowledge. **(iii)** In the above line of research, the accuracy of the student will not outperform that of the teacher. In contrast, our analysis shows our symbolic teacher can lead to neural models with higher accuracy than on its own.

Regularization LTNs (Donadello et al., 2017), DL (van Krieken et al., 2020) and DASL (Sikka et al., 2020) introduce stratified techniques for training deep models using the semantics of fuzzy logic, however, without training the logic theory. pLogicNet (Qu & Tang, 2019) applies Markov Logic Networks to learn knowledge graph embeddings. Xie et al. (2019) propose a technique to project fixed propositional (variable-free) formulas onto a manifold via graph convolutional networks. The weights of the formulas are not jointly trained with the deep model. In (Fischer et al., 2019) and in (Li & Srikumar, 2019), the authors propose frameworks where the neural models are regularised using symbolic constraints, by asserting whether the neural model satisfies *one fixed* constraint. The above means that their loss is a SAT function and the constraints are unweighted.

Beyond the difference about supporting theories rather than single constraints, Concordia distills the knowledge captured in the full probability distribution, rather than just the max of the logical component ($\mathcal{L}$) into the neural component ($\mathcal{N}$). Hence, we use more of the knowledge produced by $\mathcal{L}$ – our loss is a comparison of distributions. The experiments in Section 5 show that this additional knowledge is particularly helpful in settings with limited data.

3. Preliminaries

First-order logic A *term* is either a *variable* or a *constant*. An *atom* α is an expression of the form $p(\mathbf{t})$, where p is a *predicate* and $\mathbf{t}$ is a vector of terms; α is *ground*, if $\mathbf{t}$ includes only constants. We refer to expressions in first-order logic as *formulas*. *Rules* are universally quantified formulas of the form $\alpha_1 \wedge \cdots \wedge \alpha_N \rightarrow \alpha$, where α_i's and α are atoms and each term occurring in α also occurs in some α_i. We refer to the left-hand and right-hand sides of the implication as the *premise* and the *conclusion* of the rule, respectively. A formula is *instantiated* if each atom occurring in the formula is ground. A *theory* is a collection of formulas. Assuming a set of constants, we refer to the set of all possible ground atoms computed by instantiating the formulas in the theory as its *Herbrand base*. An *interpretation* is a mapping of the ground atoms in the Herbrand base to a truth value. In the classical Boolean logic, each ground atom is mapped to true or false. Other logic formalisms map ground atoms to $[0, 1]$.

Example 1. *Consider the rule from (Kouki et al., 2015):*

$$\textsc{Similar}(I_1, I_2) \wedge \textsc{Rates}(U, I_1) \rightarrow \textsc{Rates}(U, I_2) \quad (1)$$

According to the rule, if items I_1 and I_2 are similar, then user U will rate item I_1. In this example, I_1, I_2, and U are variables. Neither T-S nor DPL can support rule (2) due to their inability to handle transitive rules. An instantiation of rule (2), where `T1`*,* `T2`*, and* `A` *stand for* `ToyStory1`*,* `ToyStory2`*, and* `Alice`*, respectively, could be*

$$\textsc{Similar}(\texttt{T1}, \texttt{T2}) \wedge \textsc{Rates}(\texttt{A}, \texttt{T1}) \rightarrow \textsc{Rates}(\texttt{A}, \texttt{T2}) \quad (2)$$

Mappings A *substitution* σ is a total mapping from variables to constants. For a vector $\mathbf{X}$, $\sigma(\mathbf{X})$ is obtained by replacing each occurrence of $X \in \mathbf{X}$ in the domain of σ with $\sigma(X)$.

Markov Random Fields A *Markov Random Field* (MRF) is a model for the joint distribution of a set of random variables (RVs) $\mathbf{X} = (X_1, \ldots, X_N)$. It is composed of an undirected graph G and a set of *potentials* $\phi_1, \ldots, \phi_M$, where each ϕ_i is usually represented by an indicator function f_i weighted by λ_i. Graph G has a node for each variable, while the MRF has a potential ϕ_i for each clique of variables $\mathbf{X}_i$ in G. Potential ϕ_i maps each instantiation of $\mathbf{X}_i$ to a non-negative real value. The joint distribution represented by an MRF is given by:

$$P(\mathbf{X} = \mathbf{x}) = \frac{1}{Z} \exp\left(\sum_{i=1}^{M} \lambda_i f_i(\mathbf{X}_i = \mathbf{x}_i)\right), \quad (3)$$

where Z is a normalization constant and $\mathbf{x}_i$ denotes an instantiation of $\mathbf{X}_i$ as per $\mathbf{x}$, i.e., assuming $\mathbf{x} = \sigma(\mathbf{X})$, for a substitution σ, then $\mathbf{x}_i = \sigma(\mathbf{X}_i)$. We will see examples of f_i's later in the section.

We denote the marginal distribution of the subset $\mathbf{Y}$ of $\mathbf{X}$ by $MARG(\mathbf{Y} = \mathbf{y})$ and compute conditional probabilities using the Bayes rule. Let $\mathbf{X}^o$ denote a tuple of *observed* RVs, i.e., RVs with known values, and $\mathbf{X}^u$ denoted a tuple of *unobserved* RVs, i.e., RVs with unknown values. The *Most Probable Explanation* (MPE) or *Maximum A Posteriori State* (MAP) is the most likely assignment to the variables in $\mathbf{X}^u$ given $\mathbf{X}^o$:

$$\begin{aligned} &MPE(\mathbf{X}^u = \mathbf{x}^u \mid \mathbf{X}^o = \mathbf{x}^o) \\ &= \arg\max_{\mathbf{x}^u} P(\mathbf{X}^u = \mathbf{x}^u \mid \mathbf{X}^o = \mathbf{x}^o) \end{aligned} \quad (4)$$

LGMs To establish the connection between a theory in first order logic $\mathcal{L}$ and probability theory, *lifted graphical models* (LGMs) treat each ground atom in the Herbrand base of $\mathcal{L}$ as a random variable (RV) with the same domain as the one of the atom. Each formula r_i in $\mathcal{L}$ serves as a template for defining potentials $\phi_{i,1}, \ldots, \phi_{i,M_i}$. Those potentials share the same weights λ_i (as we shall later see, this is going to be the weight of r_i) and map each tuple of constants $\mathbf{x}$ to the same value. To summarize, an LGM is a set of weighted formulas $\lambda_i :: r_i$, for $1 \leq i \leq M$, where each r_i defines a set of potentials $\phi_{i,1}, \ldots, \phi_{i,M_i}$ in a MRF with $\phi_{i,j} = \phi_{i,k}$, for each $j, k \in \{1, \ldots, M_i\}$. The log-linear representation of LGMs defines the joint distribution

$$P(\mathbf{X} = \mathbf{x}) = \frac{1}{Z} \exp\left(\sum_{i=1}^{M} \sum_{j=1}^{M_i} \lambda_i f_{i,j}(\mathbf{X}_{i,j} = \mathbf{x}_{i,j})\right),$$

where Z is the normalization constant and $\mathbf{x}_{i,j}$ is defined analogously to (3).

Probabilistic Soft Logic (PSL) (Bach et al., 2017) is an example of an LGM. PSL adopts the semantics of fuzzy logic to interpret the formulas and the Lukasiewicz t-(co)norms to compute the truth values of the instantiated formulas. Due to the adoption of fuzzy logic, interpretations map atoms to soft truth values in $[0, 1]$. Returning back to Example 1, the truth value of atom SIMILAR(`ToyStory1`,`ToyStory2`) is in $[0, 1]$ in PSL. That allows us to incorporate uncertainty to the level of similarity between elements, something that is not possible with Boolean first-order logic. The $f_{i,j}$ functions are given by

$$f_{i,j}(\mathbf{X}_{i,j} = \mathbf{x}_{i,j}) = (1 - r_i(\mathbf{X}_{i,j} = \mathbf{x}_{i,j}))^p \quad (5)$$

where $r_i(\mathbf{x}_{i,j})$ denotes the truth value of formula r_i when instantiated using the ground atoms $\mathbf{x}_{i,j}$ and $p \in \{1, 2\}$ provides a choice of penalty. If rule r_i cannot be instantiated using $\mathbf{x}_{i,j}$, then $r_i(\mathbf{x}_{i,j}) = 0$.

4. Combining logic with neural networks

This section presents the main contribution of this work, the Concordia framework. An overview of the framework is presented in Figure 1.

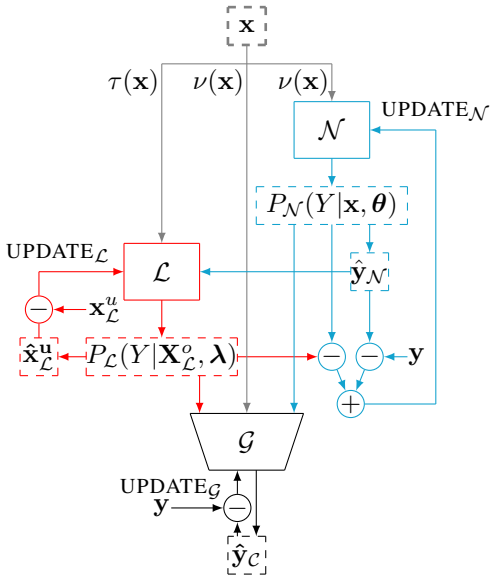

Figure 1. Overview of the Concordia architecture. Grey components are the input, blue components are the neural component, red components are the logic component, and the components parts are the gating network.

Let $\mathcal{D}$ denote the set of training data. Each training datum is denoted by $(\mathbf{x}, \mathbf{y})$, where $\mathbf{x} \in \mathcal{X}$ and $\mathbf{y} \in \mathcal{Y}$. Concordia uses each $(\mathbf{x}, \mathbf{y})$ to train the neural component, the logical theory, and a gating network, denoted by $\mathcal{N}$, $\mathcal{L}$ and $\mathcal{G}$, respectively. We first present the $\mathcal{N}$ and $\mathcal{L}$ components and then proceed with the description of the overall architecture.

4.1. The logical component

The first building block of our architecture is an LGM $\mathcal{L}$. We denote the vector of all formulas' weights as $\boldsymbol{\lambda}$. Treating each ground atom in the Herbrand base of $\mathcal{L}$ as a RV, module $\mathcal{L}$ defines a probability distribution $P_{\mathcal{L}}(\mathbf{X}_{\mathcal{L}}|\boldsymbol{\lambda})$, see Section 3.

