DomiKnowS: A Library for Integration of Symbolic Domain Knowledge in Deep Learning

Hossein Rajaby Faghihi[1], **Quan Guo**[2], **Andrzej Uszok**[3], **Aliakbar Nafar**[1], **Elaheh Raisi**[1], and **Parisa Kordjamshidi**[1]
[1] Michigan State University, [2] Sichuan University
[3] Florida Institute for Human and Machine Cognition
rajabyfa@msu.edu, guoquan@scu.edu.cn, auszok@ihmc.org,
{nafarali, raisiela, kordjams}@msu.edu

Abstract

We demonstrate a library for the integration of domain knowledge in deep learning architectures. Using this library, the structure of the data is expressed symbolically via graph declarations and the logical constraints over outputs or latent variables can be seamlessly added to the deep models. The domain knowledge can be defined explicitly, which improves the explainability of the models in addition to their performance and generalizability in the low-data regime. Several approaches for such integration of symbolic and sub-symbolic models have been introduced; however, there is no library to facilitate the programming for such integration in a generic way while various underlying algorithms can be used. Our library aims to simplify programming for such integration in both training and inference phases while separating the knowledge representation from learning algorithms. We showcase various NLP benchmark tasks and beyond. The framework is publicly available at Github[1].

1 Introduction

Current deep learning architectures are known to be data-hungry with issues mainly in generalizability and explainability (Nguyen et al., 2015). While these issues are hot research topics, one approach to address them is to inject external knowledge directly into the models when possible. While learning from examples revolutionized the way that intelligent systems are designed to gain knowledge, many tasks lack adequate data resources. Generating examples to capture knowledge is an expensive and lengthy process and especially not efficient when such a knowledge is available explicitly. Therefore, one main motivation of our proposed framework (DomiKnowS) is to facilitate the integration of domain knowledge in deep learning architectures, in particular when this knowledge is represented symbolically.

In this demonstration paper, we highlight the components of this framework that help to combine learning from data and exploiting knowledge in learning, including: 1) Learning problem specification 2) Knowledge representation 3) Algorithms for integration of knowledge and learning. Currently, DomiKnowS implementation relies on PyTorch and off-the-shelf optimization solvers such as Gurobi.[2] However, it can be extended by developing hooks to other solvers and deep learning libraries since the interface is generic and independent from the underlying computational modules.

In general, the integration of domain knowledge can be done 1) using pretrained models and transferring knowledge (Devlin et al., 2019; Mirzaee et al., 2021), 2) designing architectures that integrate knowledge expressed in knowledge bases (KB) and knowledge graphs (KG) in a way that the KB/KG context influences the learned representations (Yang and Mitchell, 2017; Sun et al., 2018), or 3) using the knowledge explicitly and logically as a set of constrains or preferences over the inputs or outputs (Li and Srikumar, 2019a; Nandwani et al., 2019b; Muralidhar et al., 2018; Stewart and Ermon, 2017). Our current library aims at facilitating the third approach. It is still an open problem to know which method of integrating prior knowledge with neural modules is the best and the performance heavily relies on each task specifications and experimental settings. However, there are many ongoing research on the integration of soft/hard constraints (Li and Srikumar, 2019a; Nandwani et al., 2019b; Muralidhar et al., 2018; Stewart and Ermon, 2017) on the output variables of neural modules which shows the effectiveness of such approaches in gaining better performance, especially on low-resource or even semi-supervised

[1]https://github.com/HLR/DomiKnowS

[2]Gurobi provides free academic license, moreover it provides a limited free version for all which is sufficient to solve problems with a small number of variables. We, also, have started adding hooks to other optimization tools that are freely available such as GEKKO.

tasks (Ratner et al., 2017). While applying the constraints on input is technically trivial and could be done in a data pre-processing step, applying constraints over outputs and considering those structural constraints during training is a research challenge (Nandwani et al., 2019b; Li and Srikumar, 2019a; Guo et al., 2020). This requires encoding the knowledge at the algorithmic level. However, given that the constraints can be expressed logically and symbolically, having a language to express such a knowledge in a principled way is lacking in the current machine learning libraries. Using our developed DomiKnowS library, the domain knowledge will be provided symbolically and by the user utilizing a logical language that we have defined. This knowledge is used in various ways: a) As soft constraints by considering the violations as a part of loss function, this is done using a prim-dual formulation (Nandwani et al., 2019b) and can be expanded to probabilistic and sampling-based approaches (Xu et al., 2018) or by mapping the constraint to differentiable operations (Li and Srikumar, 2019b) b) mapping the constrains to an integer linear program setting and performing inference-based training by masking the loss (Guo et al., 2020). Independent form the training paradigm the constraints can be always used as hard constraints during inference or not used at all.