Component $\mathcal{L}$ should infer information for each possible target in $\mathcal{Y}$. This implies that the Herbrand base of $\mathcal{L}$ needs to include a ground atom α_j denoting the truth of each target class $c_j \in \mathcal{Y}$. We refer to atoms α_j as *target atoms*. If $\mathcal{Y}$ is a continuous space, as in regression tasks, the adoption of PSL allows us to use rules so that the Herbrand base of $\mathcal{L}$ includes a single ground atom per target.

The training data defines instantiations of the RVs in $\mathbf{X}_{\mathcal{L}}$. In particular, $\tau(\mathbf{x})$ and $\tau(\mathbf{xy})$ are vectors of (partial) truth assignments to the elements in $\mathbf{X}_{\mathcal{L}}$. Logic allows us to train under *uncertain data* by assigning values to the atoms in the range [0,1]. By abusing the notation, we can also treat the τ-vectors as mappings of atoms to their truth values. We use $\mathbf{X}^o_{\mathcal{L}}$ to denote the vectors of RVs that are instantiated (observed) given the vectors in $\tau(\mathbf{x})$ or $\tau(\mathbf{xy})$; we use $\mathbf{X}^u_{\mathcal{L}}$ to denote the RVs that are left non-instantiated.

Example 2. *Reconsider rule* (1)*, where we now use* λ *to denote its weight. Each datum* $(\mathbf{x}, \mathbf{y})$ *is a user-item pair. In particular, vector* $\tau(\mathbf{xy})$ *includes the relational representation of users and items, e.g., if* $(\mathbf{x}, \mathbf{y})$ *is the rating, say 3, of user* `Alice` *on the movie* `Toy Story`*, then the vector* $\tau(\mathbf{xy})$ *assigns to atom* RATES(`Alice`, `Toy Story`) *the rating 3 normalized into* $[0, 1]$ *(that is 0.5).*

Inference Inference proceeds by means of a conditional distribution $P_{\mathcal{L}}(Y|\mathbf{X}^o_{\mathcal{L}}, \boldsymbol{\lambda})$, where Y is a RV with domain $\mathcal{Y}$, denoting the likelihood of a target for given observations (instantiations to the observed variables $\mathbf{X}^o_{\mathcal{L}}$) and formula weights $\boldsymbol{\lambda}$. Continuing with Example 2, assuming our goal is to predict the rating of `Alice` on `Toy Story`, then RATES(`Alice`, `Toy Story`) is the single target atom and $P_{\mathcal{L}}(Y|\mathbf{X}^o_{\mathcal{L}}, \boldsymbol{\lambda})$ denotes the conditional distribution over the values of the target.

To define $P_{\mathcal{L}}(Y|\mathbf{X}^o_{\mathcal{L}}, \boldsymbol{\lambda})$ in classification tasks, we have two options depending on the logic semantics. When the logic admits Boolean interpretations, i.e., an atom is either true or false with some probability as in Markov Logic Networks (Richardson & Domingos, 2006), then we can compute the likelihood of each α_j being true using marginals and arrange the computed likelihoods into distributions over the classes.

In contrast, when the logic interpretations map each ground atom to $[0, 1]$, $P_{\mathcal{L}}(Y = c_j|\mathbf{X}^o_{\mathcal{L}} = \tau(\mathbf{x}), \boldsymbol{\lambda})$ is the soft truth value of the atom α_j in the interpretation $\tau(\mathbf{x})\hat{\mathbf{x}}^u_{\mathcal{L}}$, where

$$\hat{\mathbf{x}}^u_{\mathcal{L}} = \arg\max_{\mathbf{x}^u_{\mathcal{L}}} P_{\mathcal{L}}(\mathbf{X}^u_{\mathcal{L}} = \mathbf{x}^u_{\mathcal{L}}|\mathbf{X}^o_{\mathcal{L}} = \tau(\mathbf{x}), \boldsymbol{\lambda}) \qquad (6)$$

Above, $\hat{\mathbf{x}}^u_{\mathcal{L}}$ denotes the most likely assignment of the unobserved variables given the input data and the current weights and $\tau(\mathbf{x})\hat{\mathbf{x}}^u_{\mathcal{L}}$ denotes the assignment of the atoms in the Herbrand base of $\mathcal{L}$ by taking the union of the assignments in $\tau(\mathbf{x})$ and $\hat{\mathbf{x}}^u_{\mathcal{L}}$. The discussion on inference for regression tasks is deferred to Appendix D.

Constraints When $\mathcal{L}$ admits Boolean interpretations, we require it to impose mutual exclusiveness constraints in the elements in $\mathcal{Y}$, i.e., that the elements in $\mathcal{Y}$ cannot be simultaneously true. When $\mathcal{L}$ admits interpretations in $[0, 1]$, then we require $\mathcal{L}$ to impose the constraint that the sum of the soft truth values of the α_j's in each interpretation equals to 1. Both constraints ensure that the distribution over classes is a valid one, i.e., the sum over all possible outcomes is 1.

Training Training aims to learn the weights $\boldsymbol{\lambda}$. The task is formalized as finding the $\boldsymbol{\lambda}$ maximizing the log likelihood of the assignments $\tau(\mathbf{xy})$:

$$\hat{\boldsymbol{\lambda}} = \arg\max_{\boldsymbol{\lambda}} \prod_{(\mathbf{x},\mathbf{y})\in\mathcal{D}} P_{\mathcal{L}}(\mathbf{X}_{\mathcal{L}} = \tau(\mathbf{xy}), \boldsymbol{\lambda}) \qquad (7)$$

Equation (7) works when $\tau(\mathbf{xy})$ provides truth assignments to all variables in $\mathbf{X}_{\mathcal{L}}$. If that assumption is violated, training resorts to an expectation-maximization problem. Parameter learning in lifted graphical models as well as probabilistic logic programs works via gradient ascent (Richardson & Domingos, 2006; Bach et al., 2017). We use $\text{UPDATE}_{\mathcal{L}}(\mathbf{x}, \mathbf{y}, \tau, \boldsymbol{\lambda}_t) \rightarrow \boldsymbol{\lambda}_{t+1}$ to denote updating of the parameters in $\mathcal{L}$ given $(\mathbf{x}, \mathbf{y})$.

4.2. The neural component

Given a neural model $\mathcal{N}$, then, for a fixed assignment to the neural weights $\boldsymbol{\theta}$, $\mathcal{N}$ defines a conditional distribution $P_{\mathcal{N}}(Y|\mathbf{X}_{\mathcal{N}}, \boldsymbol{\theta})$ by using a softmax output layer producing a $|\mathcal{Y}|$-dimensional prediction vector. Then, $P_{\mathcal{N}}(Y = c_j|\mathbf{X}_{\mathcal{N}}, \boldsymbol{\theta})$ defines the likelihood the label is class $c_j \in \mathcal{Y}$.

Similarly to the previous section, we have to specify the semantics of the training data. We denote by $\nu(\mathbf{x})$ the vector representation of $\mathbf{x}$ in the input format of $\mathcal{N}$.

Inference For fixed $\boldsymbol{\theta}$ and $\nu(\mathbf{x})$, $\mathcal{N}$ returns prediction $\hat{\mathbf{y}}$:

$$\hat{\mathbf{y}} = \arg\max_{\mathbf{y}} P_{\mathcal{N}}(Y = \mathbf{y}|\mathbf{X}_{\mathcal{N}} = \nu(\mathbf{x}), \boldsymbol{\theta}) \tag{8}$$

We use $\text{INFER}_{\mathcal{N}}(\mathbf{x}, \nu, \boldsymbol{\theta}) \rightarrow \hat{\mathbf{y}}$ to denote inference in $\mathcal{N}$.

Training Training is achieved by backpropagation, as standard in deep networks, via a loss function measuring the discrepancy between the neural predictions and the true label.

4.3. Integration of components

The integration of the two components in Concordia is based on a gating network $\mathcal{G}$ with learnable parameters $\boldsymbol{\gamma}$. In our implementation, the input domain of $\mathcal{G}$ is the same as of $\mathcal{N}$. The output domain is $[0, 1]$. The idea is that depending on the data the benefit provided by either model can vary. Prior art discards the logical model entirely at the end of the training, or simply multiplies the two distributions under the assumption that the two distributions are independent. Our approach is more flexible and takes advantage of the logical model during testing.

Inference Inference is described in Algorithm 1. For each input datatum $\mathbf{x}$, we first provide $\nu(\mathbf{x})$ and $\tau(\mathbf{x})$ to $\mathcal{N}$ and $\mathcal{L}$, to get the distributions $P_{\mathcal{N}}(Y|\mathbf{X}_{\mathcal{N}} = \nu(\mathbf{x}), \boldsymbol{\theta})$ and $P_{\mathcal{L}}(Y|\mathbf{X}^o_{\mathcal{L}} = \tau(\mathbf{x}), \boldsymbol{\lambda})$. Then, we combine those distributions using $\mathcal{G}$. In particular, Concordia defines the following conditional probability distribution:

$$\begin{aligned} &P_{\mathcal{C}}(Y|\mathbf{X}_{\mathcal{N}}, \mathbf{X}^o_{\mathcal{L}}, \boldsymbol{\theta}, \boldsymbol{\lambda}, \boldsymbol{\gamma}) \\ &= \mathcal{G}(\mathbf{X}_{\mathcal{N}}, \boldsymbol{\gamma}) P_{\mathcal{N}}(Y|\mathbf{X}_{\mathcal{N}}, \boldsymbol{\theta}) \\ &+ (1 - \mathcal{G}(\mathbf{X}_{\mathcal{N}}, \boldsymbol{\gamma})) P_{\mathcal{L}}(Y|\mathbf{X}^o_{\mathcal{L}}, \boldsymbol{\lambda}) \end{aligned} \tag{9}$$

Prediction $\hat{\mathbf{y}}$ given input $\mathbf{x}$ and $\boldsymbol{\theta}$, $\boldsymbol{\lambda}$, $\boldsymbol{\gamma}$ is then computed as

$$\hat{\mathbf{y}} = \arg\max_{\mathbf{y}} P_{\mathcal{C}}(Y = \mathbf{y}|\mathbf{X}_{\mathcal{N}} = \nu(\mathbf{x}), \mathbf{X}^o_{\mathcal{L}} = \tau(\mathbf{x}), \boldsymbol{\theta}, \boldsymbol{\lambda}, \boldsymbol{\gamma}) \tag{10}$$

Notice that above formulation is substantially different from the ones proposed in T-S and DPL: T-S uses posterior regularization; DPL is based on multiplying the predictions.

Forward inference in Concordia is denoted via $\text{INFER}_{\mathcal{C}}(\mathbf{x}, \nu, \tau, \boldsymbol{\theta}, \boldsymbol{\lambda}, \boldsymbol{\gamma}) \rightarrow \hat{\mathbf{y}}$.

Algorithm 1 $\text{INFER}_{\mathcal{C}}(\mathbf{x}, \nu, \tau, \boldsymbol{\theta}, \boldsymbol{\lambda}, \boldsymbol{\gamma})$

$\Delta_{\mathcal{N}}(Y) \leftarrow P_{\mathcal{N}}(Y|\mathbf{X}_{\mathcal{N}} = \nu(\mathbf{x}), \boldsymbol{\theta})$
$\Delta_{\mathcal{L}}(Y) \leftarrow P_{\mathcal{L}}(Y|\mathbf{X}^o_{\mathcal{L}} = \tau(\mathbf{x}), \boldsymbol{\lambda})$
$\kappa \leftarrow \mathcal{G}(\mathbf{X}_{\mathcal{N}}, \boldsymbol{\gamma})$
$\Delta_{\mathcal{C}}(Y) \leftarrow \kappa\Delta_{\mathcal{N}}(Y) + (1 - \kappa)\Delta_{\mathcal{L}}(Y)$
return $\arg\max_{\mathbf{y}} \Delta_{\mathcal{C}}(Y = \mathbf{y})$

Training Parameter update in Concordia is summarized in Algorithm 2. For each $(\mathbf{x}, \mathbf{y})$ training proceeds as follows. Firstly, $\mathcal{L}$ is trained based on $(\mathbf{x}, \mathbf{y})$ and the interface $\text{UPDATE}_{\mathcal{L}}$ as described in Section 4.1. Secondly, $\mathcal{N}$ is trained to minimize the difference between its predictions and the true labels $\mathbf{y}$, as well as to minimize the difference between distributions $P_{\mathcal{N}}(Y|\mathbf{X}_{\mathcal{N}} = \nu(\mathbf{x}), \boldsymbol{\theta})$ and $P_{\mathcal{L}}(Y|\mathbf{X}^o_{\mathcal{L}} = \tau(\mathbf{x}), \boldsymbol{\lambda})$. The first term aims to keep $\mathcal{N}$'s predictions close to the ground truth. The second term aims to keep $\mathcal{N}$'s predictions close to the ones of $\mathcal{L}$. Hence $\mathcal{L}$ supervises $\mathcal{N}$ in a weak fashion during the training process. The neural weights get updated at each step t as follows:

$$\begin{aligned} &\boldsymbol{\theta}_{t+1} = \arg\min_{\boldsymbol{\theta}} \ell(\hat{\mathbf{y}}, \mathbf{y}) + \\ &KL(P_{\mathcal{N}}(Y|\mathbf{X}_{\mathcal{N}} = \nu(\mathbf{x}), \boldsymbol{\theta}_t)||P_{\mathcal{L}}(Y|\mathbf{X}^o_{\mathcal{L}} = \tau(\mathbf{x}), \boldsymbol{\lambda}_t)) \end{aligned} \tag{11}$$

Above, $\hat{\mathbf{y}} = \text{INFER}_{\mathcal{N}}(\mathbf{x}, \nu, \boldsymbol{\theta})$ and KL denotes the Kullback–Leibler divergence between distributions. The $\mathcal{N}$'s parameter update process we described above is denoted via $\text{UPDATE}_{\mathcal{N}}(\mathbf{x}, \mathbf{y}, \nu, \boldsymbol{\theta}_t) \rightarrow \boldsymbol{\theta}_{t+1}$.

Lastly, parameters $\boldsymbol{\gamma}$ of $\mathcal{G}$ are amended so that the labels predicted by the whole framework $\hat{\mathbf{y}}_{\mathcal{C}}$ fit the ground truth:

$$\boldsymbol{\gamma}_{t+1} = \arg\min_{\boldsymbol{\gamma}} \ell(\mathbf{y}, \hat{\mathbf{y}}_{\mathcal{C}}) \tag{12}$$

Above, $\hat{\mathbf{y}}_{\mathcal{C}} = \text{INFER}_{\mathcal{C}}(\mathbf{x}, \nu, \tau, \boldsymbol{\theta}, \boldsymbol{\lambda}, \boldsymbol{\gamma})$.

Algorithm 2 $\text{UPDATE}_{\mathcal{C}}(\mathbf{x}, \mathbf{y}, \nu, \tau, \boldsymbol{\theta}_t, \boldsymbol{\lambda}_t, \boldsymbol{\gamma}_t)$

$\boldsymbol{\theta}_{t+1} \leftarrow \text{UPDATE}_{\mathcal{N}}(\mathbf{x}, \mathbf{y}, \nu, \boldsymbol{\theta}_t)$
$\boldsymbol{\lambda}_{t+1} \leftarrow \text{UPDATE}_{\mathcal{L}}(\mathbf{x}, \mathbf{y}, \tau, \boldsymbol{\lambda}_t)$
$\hat{\mathbf{y}}_{\mathcal{C}} \leftarrow \text{INFER}_{\mathcal{C}}(\mathbf{x}, \boldsymbol{\theta}_t, \boldsymbol{\lambda}_t, \boldsymbol{\gamma}_t)$
$\boldsymbol{\gamma}_{t+1} = \arg\min_{\boldsymbol{\gamma}} \ell(\mathbf{y}, \hat{\mathbf{y}}_{\mathcal{C}})$
return $(\boldsymbol{\theta}_{t+1}, \boldsymbol{\lambda}_{t+1}, \boldsymbol{\gamma}_{t+1})$

Unsupervised learning So far, we discussed supervised learning. Unsupervised learning is supported by discarding the first term in (11) and the gating network. Notice that in the unsupervised setting, we do not train $\mathcal{L}$. Training of $\mathcal{L}$ in an unsupervised setting is left for future work.

Multitasking Above, the explanations focused on a single target atom. However, the architecture is easily extendable to multitasking. To this end, we consider instead of a single RV Y a vector of RVs $\mathbf{Y}$ and the probability of $\mathcal{N}$ becomes $P_{\mathcal{N}}(\mathbf{Y} = \mathbf{y}|\mathbf{X}_{\mathcal{N}} = \nu(\mathbf{x}), \boldsymbol{\theta})$. This allows on one hand to achieve different tasks in parallel and on the other to optimize predictions concurrently. For example, in the case of recommendation above, users and items are dependent on each other and need to be optimized concurrently. Multitasking is not supported by prior art.

4.4. Neural predictions as priors

We can use the neural predictions as weak supervision signal to train the logical component. Adding neural priors can be achieved by considering the neural predictions as additional observed RVs in $\mathcal{L}$. The above is always possible via two steps: introducing additional rules propagating information from $\mathcal{N}$ to $\mathcal{L}$, and assigning to the atoms denoting the neural predictions the confidences predicted by $\mathcal{N}$.

Example 3. *We demonstrate propagating information from $\mathcal{N}$ to $\mathcal{L}$ we extend Example 2: To introduce the rating predictions of $\mathcal{N}$ as priors to $\mathcal{L}$, we firstly add*

$$\lambda : \text{DNN}(U, I) \rightarrow \text{RATES}(U, I)$$

Atom $\text{DNN}(U, I)$ *represents ratings of user U on item I as predicted by the deep network.*

Algorithm 3 formalizes the creation of variable instantiations based on the neural predictions. After adding additional rules to $\mathcal{L}$ as in Example 3, its Herbrand base includes a ground atom DNN_j associated with the likelihood to which $\mathcal{N}$ predicts the class c_j. We refer to $\text{DNN}j$ as the *neural atom* for c_j in the Herbrand base of $\mathcal{L}$. Algorithm 3 simply assigns to $\text{DNN}j$ the likelihood $P_{\mathcal{N}}(Y = c_j|\mathbf{X}_{\mathcal{N}} = \nu(\mathbf{x}), \boldsymbol{\theta})$.

To incorporate neural priors in Concordia's inference or learning process, we simply need to update the instantiations of $\mathbf{X}^o_{\mathcal{L}}$ from $\tau(\mathbf{x})$ or $\tau(\mathbf{xy})$ to $\tau(\mathbf{x})\mathbf{z}$ and $\tau(\mathbf{xy})\mathbf{z}$, where $\mathbf{z}$ denotes the computed instantiations (Alg. 3).

Algorithm 3 $\text{TRANSLATE}(\mathbf{x}, \nu, \boldsymbol{\theta}, \mathcal{L}) \rightarrow \mathbf{z}$

```
z ← ∅ {z maps atoms to their truth values.}
for each c_j ∈ 𝒴 do
    DNN_j denotes the neural atom for c_j in the Herbrand base of ℒ
    set the value of DNN_j in z to P_𝒩(Y = c_j | X_𝒩 = ν(x), θ)
end for
return z
```

5. Experiments

Scenarios We consider scenarios that have been adopted by prior symbolic and neurosymbolic techniques: collective activity detection (CAD) (Wu et al., 2019; Kuang & Tie, 2020; London et al., 2013), recommendation (Kouki et al., 2015) and entity linking (Wang & Poon, 2018). To ensure a fair comparison, we adopt the same datasets and baselines with those techniques for each case. Each scenario includes a neural component $\mathcal{N}$ and a logical component $\mathcal{L}$. $\mathcal{L}$ is implemented in PSL for each scenario. We denote by $\mathcal{C}(\mathcal{N}, \mathcal{L})$ the Concordia instantiations. Further details on the experimental setup and training parameters can be found in Appendix B.

Baselines We consider the following baselines:

- **Recommendation.** We compare against (i) the deep models **`NNMF`** (Dziugaite & Roy, 2015), **`NeuMF`** (He et al., 2017b) and **`GraphRec`** (Rashed et al., 2019); and (ii) the purely symbolic technique from (Kouki et al., 2015), which performs the task using PSL. This dataset is used to test Concordia on regression tasks.