An interactive online demo of DomiKnowS is available at Google Colab[3] and the framework is accessible on GitHub[4].

2 Related Research

Integration of domain knowledge in learning relates to tools that try to express the prior or posterior information about variables beyond what is in the data. This relates to probabilistic programming languages such as (Pfeffer, 2016), Venture (Mansinghka et al., 2014), Stan (Carpenter et al., 2017), and InferNet (Minka et al., 2012). The logical expression of domain knowledge is used in probabilistic logical programming languages such as ProbLog (De Raedt et al., 2007), PRISM (Sato and Kameya, 1997), the recent version of Problog,i.e., Deep Problog (Manhaeve et al., 2018), Statistical Relational Learning tools, such as Markov logic networks (Domingos and Richardson, 2004), Probabilistic soft logic (Broecheler et al., 2010), Bayesian Logic (BLOG) (Milch et al., 2005), and slightly related to learning over graph structures (Zheng et al., 2020). Considering the structure of the output without its explicit declaration is considered in structured output prediction tools (Rush, 2020). Our library is mostly related to the previous efforts for learning based programming and the integration of logical constraints in learning with classical machine learning approaches (Rizzolo and Roth, 2010; Kordjamshidi et al., 2015, 2016). Our framework makes this connection to deep neural network libraries and arbitrarily designed architectures. The unique feature of our library is that, the graph structure is defined symbolically based on the concepts in the domain. Unlike Torch-struct (Rush, 2020), our library is independent from the underlying algorithms, and arbitrary structures can be expressed and used based on various underlying algorithms. In contrast to DeepProbLog, we are not limited to probabilistic inference and any solver can be used for inference depending on the training paradigm that is used for exploiting the logical constraints. Probabilistic soft logic is another framework that considers logical constraints in learning by mapping the constraint declarations to a Hing loss Markov random field (Bach et al., 2017). DRaiL is another declarative framework that is using logical constraints on top of deep learning and converts them to an integer linear program at the inference time (Zhang et al., 2016). None of the above mentioned frameworks accommodate working with raw sensory data nor help in putting that in an operational structure that can form the domain predicates and be used by learning modules while our framework tries to address that challenge. We support training paradigms that make use of the inference as a black box and in those cases any constraint optimization, logical inference engine or probabilistic inference tools can be integrated and used based on our abstraction and the provided modularity.

3 Declarative Learning-based Programming

We use the Entity-Mention-Relation (EMR) extraction task to describe the framework. We discuss more showcases in Section 5.

Given an input text such as "*Washington is employed by Associated Press.*", the task is to extract the entities and classify their types (e.g., people, organizations, and locations) as well as relations

[3] https://tinyurl.com/t9wr9n4
[4] https://github.com/HLR/DomiKnowS

between them (e.g., works for, lives in). For example, for the above sentence *[Washington]* is a *person* *[Associated Press]* is an *organization* and the relationship between these two entities is *work-for*. We choose this task as it includes the prediction of multiple outputs at the sentence level, while there are global constraints over the outputs.[5]

In DomiKnowS, first, using our python-based specification language, the user describes the problem and its logical constraints declarativly and independent from the solutions. Second, it defines the necessary computational units (here, PyTorch-based architectures) and connect the solution to the problem specification. Third, a program instance is created to execute the model using a background knowledge integration method with respect to the problem description.

3.1 Problem Specification

To model a problem in DomiKnowS, the user should specify the problem domain as a conceptual graph $\mathcal{G}(V, E)$. The nodes in V represent concepts and the edges in E are relationships. Each node can take a set of properties $P = P_1, P_2, ..., P_n$. Later, the logical constraints are expressed using the concepts in the graph. In EMR task, the graph contains some initial NLP concepts such as *sentence*, *phrase*, *pair* and additional domain concepts such as *people*, *organization*, and *work-for*.