- **CAD.** We compare against (i) the state-of-the-art models from (Wu et al., 2019) that use **`MobileNet`** and **`Inception-v3`** as backbone networks; (ii) the state-of-the-art neurosymbolic technique **`IARG`**[2] (Kuang & Tie, 2020) that is based on the above deep architectures, denoted as `IARG(MobileNet)` and `IARG(Inception-v3)`, respectively; and (iii) the stratified neurosymbolic technique from (London et al., 2013). This dataset is used to test Concordia on complex classification.

- **Entity linking.** We adopt the same experimental setting with (Wang & Poon, 2018) and compare against (i) **`BiLSTM`**, a Bi-LSTM recurrent neural network proposed in (Peng et al., 2017); (ii) **`DPL`**; (iii) the symbolic theory adopted by the `DPL` authors alone; and (iv) **`DistilBERT`** (Sanh et al., 2019) a distilled version of BERT, a pre-trained transformer model originally proposed in (Devlin et al., 2018). Below, we denote `DistilBERT` by `BERT` for readability purposes. This dataset is used to have a direct comparison with the closest competitor in the literature, as well as testing Concordia's performance in unsupervised and semi-supervised settings.

[2]The authors adopt a Graph Convolutional Network (GCN) and extend on (Wu et al., 2019) with a more sophisticated message passing algorithm.

Other neuro-symbolic comparisons Neither T-S nor DPL supports the theories adopted in CAD, due to latent variables, and recommendations, due to recursiveness, and, hence, they are not considered in those scenarios. In the appendix, we provide further details on the rules that are not supported by T-S and DPL.

Furthermore, **TensorLog** (Cohen et al., 2020) only supports rules of a specific transitive form and therefore cannot be used in our experiments, while techniques such as Neural Theorem Proving (Minervini et al., 2018) are not relevant as they do not concern knowledge distillation, but question answering in Prolog. Based on discussions with the authors of **DeepProbLog** (Manhaeve et al., 2018), a comparison cannot be performed either: firstly, facts cannot be updated in ProbLog, secondly, it does not support learning across time sequences. Both limitations make DeepProbLog inapplicable to our scenarios. Additionally, the recommendation task has a very large number of entities in the knowledge base that can increase the training overhead of DeepProbLog to the extend that training is prohibitively slow (DeepProbLog faces significant scalability restrictions as observed in (Tsamoura et al., 2021)). Neurosymbolic frameworks like **NeuroLog** (Tsamoura et al., 2021) and **ABL** (Dai et al., 2019) that rely on the same principles with DeepProbLog cannot be applied for the same reasons. Finally, regarding **DL2** (Fischer et al., 2019) and (Li & Srikumar, 2019), beyond the limitations mentioned in Section 2 including the fact that these frameworks only support a single fixed constraint, they do not support passing latent variables. Furthermore, (Li & Srikumar, 2019) expects the rules to be of a specific acyclic form. For the above reasons, these two frameworks do not support the experiments in this section.

Learning paradigms. Learning proceeds in a supervised fashion in the first two scenarios and in an unsupervised and semi-supervised fashion in the last one. To assess the robustness of Concordia to the input theory, we consider a large set of noisy rules (i.e., rules that do not contribute to the task) with unknown weights and learn the weights of the rules at training time, in the last scenario.

5.1. Item recommendation

Given a user and his ratings, we aim to determine the user's prospective rating on an item. We used the 2020 Yelp and MovieLens-100k datasets (Harper & Konstan, 2015) and ran the setups $\mathcal{C}(\texttt{NNMF}, \mathcal{L}_{\texttt{REC}})$, $\mathcal{C}(\texttt{NeuMF}, \mathcal{L}_{\texttt{REC}})$ as well as $\mathcal{C}(\texttt{GraphRec}, \mathcal{L}_{\texttt{REC}})$ on MovieLens-100k, where $\mathcal{L}_{\texttt{REC}}$ is the PSL theory from (Kouki et al., 2015).

Results Figure 2 shows the RMSE for different percentages of training data used (100% is equivalent to the whole dataset). We measured performance in terms of Root Mean Squared Error (RMSE). The results are shown for $\mathcal{L}_{\texttt{REC}}$

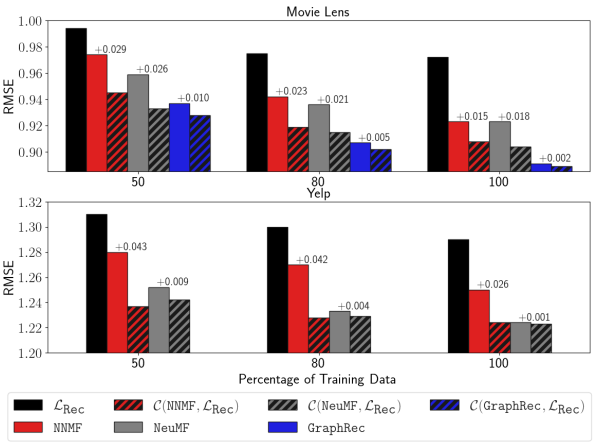

Figure 2. Results on a recommendation task on MovieLens (top) and Yelp (bottom).

being trained independently of the neural components. Concordia can substantially improve the performance of the respective deep models in most cases. For instance, when the full MovieLens-100k dataset is used the RMSE drops from 0.923 to 0.908 with regards to `NNMF`; with regards to `NeuMF`, the RMSE drops from 0.923 to 0.904 and even on `GraphRec`, a GNN designed specifically for recommendation systems, we improve from 0.891 to 0.889. We tested our hypothesis that the symbolic reasoning will increase even further when less data is available and found this to hold across models. In addition, we can see that except for `GraphRec` the Concordia models with 80% data outperform their purely neural models with 100%, and with 50% data outperform the neural models with 80% data. This shows that independent of the model, introducing symbolic rules further helps the neural models and reduces the need for data. The same behaviour can be observed on the Yelp dataset. `GraphRec` was not capable of performing training on this dataset due to its size. This shows that while GNNs can help in a similar fashion to symbolic models, as they represent relations well, they cannot be used in larger datasets where Concordia is still able to perform on other neural models.

5.2. Collective activity detection

CAD asks to identify the activity taking place by a group of actors in a collection of video frames (Ibrahim & Mori, 2018; Qi et al., 2018). We used the dataset from (Choi et al., 2011) and the train/test splits from (Qi et al., 2018). We denote by $\mathcal{C}(\texttt{MobileNet}, \mathcal{L}_{\texttt{CAD}})$ and $\mathcal{C}(\texttt{Inception-v3}, \mathcal{L}_{\texttt{CAD}})$ the two different Concordia setups, where $\mathcal{L}_{\texttt{CAD}}$ is taken from (London et al., 2013).

Results Table 1 reports the average, best and worst accuracy over five different training/testing runs. Accuracy

is defined as the percentage of the correctly predicted labels for group activity. In contrast to what was reported in (Kuang & Tie, 2020) and despite that we used their code base, IARG was less effective than MobileNet and Inception-v3 alone in most cases. Table 1 shows that $\mathcal{C}(\texttt{MobileNet}, \mathcal{L}_{\texttt{CAD}})$ improves over MobileNet both in terms of best and average accuracy. In particular, the average accuracy improves from 90.07% to 90.73%, while the best one improves from 91.36% to 93.19%. With regards to Inception-v3, the accuracy improvements brought by Concordia are even more significant. In particular, the average accuracy improves from 89.72% to 92.75%, the best accuracy from 91.83% to 93.34% and the worst accuracy from 86.84% to 92.31%. Notice that these results are state-of-the-art in CAD outperforming the ones from (Wu et al., 2019; Kuang & Tie, 2020). For completeness, Table 1 also copies the results obtained when using $\mathcal{L}_{\texttt{CAD}}$ in a stratified fashion over action context descriptors (ACD) (Lan et al., 2012) for solving the task (ACD + $\mathcal{L}_{\texttt{CAD}}$). We also ran the models on different dataset sizes by taking 50%, 80% and 100% of the training data to test our hypothesis that the fewer data we have the larger the impact. Similarly to the recommendation task, we found that to hold across models, where on MobileNet the addition of symbolic knowledge increase from 0.67% improvement in accuracy to 1.27%, while on the Inception-v3 model the improvement was from 3.03% for 100% data to 4.14% improvement on 50% of the data.

Table 1. Results on CAD.

Model	AA (%)	BA (%)	WA (%)
ACD+$\mathcal{L}_{\texttt{CAD}}$	86.00	-	-
MNet	90.07	91.36	89.61
IARG(MNet)	90.18	92.39	87.55
$\mathcal{C}(\texttt{MNet}, \mathcal{L}_{\texttt{CAD}})$	90.73	93.19	89.54
Incept	89.72	91.83	86.84
IARG(Incept)	88.88	91.67	85.33
$\mathcal{C}(\texttt{Incept}, \mathcal{L}_{\texttt{CAD}})$	**92.75**	**93.34**	**92.31**

Table 2. Results on CAD over different dataset sizes.

Model	50 %	80 %	90%
MNet	68.70	77.1	90.07
$\mathcal{C}(\texttt{MNet}, \mathcal{L}_{\texttt{CAD}})$	69.97	78.29	90.73
Difference	1.27	1.19	0.67
Incept	71.10	84.55	89.72
$\mathcal{C}(\texttt{Incept}, \mathcal{L}_{\texttt{CAD}})$	**75.24**	**88.59**	**92.75**
Difference	4.14	4.04	3.03

Table 3. Results on entity linking.

Model	F_1	Pr	Rec	Acc (%)
$\mathcal{L}_{\texttt{EL}}$ (u)	0.77	0.80	0.75	74.7
DPL(BiLSTM) (u)	0.76	0.68	**0.86**	70.0
$\mathcal{C}(\texttt{BiLSTM}, \mathcal{L}_{\texttt{EL}})$ (u)	**0.78**	**0.84**	0.72	**75.8**
$\mathcal{L}_{\texttt{EL}}$ (sp)	0.76	0.85	0.69	75.2
BiLSTM (sp)	0.74	0.58	**1.00**	58.5
$\mathcal{C}(\texttt{BiLSTM}, \mathcal{L}_{\texttt{EL}})$ (sm)	0.82	0.91	0.74	**80.1**
BERT (sp)	0.88	**0.99**	0.78	88.5
$\mathcal{C}(\texttt{BERT}, \mathcal{L}_{\texttt{EL}})$ (sm)	**0.91**	**0.99**	0.81	**91.4**

5.3. Entity linking

To compare Concordia to DPL (Wang & Poon, 2018), we used the PubMedParsed dataset from (Moen & Ananiadou, 2013) and its extension from (Wang & Poon, 2018). We initially used the setup $\mathcal{C}(\texttt{BiLSTM}, \mathcal{L}_{\texttt{EL}})$, where $\mathcal{L}_{\texttt{EL}}$ is taken from (Wang & Poon, 2018) and is encoded in PSL to have a direct comparison using the neural model used by the DPL authors. Then, we also used the setup $\mathcal{C}(\texttt{BERT}, \mathcal{L}_{\texttt{EL}})$ to compare to more modern NLP models and understand how Concordia performs compared to large pre-trained neural models and to test whether Concordia only improves on simple neural models or also on large complex neural models. Firstly, as in (Wang & Poon, 2018), we trained Concordia in an unsupervised fashion using only unlabelled data. Secondly, we used half of the labelled data to train Concordia in a semi-supervised fashion: in each epoch, we firstly trained the BiLSTM , BERT and $\mathcal{L}_{\texttt{EL}}$ in a supervised fashion (using the labelled data) and then trained BiLSTM and BERT in an unsupervised fashion (using the unlabelled data). In the unsupervised case, all rule weights were set to 1. In the semi-supervised one, the rules were trained on half of the labelled data in advance. In addition, we also considered BiLSTM and BERT alone trained in a supervised fashion on the labelled data. We also considered $\mathcal{L}_{\texttt{EL}}$ alone in two settings. In the unsupervised setting, $\mathcal{L}_{\texttt{EL}}$ performs predictions using rules which are all weighted 1. In the supervised case, $\mathcal{L}_{\texttt{EL}}$ is trained alone (i.e., independently of Concordia and the neural component) using the labelled data. The final baseline is BiLSTM regularized via DPL in an unsupervised fashion, denoted by DPL(BiLSTM). All models were trained across five different data folds.

Results Table 3 reports the results of our experiments: (u) stands for unsupervised, (sm) for semi-supervised and (sp) for supervised learning. For DPL, we copied the results from (Wang & Poon, 2018).

DPL uses priors in their factors that were set based on information extracted from the labelled data– that contradicts the statement in their paper that the training was unsupervised. In contrast, we do not provide these priors to Concordia

setting the weights of the rules in $\mathcal{L}_{\texttt{EL}}$ to 1. Still, even with rules with untrained weights, Concordia outperforms DPL by 5.8% in accuracy and by 0.02 in F_1. Concordia in an entirely unsupervised case (Table 3 row 3) outperforms `BiLSTM` (Table 3 row 5) trained in a supervised fashion on the small labelled data set by 17.3%. Our semi-supervised experiment outperforms the supervised `BiLSTM` by 21.6% and even `BERT` by 2.9%, while the F_1 score improves by 0.08 and 0.03 respectively, proving again our hypothesis that Concordia can help significantly in settings with small amount of labelled data.

DPL uses Markov Networks (and not MLNs) and takes, according to the authors, 2.5h to train in a cluster with 25 CPU cores and 1 GPU. Due to approximation techniques of probabilistic logical solvers, which have worst case polynomial time complexity (Bach et al., 2017; Richardson & Domingos, 2006), Concordia is much more time-efficient taking under 6min to train on 1 GPU and 4 CPU cores.

6. Discussion and Conclusions

Theory learning. Concordia assumes that the formulas of the theory are given. Our assumption is along the lines of the assumptions made by several other neurosymbolic techniques such as (Manhaeve et al., 2018; Yang et al., 2020; Tsamoura et al., 2021), as well as T-S and DPL. Despite that this assumption might be considered as restrictive, one could argue that purely neural teacher-student architectures (Mobahi et al., 2020; Dao et al., 2021), as well as deep learning techniques in general, also assume prior knowledge. In the former case, the prior knowledge is encoded into the teacher in subsymbolic form, while in the latter case, prior knowledge is encoded into the deep architecture. One option to circumvent this shortcoming if no rules are available is to use structure learning approaches (Feldstein et al., 2023; Kok & Domingos, 2010; Khot et al., 2015), these are frameworks that aim at learning logical formulae from data and then pass these formulae into the logical model. However, weight learning can also serve as the means to learn the structure of the rules as previously shown in (Qu & Tang, 2019) in the context of learning knowledge graph embeddings: we can simply inject many arbitrary rules into the model (e.g., in the worst-case all possible rules of a specific form[3]) and then at training time, the weights of the useless rules will drop to 0. In fact, this phenomenon was observed in our entity linking scenario. There, most rules turned out to be useless and their weights dropped to 0.

Expressiveness. We focus on LGMs as they offer a good balance between expressivity and complexity. Probabilistic frameworks such as ProbLog have been criticized for their high inference overhead (Aditya et al., 2019). This overhead has been the reason for which prior neurosymbolic techniques such as (Zhu et al., 2014) have resorted to LGMs. Notice that LGMs support (ground) recursive rules as they are based on first-order-logic.

Beyond LGMs. We presented an instantiation of Concordia based on LGMs (see Section 3). However, Concordia does not make any assumptions on the logical theory, as long as it abides by the generic interface presented here, see Section 4. Beyond lifted graphical models, Concordia, can also support languages like ProbLog. Similarly to MLNs and PSL, ProbLog treats the ground atoms in the theory as RVs. Secondly, ProbLog supports both MAP (or MPE) inference (Fierens et al., 2015) and supervised parameter learning (Gutmann et al., 2008) and hence can implement Concordia's interfaces $P_{\mathcal{L}}(Y|\mathbf{X}^o_{\mathcal{L}}, \boldsymbol{\lambda})$ and $\text{UPDATE}_{\mathcal{L}}(\mathbf{x}, \mathbf{y}, \tau, \boldsymbol{\lambda}_t) \rightarrow \boldsymbol{\lambda}_{t+1}$. We leave the integration of ProbLog as part of future work.

Conclusions. We presented Concordia, a parallel neurosymbolic framework that is based on the formal semantics of probabilistic logical theories. Concordia can significantly improve the accuracy of deep models by injecting into them knowledge represented by probabilistic logic theories, leading to state-of-the-art results in a variety of tasks. Future work includes applying Concordia to other tasks, learning the structure of the rules at training time, and training the logical component in unsupervised settings.

[3]This is similar to inductive logic programming (Evans & Grefenstette, 2018), where users specify patterns of rules to learn.

References

Aditya, S., Yang, Y., and Baral, C. Integrating knowledge and reasoning in image understanding. In *IJCAI*, pp. 6252–6259, 2019.

Bach, S. H., Broecheler, M., Huang, B., and Getoor, L. Hinge-loss markov random fields and probabilistic soft logic. *Journal of Machine Learning Research*, 18:109:1–109:67, 2017.

Choi, W., Shahid, K., and Savarese, S. Learning context for collective activity recognition. In *CVPR*, pp. 3273–3280. IEEE, 2011.

Cohen, W. W., Yang, F., and Mazaitis, K. Tensorlog: A probabilistic database implemented using deep-learning infrastructure. *J. Artif. Intell. Res.*, 67:285–325, 2020. doi: 10.1613/jair.1.11944.

Dai, W.-Z., Xu, Q., Yu, Y., and Zhou, Z.-H. Bridging Machine Learning and Logical Reasoning by Abductive Learning. In *NeurIPS*, pp. 2815–2826, 2019.

Dao, T., Kamath, G. M., Syrgkanis, V., and Mackey, L. Knowledge distillation as semiparametric inference. In *ICLR*, 2021.

d'Avila Garcez, A. S., Broda, K., and Gabbay, D. M. *Neural-symbolic learning systems: foundations and applications*. Perspectives in neural computing. Springer, 2002.

De Raedt, L. and Kimmig, A. Probabilistic (logic) programming concepts. *Machine Learning*, 100(1):5–47, 2015.

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

Donadello, I., Serafini, L., and d'Avila Garcez, A. S. Logic tensor networks for semantic image interpretation. In *IJCAI*, pp. 1596–1602, 2017.

Dziugaite, G. K. and Roy, D. M. Neural network matrix factorization. *arXiv preprint arXiv:1511.06443*, 2015.

Evans, R. and Grefenstette, E. Learning explanatory rules from noisy data. *J. Artif. Intell. Res.*, 61:1–64, 2018. doi: 10.1613/jair.5714.

Feldstein, J., Phillips, D., and Tsamoura, E. Principled and efficient motif finding for structure learning in lifted graphical models. *arXiv preprint arXiv:2302.04599*, 2023.

Fierens, D., Van den Broeck, G., Renkens, J., Shterionov, D. S., Gutmann, B., Thon, I., Janssens, G., and De Raedt, L. Inference and learning in probabilistic logic programs using weighted boolean formulas. *Theory and Practice of Logic Programming (TPLP)*, 15(3):358–401, 2015.

Fischer, M., Balunovic, M., Drachsler-Cohen, D., Gehr, T., Zhang, C., and Vechev, M. Dl2: training and querying neural networks with logic. In *International Conference on Machine Learning*, pp. 1931–1941. PMLR, 2019.

Ganchev, K., Graça, J. a., Gillenwater, J., and Taskar, B. Posterior regularization for structured latent variable models. *Journal of Machine Learning Research*, 11:2001–2049, 2010.

Gutmann, B., Kimmig, A., Kersting, K., and De Raedt, L. Parameter learning in probabilistic databases: A least squares approach. In *Machine Learning and Knowledge Discovery in Databases*, pp. 473–488, 2008.

Harper, F. M. and Konstan, J. A. The movielens datasets: History and context. *ACM Transactions on Interactive Intelligent Systems*, 5(4), 2015.

He, K., Gkioxari, G., Dollár, P., and Girshick, R. B. Mask R-CNN. In *ICCV*, pp. 2980–2988, 2017a.

He, X., Liao, L., Zhang, H., Nie, L., Hu, X., and Chua, T.-S. Neural collaborative filtering. In *WWW*, pp. 173–182, 2017b.

Hinton, G. E., Vinyals, O., and Dean, J. Distilling the knowledge in a neural network. *CoRR*, abs/1503.02531, 2015.

Hu, Z., Ma, X., Liu, Z., Hovy, E., and Xing, E. Harnessing deep neural networks with logic rules. In *ACL*, pp. 2410–2420, 2016a.