3.1.1 Concepts

Each problem definition can contain three main types of concepts (nodes).

Basic Concepts define the structure of the input of the learning problem. For instance *sentence*, *phrase*, and *word* are all basic concepts that can be defined in the EMR task.

Compositional Concepts are used to define the many-to-many relationships between the basic concepts. Here, the *pair* concept in the EMR task is a compositional concept. This is used as the basic concept for the relation extraction task. We will further discuss this when describing edges in Section 3.1.2.

Decision Concepts are derived concepts which are usually the outputs of the problem and subject to prediction. They are derived from the basic or compositional concepts. The *people*, *organization*, and *work-for* are examples of derived concepts in the EMR conceptual graph. Following is a partial snippet showing the definition of basic and compositional concepts for EMR task.

```
word = Concept(name='word')
phrase = Concept(name='phrase')
sentence = Concept(name='sentence')
pair = Concept(name='pair')
```

The following snippet also shows the definition of some derived concepts in EMR example.

```
entity = phrase(name='entity')
people = entity(name='people')
org = entity(name='organization')
location = entity(name='location')
work_for = pair(name='work_for')
located_in = pair(name='located_in')
```

The *entity*, *people*, *organization* and *location* are the derived concepts from the *phrase* concept and the rest are derived from the *pair* concept.

3.1.2 Edges

After defining the concepts, the user should specify existing relationships between them as edges in the conceptual graph. Edges are used to either map instances from one concept to another, or generate instances of a concept from another concept. DomiKnowS only supports a set of predefined edge types, namely *is_a*, *has_a*, and *contains*.

is_a is automatically defined between a derived concept and its parent. In the EMR example, there is an *is_a* edge between *people* and *entity*. The *is_a* edge is mostly used to introduce hierarchical constraints and relate the basic and derived concepts.

Has_a connects a compositional concept to its components (also referred to as *arguments*). In the EMR example, *pair* concept has two *has_a* edges to the *phrase* concept to specify the *arg1* and *arg2* of the composition. We allow an arbitrary number of arguments in a *has_a* relationship, see below.

```
pair.has_a(arg1=phrase, arg2=phrase)
```

Contains edge defines a one-to-many relationship for two concepts to represent a (parent, child) relationship between them. Here, the number of parents of a concept is not necessarily limited to be only one. Following is a sample snippet to define a *contains* edge between *sentence* and *phrase*:

```
sentence.contains(phrase)
```

[5]Please note this is just an example of a learning problem and does not have anything to do with the main functionality of the framework.

3.1.3 Global Constraints

The constraint definition is the part where the prior knowledge of the problem is defined to enable domain integration. The constraints of each task should be defined on top of the problem using the specified concepts and relationships there.

The constraints can be 1) automatically inferred from the conceptual graph structure, 2) extracted from the standard ontology formalism (here OWL[6]), 3) explicitly defined using the logical constraint language of DomiKnowS. The framework internally uses the defined constraints at the training-time or the inference-time optimization depending on the integration method selected for the task. We discuss the inference phase in more details in section 4.1. Here is an example of a constraint written in DomiKnowS's logical constraint language for the EMR task:

```
ifL(work_for('x'), andL(people(path=
↪  ('x',arg1)),
↪  organization(path='x',arg2)))
```

The above constraint indicates that a *work_for* relationship only holds between *people* and *organization*. Other syntactic variations of this constraint are shown in the Appendix.

To process constraints, DomiKnowS maps those to a set of equivalent algebraic inequalities or their soft logic interpretation depending on the integration method. We discuss this more in Section 4.1.

3.2 Model Declaration

Model declaration phase is about defining the computational units of the task. The basic building blocks of the model in DomiKnowS are *sensors* and *learners*, which are used to define either deterministic or probabilistic functionalities of the model. Sensor/Learners interact with the conceptual graph by defining properties on the concepts (nodes). Each sensor/learner receives a set of inputs either from the raw data or property values on the graph and introduces new property values. Sensors are computational units with no trainable parameters; and learners are the ones which contain the neural models. As stated before, the model declaration phase only defines the connection of the graph properties to the computational units and the execution is done later by the program instances.