Hu, Z., Yang, Z., Salakhutdinov, R., and Xing, E. Deep neural networks with massive learned knowledge. In *EMNLP*, pp. 1670–1679, 2016b.

Ibrahim, M. S. and Mori, G. Hierarchical relational networks for group activity recognition and retrieval. In *ECCV*, pp. 742–758, 2018.

Jacobs, R. A., Jordan, M. I., Nowlan, S. J., and Hinton, G. E. Adaptive mixtures of local experts. *Neural computation*, 3(1):79–87, 1991.

Khot, T., Natarajan, S., Kersting, K., and Shavlik, J. Gradient-based boosting for statistical relational learning: the markov logic network and missing data cases. *Machine Learning*, 100(1):75–100, 2015.

Kok, S. and Domingos, P. M. Learning markov logic networks using structural motifs. In *ICML*, volume 10, pp. 551–558, 2010.

Koren, Y., Bell, R., and Volinsky, C. Matrix factorization techniques for recommender systems. *Computer*, 42(8): 30–37, 2009.

Kouki, P., Fakhraei, S., Foulds, J., Eirinaki, M., and Getoor, L. Hyper: A flexible and extensible probabilistic framework for hybrid recommender systems. In *RecSys*, pp. 99–106, 2015.

Kuang, Z. and Tie, X. Video understanding based on human action and group activity recognition. *CoRR*, abs/2010.12968, 2020.

Lan, T., Wang, Y., Mori, G., and Robinovitch, S. N. Retrieving actions in group contexts. In *ECCV Workshops*, pp. 181–194, 2012.

Li, T. and Srikumar, V. Augmenting neural networks with first-order logic. *arXiv preprint arXiv:1906.06298*, 2019.

London, B., Khamis, S., Bach, S. H., Huang, B., Getoor, L., and Davis, L. Collective activity detection using hinge-loss markov random fields. In *CVPR Workshops*, pp. 566–571, 2013.

Manhaeve, R., Dumancic, S., Kimmig, A., Demeester, T., and De Raedt, L. Deepproblog: Neural probabilistic logic programming. In *NeurIPS*, pp. 3749–3759, 2018.

Marra, G., Diligenti, M., Giannini, F., Gori, M., and Maggini, M. Relational neural machines. In *ECAI*, volume 325, pp. 1340–1347, 2020.

Minervini, P., Bosnjak, M., Rocktäschel, T., and Riedel, S. Towards neural theorem proving at scale. *arXiv preprint arXiv:1807.08204*, 2018.

Mobahi, H., Farajtabar, M., and Bartlett, P. L. Self-distillation amplifies regularization in hilbert space. In *NeurIPS*, 2020.

Moen, S. and Ananiadou, T. S. S. Distributional semantics resources for biomedical text processing. *Proceedings of LBM*, pp. 39–44, 2013.

Ning, X., Desrosiers, C., and Karypis, G. *A Comprehensive Survey of Neighborhood-Based Recommendation Methods*, pp. 37–76. Springer, 2015.

Peng, N., Poon, H., Quirk, C., Toutanova, K., and Yih, W.-t. Cross-sentence n-ary relation extraction with graph lstms. *Transactions of the Association for Computational Linguistics*, 5:101–115, 2017.

Qi, M., Qin, J., Li, A., Wang, Y., Luo, J., and Gool, L. V. stagnet: An attentive semantic RNN for group activity recognition. In *ECCV*, pp. 104–120, 2018.

Qu, M. and Tang, J. Probabilistic logic neural networks for reasoning. In *NeurIPS*, pp. 7710–7720, 2019.

Rashed, A., Grabocka, J., and Schmidt-Thieme, L. Attribute-aware non-linear co-embeddings of graph features. In *Proceedings of the 13th ACM conference on recommender systems*, pp. 314–321, 2019.

Richardson, M. and Domingos, P. M. Markov logic networks. *Mach. Learn.*, 62(1-2):107–136, 2006.

Salakhutdinov, R. and Mnih, A. Bayesian probabilistic matrix factorization using markov chain monte carlo. In *ICML*, pp. 880–887, 2008.

Sanh, V., Debut, L., Chaumond, J., and Wolf, T. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108*, 2019.

Sikka, K., Silberfarb, A., Byrnes, J., Sur, I., Chow, E., Divakaran, A., and Rohwer, R. Deep adaptive semantic logic (dasl): Compiling declarative knowledge into deep neural networks. *arXiv preprint arXiv:2003.07344*, 2020.

Tsamoura, E., Hospedales, T., and Michael, L. Neural-symbolic integration: A compositional perspective. In *AAAI*, 2021.

van Krieken, E., Acar, E., and van Harmelen, F. Analyzing Differentiable Fuzzy Implications. In *KR*, pp. 893–903, 2020.

Wang, H. and Poon, H. Deep probabilistic logic: A unifying framework for indirect supervision. In *EMNLP*, pp. 1891–1902, 2018.

Wu, J., Wang, L., Wang, L., Guo, J., and Wu, G. Learning actor relation graphs for group activity recognition. In *CVPR*, pp. 9964–9974, 2019.

Xie, Y., Xu, Z., Meel, K. S., Kankanhalli, M. S., and Soh, H. Embedding symbolic knowledge into deep networks. In *NeurIPS*, pp. 4235–4245, 2019.

Yang, Z., Ishay, A., and Lee, J. NeurASP: Embracing neural networks into answer set programming. In *IJCAI*, pp. 1755–1762, 2020.

Zhu, Y., Fathi, A., and Fei-Fei, L. Reasoning about object affordances in a knowledge base representation. In *ECCV*, volume 8690, pp. 408–424, 2014.

A. Parallel neurosymbolic architectures

Let $\mathcal{X}$ and $\mathcal{Y}$ denote the input and target domain, respectively, $(\mathbf{x}, \mathbf{y}) \in \mathcal{X} \times \mathcal{Y}$ denote a training datum and $\mathcal{D}$ denote the set of training data. Both T-S and DPL assume that the logical theory includes constraints of the form $\mathcal{X} \times \mathcal{Y} \rightarrow [0, 1]$ where each constraint is associated with some level of confidence $\lambda \in [0, 1]$. These constraints are functions checking the validity of a specific training datum (returning zero for no validity and one for perfect validity) and can be captured using graphical models.

A.1. The T-S framework

Assuming that the neural model with parameters $\boldsymbol{\theta}$ defines a conditional probability distribution $p_{\boldsymbol{\theta}}(\mathbf{Y}|\mathbf{X})$, T-S builds a teacher probabilistic model $q(\mathbf{Y}|\mathbf{X})$ and uses this network to update the parameters of the neural model $\boldsymbol{\theta}$. The teacher network is found by solving the following optimization problem:

$$\begin{aligned} &\min_{q,\xi \geq 0} KL((q(\mathbf{Y}|\mathbf{X})||(\mathbf{Y}|\mathbf{X})) + C \sum_l \xi_l \\ &\text{s.t. } \lambda_l(1 - \mathbb{E}[Z_l] \leq \xi_l, \forall l \end{aligned} \tag{13}$$

In the above formulation, C denotes a balancing parameter, ξ_l denotes a slack variable along the lines of (Ganchev et al., 2010) and ξ is the vector over all the ξ_l occurring in the optimization objective. Let us define a random variable Z_l with domain the set of possible confidences $\{r_l(\mathbf{x}, \mathbf{y})\}_{(\mathbf{x},\mathbf{y})\in\mathcal{D}}$ when provided in the input with different items from the training set, where the probability of each $r_l(\mathbf{x}, \mathbf{y})$ is $q(\mathbf{Y}|\mathbf{X})$. Then $\mathbb{E}[Z_l]$ denotes the expectation of Z_l. In words, the teacher probability function is one that stays close to the neural predictions (that is what the KL term accounts for) and fits the rules (that is what the second term accounts for).

The T-S formulation makes several assumptions. First, the shape of the constraints r_l does not allow expressing generic first order logic theories. For instance, T-S cannot support the rule , as the rule references atoms that do not occur in the training data. Secondly, the optimization objective does not abide by the semantics of lifted graphical models. In particular, in the optimal solution, the ξ_l's become 0 so that $\mathbb{E}[Z_l]$ becomes 1. However, in such a case, the confidence of each constraint λ_l is not taken into account.

A.2. The DPL framework

DPL assumes that the training data is of the form $\{\mathbf{x}_i\}_{i=1}^N$. Let Φ encode the probability distribution defined by the logical theory. Let us denote the conditional probability distribution defined by the neural model with parameters $\boldsymbol{\theta}$ by $p_{\boldsymbol{\theta}}(\mathbf{Y}|\mathbf{X})$. DPL defines the joint probability distribution:

$$\begin{aligned} &P^{DPL}(Y_1, \ldots, Y_N|X_1, \ldots, X_N) = \\ &\Phi(X_1, \ldots, X_N|Y_1, \ldots, Y_N) \cdot \prod_i p_\theta(Y_i|X_i)) \end{aligned} \tag{14}$$

Above, Y_i denotes a random variable with domain the possible labels of $\mathbf{x}_i$.

DPL uses the distribution from (14) to train the components. In particular, at each training step, DPL first computes the marginal distributions $q_i(Y_i) = P^{DPL}(Y_1|X_1, \ldots, X_N)$. The product of those distributions over all i's, where $1 \leq i \leq N$, defines a new probability distribution $q(Y_1, \ldots, Y_N)$. The weights of the logical theory are updated by firstly, computing the KL divergence between $q(Y_1, \ldots, Y_N)$ and $\Phi(Y_1, \ldots, Y_N|X_1, \ldots, X_N)$ and, secondly, updating the weights of Φ so that the KL divergence is minimized. The weights of the deep network are amended in the same fashion.

The above formulation exposes several restrictions. Firstly, similarly to T-S, DPL does not straightforwardly extend to more expressive probabilistic logic theories as the constraints can be expressed as factors in MRFs (see section 3). The extension to lifted graphical models is not discussed. Hence, it does not allow expressing information in latent variables. Secondly, the assumption of independence between the logic and the deep component is not justified.

B. Details on the empirical analysis

Concordia has been developed in PyTorch 1.10. The logic component of each task was implemented using the pslpython library[4].

Computational environment All experiments ran on a Linux machine with a NVidia GeForce GTX 1080 Ti GPUs, 64 Intel(R) Xeon(R) Gold 6130 CPUs, and 256GB of RAM.