The user can use any deep learning architecture compatible with PyTorch modules alongside the set of pre-designed and commonly-used neural architectures currently existing in the framework. To facilitate modeling different architectures and computational algorithms in DomiKnowS, we provide a set of predefined sensors to do basic mathematical operations and linguistic feature extraction. Following is a short snippet of defining some sensors/learners for the EMR task.

```
phrase['w2v'] = FunctionalSensor('text',
↪  forward=word2vec)
phrase[people] = ModuleLearner('w2v',
↪  module=Classifier(FEATURE_DIM))
pair[work_for] = ModuleLearner('emb',
↪  module=Classifier(FEATURE_DIM*2))
```

In this example, the sensor *Word2Vec* is used to obtain token representations from the "text" property of each *phrase*. There is also a very simple and straightforward linear neural model to classify *phrases* and *pairs* into different classes such as *people*, *organization*, etc.

4 Learning & Evaluation in DomiKnowS

To execute the defined model considering the specified conceptual graph, DomiKnowS uses program instances. A program instance is responsible to run the model, apply loss functions, optimize the parameters, connect the output decisions to the inference algorithms, and generate the final results and metrics. Executing the program instance relies on the problem graph, model declaration, dataloaders and a backbone data structure called DataNode. **DataLoader** provides an iterable object to loop over the data. **DataNode** is an instance of the conceptual graph to keep track of the data instances and store the computational results of the sensors and learners. For the EMR task, the program definition is as follows:

```
program = Program(graph,
↪  poi=(sentence, phrase, pair),
↪  loss=NBCrossEntropyLoss(),
↪  metric=PRF1())
```

Here, the concepts passed to the *poi* field specifies the training points of the program. This enables the user to train the task based on any subsets of the concepts defined in the model.

For each program instance, the user should specify the domain knowledge integration method. The available methods for integration is discussed in the next sections. After initializing the program,

[6]Ontology Web Language

the user can call *train*, *test*, and *prediction* functionalities to train and evaluate the designed model. The below snippet is to run training and evaluation on the EMR task:

```
program.train(train_reader,
 ↪   test_reader, epochs=10,
 ↪   Optim=torch.optim.SGD(param,
 ↪   lr=.001))
program.test(new_test_reader)
```

Here, the user will specify the dataloaders for different sets of the data and the hyper-parameters required to train the model.

Programs can be composed to address different training paradigms such as end-to-end or pipeline training by defining different training points for each program. More details are available in the Appendix. Following is an alternative program definition for pre-training the phrases first and then learning based on the pairs:

```
program_1 = Program(graph,
 ↪   poi=(phrase, sentence))
program_2 = Program(graph,
 ↪   poi=(pair))
program_1.train(); program_2.train()
```

4.1 Inference and Optimization

DomiKnowS provides access to a set of approaches to integrate background knowledge in the form of constraints on the output decisions or latent variables/concepts. Currently, DomiKnowS addresses three different paradigms for integration: 1) Learning + prediction time inference (L+I) 2) Training-time integration with hard constraints 3) Training-time integration with soft constraints. The first method, which we refer to it as enforcing global constraints can also be combined and applied on top of the second and third approaches at inference-time.

Prediction-time Inference: In the back-end of DomiKnowS, ILP [7] solvers are used to make inference under global linear constraints (Roth and Yih, 2005). The constraints are denoted by $\mathcal{C}(\cdot) \leq 0$. Without loss of generality, we can denote the structured output as a binary vector $y \in \mathcal{R}^n$. Given local predictions $F(\theta)$ from the neural network, the global inference can be modeled to maximize the combination of log probability scores subject to the constraints (Roth and Yih, 2005; Guo et al., 2020) as follows,

$$\begin{aligned} F^*(\theta) = \underset{y}{\operatorname{argmax}} \log F(\theta)^\top y \\ \text{subject to} \quad \mathcal{C}(y) \leq 0. \end{aligned} \tag{1}$$

To handle constraints in ILP, we create variables for each local decision of instances and transform the logical constraints to algebraic inequalities (Rizzolo and Roth, 2010) in terms of those variables. Auxiliary variables are added to represent the nested constraints. The inference method can be extended to support other approaches such as probabilistic inference and dynamic programming in future without any modifications to the other parts of the framework.