B.1. CAD

Dataset We used the Collective Activity Augmented Dataset (CAAD) (Choi et al., 2011). The dataset includes 44 video sequences with five different group activities (crossing, waiting, queuing, walking and talking) and six different individual actions (N/A, crossing, waiting, queuing, walking and talking). The group activity of each frame is defined as the activity followed by most of the actors in the frame. Each input datum $\mathbf{x}$ includes a video frame and a set of bounding boxes within the given frame. Each label datum $\mathbf{y}$ defines the action within each bounding box, as well as the activity in the frame. We used the train and test splits proposed in (Qi et al., 2018), namely choosing 2/3 of the video sequences for training and the rest for testing.

[4]https://pypi.org/project/pslpython/

B.1.1. LOGICAL COMPONENT

Theory $\mathcal{L}_{\texttt{CAD}}$ includes the rules:

$\lambda_1 : \text{FRAME}(B, F) \wedge \text{FLABEL}(F, A) \rightarrow \text{DOING}(B, A)$

$\lambda_2 : \text{DOING}(B_1, A) \wedge \text{CLOSE}(B_1, B_2) \rightarrow \text{DOING}(B_2, A)$

$\lambda_3 : \text{SEQUENCE}(B_1, B_2) \wedge \text{CLOSE}(B_1, B_2)$
$\rightarrow \text{SAME}(B_1, B_2)$

$\lambda_4 : \text{DOING}(B_1, A) \wedge \text{SAME}(B_1, B_2) \rightarrow \text{DOING}(B_2, A)$

$\lambda_5 : \text{DNN}(B, A) \rightarrow \text{DOING}(B, A)$

Atom $\text{CLOSE}(B_1, B_2)$ denotes that bounding boxes B_1 and B_2 are close to each other. Atom $\text{DOING}(B, A)$ denotes that the actor within bounding box B is doing action A. Atom $\text{FRAME}(B, F)$ denotes that bounding box B belongs to frame F, atom $\text{FLABEL}(F, A)$ denotes that the group activity of frame F is A, atom $\text{SEQUENCE}(B_1, B_2)$ denotes that two bounding boxes are from two frames in a direct sequence of each other, and atom $\text{SAME}(B_1, B_2)$ denotes whether the actor within bounding boxes B_1 and B_2 is the same. The first rule states that the activity of an actor is the same with the activity of the frame. The second rule states that two actors that are close to each other perform the same activity. The third rule states that if two bounding boxes are from a direct sequence of frames, and the two bounding boxes across the two frames are close to each other, then they describe the same actor. The fourth rule states that if the actor within two bounding boxes is the same, then it is likely that she is performing the same activity. The last rule is as in Example 3. The above rules hold with some uncertainty which is captured by the λ parameters.

At training time, we provided instantiations of the predicates FRAME, FLABEL, CLOSE, DNN, SEQUENCE and DOING using the training data. To instantiate the predicate CLOSE we used an RBF kernel to measure the closeness of the bounding boxes, which outputs a value in [0, 1]. To instantiate predicate DNN, we followed the steps in Section 4.3. At testing time, we provided instantiations to all predicates but DOING.

Note that neither DPL nor T-S can implement this logic due to the latent variable SAME, as both expect all conclusions of each rule to be target atoms. In addition, they are not capable of handling this task due to its multi-task nature, as it predicts both labels for each bounding box in the frame, as well as the group activity of the frame.

B.1.2. NEURAL COMPONENT

We considered the same architecture as the state-of-the-art (Wu et al., 2019; Kuang & Tie, 2020): `B` $\rightarrow$ `RoIAlign` $\rightarrow$ `L`. Component `B` takes as input an image represented by a 2-dimensional vector of size 480x720 and creates a feature map for each input frame f of size 57x87. Given the feature map computed by `B`, `RoIAlign` (He et al., 2017a) then outputs feature vectors for each bounding box within f of size 1024. `L` is a fully connected layer predicting the activity within each bounding box in f, as well as the group activity in f. To ensure a fair comparison with prior art, we used `MobileNet` and `Inception-v3` for `B` ending up with two different instantiations of the neural architecture. We will refer to those architectures as `MobileNet` and `Inception-v3`. The loss function used in this experiment is cross-entropy.

Inputs/outputs The inputs are a set of bounding boxes all given as coordinates within the frames, which in turn are passed into the neural model as 2-dimensional vectors of size 480x720. The outputs are soft-max distribution vectors over the possible activities for each bounding box and the group activity of the frame, which include crossing, waiting, queuing, walking, talking, dancing, jogging, and N/A.

Training To train `MobileNet` and `Inception-v3` we used a minibatch of size 1 and set the learning rate to 0.00001.

B.1.3. BASELINES

We considered `MobileNet` and `Inception-v3` as baselines and trained them for 30 epochs each. We also compared against the state-of-the-art architecture from (Kuang & Tie, 2020), which uses a Graph Convolutional Network on top of `MobileNet` and `Inception-v3`. The technique is referred to as IARG. We denote by IARG(`MobileNet`) and IARG(`Inception-v3`) the two different variants. We used the implementation provided by the authors[5] and used the hyper-parameters from (Wu et al., 2019; Kuang & Tie, 2020). IARG(`MobileNet`) and IARG(`Inception-v3`) were trained for 100 epochs as in (Kuang & Tie, 2020). As neither T-S nor DPL support the rules in $\mathcal{L}_{\texttt{CAD}}$, we did not consider them as baselines.

B.2. Recommendation

Dataset We considered Yelp (the academic version) and MovieLens-100k (Harper & Konstan, 2015). Yelp contains user ratings on local businesses, as well as information about business categories and friendships between users. The goal is to predict the ratings of the users on businesses they haven't rated yet. The dataset is updated on an annual basis by adding different businesses. We used the version from 2021 and considered businesses only from Cambridge (US). MovieLens-100k is a movie recommendation dataset containing categorical information of movie genres and user occupations.

We used a 90%/10% training/test split for both Yelp and MovieLens-100k. For MovieLens-100k, we used the split

[5] https://github.com/kuangzijian/Improved-Actor-Relation-Graph-based-Group-Activity-Recognition.

provided by the publishers of the dataset. For Yelp, we created a random split. We used 30% of the training data as the unobserved variables and 70% of the training data as the observed variables of each dataset to learn the rules' weights.

B.2.1. LOGICAL COMPONENT

PSL has already been successfully applied in recommendation (Kouki et al., 2015). We used the same rules in our experiments. Below, we describe the rules from $\mathcal{L}_{\text{REC}}$ in detail (omitting the rules weights for clarity).

The first two rules encode information about the item and user similarity:

$$\begin{aligned}\text{SIMILARUSERS}_{sim}(U_1, U_2) \wedge \text{RATING}(U_1, I) \\ \rightarrow \text{RATING}(U_2, I) \\ \text{SIMILARITEMS}_{sim}(I_1, I_2) \wedge \text{RATING}(U, I_1) \\ \rightarrow \text{RATING}(U, I_2)\end{aligned}$$

The first rules states that if two users are similar, then they will give similar ratings to items. The second states that users will give similar ratings to similar items. These rules are also agnostic to the similarity measure used between users and between items. We adopted the same similarity metrics used by Kouki et al. (Kouki et al., 2015), namely Pearson, Cosine, latent cosine, latent euclidean– the last two metrics are computed on latent vectors extracted using the Matrix Factorization approach on the rating matrix of all users and items (Ning et al., 2015). The first two similarity metrics are computed between rating vectors for each user and item. In addition, for item-based similarity, the adjusted cosine similarity is also used. To follow the authors, we also integrated content-based similarity. Content-based similarity included information, such as restaurant's category and user's occupation. The similarity measure used for content-based similarity is the Jaccard index. Predicates $\text{SIMILARUSERS}_{sim}$ and $\text{SIMILARITEMS}_{sim}$ are binary predicates that take values of 1 if the first constant is one of the k-nearest neighbours of the second constant. In this experiment we used $k = 50$. Predicate RATING takes values in the range [0, 1] and represents the normalized rating score user U gave to an item I.

The authors in (Kouki et al., 2015) also included rules encouraging the predicted ratings to be close to an average user rating a user gives to items and an average item rating that is usually given to the item by all other users:

$$\begin{aligned}\text{AVERAGEUSERRATING}(U) \leftrightarrow \text{RATING}(U, I) \\ \text{AVERAGEITEMRATING}(U) \leftrightarrow \text{RATING}(U, I)\end{aligned}$$

PSL can also leverage existing collaborative filtering methods to compute the ratings as defined in the rules above. To follow the original work, the theory $\mathcal{L}_{\text{REC}}$ included the same most widely used collaborative filtering methods: matrix factorization (MF) (Koren et al., 2009), Bayesian probabilistic matrix factorization (BP) (Salakhutdinov & Mnih, 2008), and item-based collaborative filtering (IB):

$$\text{RATING}_{MF}(U, I) \leftrightarrow \text{RATING}(U, I)$$

$$\text{RATING}_{BP}(U, I) \leftrightarrow \text{RATING}(U, I)$$

$$\text{RATING}_{IB}(U, I) \leftrightarrow \text{RATING}(U, I)$$

Finally, $\mathcal{L}_{\text{REC}}$ incorporated social network influences, like friends have similar tastes and thus give similar ratings:

$$\text{FRIENDS}(U_1, U_2) \wedge \text{RATING}(U_1, I) \rightarrow \text{RATING}(U_2, I)$$

The predicates were instantiated as discussed in (Kouki et al., 2015). Concretely, the RATING predicates were instantiated directly from the data. For AVERAGEUSERRATING and AVERAGEITEMRATING, we computed the average as usual, while the similarity measurements were computed on the remaining data in the dataset, such as age, occupation and sex, using popular similarity measurements such as cosine, Pearson, latent cosine, and latent euclidean measurements.

Note that neither DPL nor T-S can implement these kind of rules due to the bi-directionality, of e.g. $\text{AVERAGEUSERRATING}(U) \leftrightarrow \text{RATING}(U, I)$, as both expect all conclusions of the rules to be the target atoms.

B.2.2. NEURAL COMPONENT

We used three state-of-the-art networks: `NNMF`(Dziugaite & Roy, 2015), `NeuMF`(He et al., 2017b), and `GraphRec` (Rashed et al., 2019). All networks learn embedding vectors for each user and item (business or movie).

Inputs/outputs The inputs to the networks are pairs of user/items. The outputs are the item ratings.

Training The batch size used for `NNMF`was 32, the learning rate 0.001, and the L_2 norm set to 0.01 for regularization, and for `NeuMF`batch size was 16, learning rate 0.001, and the L_2 norm was 0.01. The parameters for `GraphRec` were as follows: batch size = 1000, learning rate was set to 0.00003, the L_2 norm was set to 0.05 for the user features and 0.02 for the item features. In all three baseline models the optimizer was set to Adam with loss function set to RMSE.

B.2.3. BASELINES

We considered four different baselines: `NNMF`, `NeuMF`, `GraphRec` and the PSL theory $\mathcal{L}_{\texttt{REC}}$ alone. The parameters were tuned according to the respective papers (Dziugaite & Roy, 2015; He et al., 2017b; Rashed et al., 2019).

B.3. Entity linking

Dataset We used the PubMedParsed data set originally put forth by (Moen & Ananiadou, 2013), where we used the data generation and processing files from (Wang & Poon, 2018) for a fair comparison. The goal in this data set is to predict which words in a text are mentions of protein names. The data contains 96k unlabelled data items, where each item is a sentence. In addition, for testing the performance of the unsupervised model, the authors in (Wang & Poon, 2018), provided a set of 12k labelled data items, where the mentions were labelled as protein or non-protein.

B.3.1. LOGICAL COMPONENT

We used the theory from (Wang & Poon, 2018) and in particular, the one using distant supervision (DS), data programming (DP) and joint inference (JI).

B.3.2. NEURAL COMPONENT

We used a Bidirectional Long Short-Term Memory (`BiLSTM`) recurrent neural network, originally proposed by (Peng et al., 2017), and used the implementation from (Wang & Poon, 2018) for a fair comparison. In addition, we compared the performance of Concordia to DistilBERT (`BERT`) (Sanh et al., 2019) a distilled version of BERT (Devlin et al., 2018) while retaining 97% of its language understanding capabilies and being 60% faster. The aim was to understand how Concordia performs with respect to large pre-trained models.

Inputs/outputs The inputs are mentions from sentences from the PubMedParsed dataset (Moen & Ananiadou, 2013). The output is a prediction on whether the mention is a protein or a non-protein. To generate the training and testing inputs, we used the data generation scripts from (Wang & Poon, 2018).

Training In the case of the `BiLSTM`, the embedding layer is initialized with a word2vec embedding trained on PubMed abstracts and entire texts. The word embedding dimension was 200 as in (Wang & Poon, 2018). We used a learning rate of 0.001 and batch size 64. In the case of `BERT`, we used the word embedding provided by the model itself as it is a pretrained model. We used a learning rate of 0.00003 and batch size 16. The loss function used in all experiments was cross-entropy.

B.3.3. BASELINES

We were not able to reproduce the results reported in (Wang & Poon, 2018) due to the exponential approach proposed by the authors to compute their factor graphs. The authors ran their experiments on clusters. Therefore, the results on DPL reported in the main body are taken from their paper.

C. Connection from LGMs to PSL

We demonstrate the notions of LGMs and PSL by example.

Example 4. *Consider the second rule r from Section B.1.1:*

$$\text{DOING}(B_1, A) \wedge \text{CLOSE}(B_1, B_2) \rightarrow \text{DOING}(B_2, A)$$

Assuming that there are two bounding boxes in total, $\texttt{b}_1$ and $\texttt{b}_2$, and one activity, `crossing`, rule r can be instantiated in two different ways:

$$\overset{X_{1,1}}{\text{DOING}}(\texttt{b}_1, \texttt{crossing}) \wedge \overset{X_{1,2}}{\text{CLOSE}}(\texttt{b}_1, \texttt{b}_2) \rightarrow \overset{X_{1,3}}{\text{DOING}}(\texttt{b}_2, \texttt{crossing}) \tag{15}$$

$$\overset{X_{1,3}}{\text{DOING}}(\texttt{b}_2, \texttt{crossing}) \wedge \overset{X_{2,2}}{\text{CLOSE}}(\texttt{b}_2, \texttt{b}_1) \rightarrow \overset{X_{1,1}}{\text{DOING}}(\texttt{b}_1, \texttt{crossing}) \tag{16}$$

Above each ground atom we show the RV associated with it, e.g., ground atom $\text{DOING}(\texttt{b}_1, \texttt{crossing})$ is associated with the RV $X_{1,1}$, while ground atom $\text{CLOSE}(\texttt{b}_1, \texttt{b}_2)$ is associated with RV $X_{1,2}$. The domain of all RVs is $[0, 1]$.

We denote by f_1 and f_2 the factors associated with the rule instantiations (15) and (16), respectively. Let $\mathbf{X}_1 = (X_{1,1}, X_{1,2}, X_{1,3})$ and $\mathbf{X}_2 = (X_{1,3}, X_{2,2}, X_{1,1})$. In PSL, each factor f_i is defined as follows

$$f_i(\mathbf{X}_i = \mathbf{x}_i) = e^{-\lambda \cdot (1 - r(\mathbf{x}_i))^p}, \tag{17}$$

where $\mathbf{x}_i$ an instantiation of $\mathbf{X}_i$, for $i = 1, 2$, and p is as defined in (5). The par-factor ϕ associated with rule r is essentially the set $\{f_1, f_2\}$. We use $\phi(\mathbf{X}_i = \mathbf{x}_i)$ to denote $f_i(\mathbf{X}_i = \mathbf{x}_i)$, for $i = 1, 2$.

Let $\mathbf{X} = (X_{1,1}, X_{1,2}, X_{2,2}, X_{1,3})$ denote the vector composed over all RVs associated with ground atoms in the Herbrand base of our theory. Probability $P(\mathbf{X} = \mathbf{x})$ induced by the par-factor graph $\{\phi\}$, where $\mathbf{x} = (x_{1,1}, x_{1,2}, x_{2,2}, x_{1,3})$ is an instantiation of $\mathbf{X}$, is defined as follows:

$$P(\mathbf{X} = \mathbf{x}) = \frac{1}{Z} \phi((x_{1,1}, x_{1,2}, x_{1,3})) \cdot \phi((x_{1,3}, x_{2,2}, x_{1,1}))$$

where Z is the normalization constant.

D. Details on inference in the logical component

In the main body of this paper, we touched on the differences of inference in the logical component of Concordia. In this section, we are revisiting inference in classification tasks, by considering the CAD example. Then, we will elaborate on the differences in regression.

D.1. Classification

We elaborated in the main body of this paper on inference in classification tasks, where we discussed the difference between the logical component admitting Boolean interpretations vs interpretations in [0,1].

Example 5. *Consider the task of collective activity detection, described in Section B.1.1. In this task, the* DOING *atoms, are what we referred to as target atoms in the main body, as the labels for our data are the activities of the different bounding boxes in a frame.*

Boolean interpretation *We mentioned in the case of Boolean interpretation of the atoms, where the atoms are mapped to true or false, we obtain a probability for each option of $c_j \in \mathcal{Y}$. That is, we get for a bounding box B_i $P(\text{DOING}(B_i, \texttt{dancing}) = \texttt{true}) = p_1$, $P(\text{DOING}(B_i, \texttt{talking}) = \texttt{true}) = p_2$, ... However, obviously, only one activity can be true at a time, and therefore, $p_1 + p_2 + \cdots + p_n = 1$. To enforce this, we need to add an additional rule to the rules mentioned in Section B.1.1, enforcing the mutual exclusiveness*

$$\text{DOING}(B_i, \texttt{dancing}) \veebar \text{DOING}(B_i, \texttt{talking}) \veebar \dots$$

Soft interpretation *When the logical interpretations accept values in $[0, 1]$, we have in effect, a probability distribution for each target atom, i.e. one distribution for* DOING$(B_i, \texttt{dancing})$, DOING$(B_i, \texttt{talking})$, ... *Then, as described in Section 4.1, we choose the probability of each class, as the MAP over the distribution of the soft truth values. Finally, the values need to be normalized such that their sum is 1 to enforce the properties of a probability distribution. This is done through the following rule*

$$\text{DOING}(B_i, \texttt{dancing}) + \text{DOING}(B_i, \texttt{talking}) + \cdots = 1$$

Figure 3 illustrates how a discrete distribution is obtained for the CAD example, where only three distinct actions are considered for illustrative purposes.

D.2. Regression

We support regression tasks via PSL. Recall that each target atom maps to a continuous RV Y in $[0, 1]$, i.e. the soft truth value (see Section 3). The conditional probability

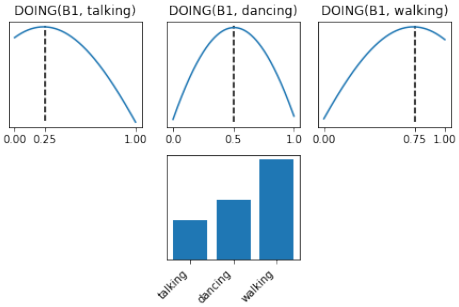

Figure 3. Illustrative example how a distribution over different labels can be obtained using PSL. Top row: each target atom has a distribution over their soft truth values. Bottom row: taking their respective MAPs and normalizing so that their sum is 1, produces a discrete distribution over the classes

$P_{\mathcal{L}}(Y = y | \mathbf{X}^o_{\mathcal{L}}, \boldsymbol{\lambda})$ denotes the likelihood the target atom Y takes soft truth value y. We define $P_{\mathcal{L}}$ by marginalizing

$$P(\mathbf{X} = \mathbf{x}) = \frac{1}{Z} \exp\left(\sum_{i=1}^{M} \sum_{j=1}^{M_i} \lambda_i f_{i,j}(\mathbf{X}_{i,j} = \mathbf{x}_{i,j}) \right),$$

over all the remaining RVs. In case the domain of the regression task does not coincide with the soft truth domain [0,1] in PSL, the soft truth values are scaled to the target domain.

Note that no additional constraint needs to be provided in regression tasks, as the distribution over the soft truth values already satisfies the axioms of a probability distribution.

Example 6. *Let us consider the recommendation task, with the logical rules as described in Section B.2.1. In this case, the target atom is* RATING*, which takes values between 1 and 5. To achieve this, we scale the interval [1,5] onto the interval [0,1], such that the soft truth value of the* RATING *atoms equals the predicted rating.*