Integration of hard constraint in training: Here, we use our proposed inference-masked loss approach (IML) (Guo et al., 2020) which constructs a mask over local predictions based on the global inference results. The main intuition is to avoid updating the model based on local violations when the global inference can recover true labels from the current predictions. Given structured prediction $F(\theta)$ from a neural network and its global inference $F^*(\theta)$ subject to the constraints, IML is extended from negative log likelihood as follows

$$\begin{aligned} &\mathcal{L}_{\text{IML}}\left(F(\theta), Y\right) = \\ &-\left(\left(1 - F^*(\theta)\right) \odot Y\right)^\top \log F(\theta), \end{aligned} \tag{2}$$

where Y is the structured ground-truth labels and $\odot$ indicates element-wise product. We implemented $\mathcal{L}_{\text{IML}(\lambda)}$ which balances between negative log likelihood and IML with a factor λ as introduced in (Guo et al., 2020). IML works best for very low-resource tasks where label disambiguation cannot be learned from the data but can be done based on the available relational constraints between output variables. The constraint mapping for the IML uses the same module in the DomiKnowSthat is implemented to use the global constraint optimization tool (here ILP).

Integration of soft constraints in training: We use the primal-dual formulation of constraints proposed in (Nandwani et al., 2019a) to integrate soft constraints in training the models. Primal-Dual considers the constraints in the neural network training by augmenting the loss function using Lagrangian multipliers Λ for the violations from the constraints by the set of predictions. The constraints are regularized by a hinge function $[\mathcal{C}(F(\theta))]^+$. The problem is formulated as a min-max optimization where

[7]Integer Linear Programming

it maximizes the Lagrangian function with the multipliers to enforce the constraints and minimize it with the parameters in the neural network. Here, instead of solving the min-max primal, we solve the max-min dual of the original problem.

$$\max_{\Lambda}\min_{\theta}\mathcal{L}\left(F\left(\theta\right),Y\right)+\Lambda^{\top}\left[\mathcal{C}\left(F\left(\theta\right)\right)\right]^{+}. \quad (3)$$

During training, we optimize by minimization and maximization alternatively. With Primal-Dual strategy, the model learns to obey the constraints without requiring any additional inference. Primal-Dual is less time-consuming at prediction-time than the previous methods as it does not need an additional inference-time optimization phase. This can also be used for semi-supervised setting while exploiting the domain knowledge instead of labeled data. Handling constraints in Primal-Dual is done by mapping them to their respective soft logical interpretations (Nandwani et al., 2019b).

It is an open research topic to identify which of the integration methods performs best for different tasks. However, DomiKnowS makes it effortless to use one problem specification and run all the aforementioned methods.

5 Showcases

The effectiveness of ILP (Roth and Yih, 2005), IML (Guo et al., 2020), and Primal-Dual (Nandwani et al., 2019a) methods have been already shown in their respective papers. Here, we provide different tasks and settings to showcase our framework's abilities and flexibility to model various problems. The results, models implementation and details of experiments are (partially) available in the Supplementary part of this paper and (fully) in the GitHub Repository of DomiKnowS.[8]

5.1 EMR

Our implementation of the EMR task is based on the CoNLL (Sang and De Meulder, 2003) benchmark and follows the same setting as in (Guo et al., 2020). The model uses pre-trained BERT (Devlin et al., 2019) for token representation and a linear boolean classifier for each derived concept. The constraints used in this experiment are the domain and range constraints of pairs and the mutual exclusiveness of different derived concepts, which were seen during the previous sections. IML and Primal-Dual methods perform the same as the baseline using both 100% and 25% of the data, while ILP inference achieves 1.3% improvement on the 100% of data and 0.6% improvement on 25% of the data. More details are available in the Appendix.

[8]https://github.com/HLR/DomiKnowS

5.2 Question Answering

We use WIQA (Tandon et al., 2019) benchmark as a sample question answering task in DomiKnowS. The problem graph contains *paragraph*, *question*, *symmetric*, and *transitive* concepts. Each *paragraph* comes with a set of *questions*. As explained by Asai and Hajishirzi, enforcing constraints between different question answers is beneficial to improve the performance. By modeling those constraints in DomiKnowS, ILP improves the accuracy from 74.22% to 79.05%, IML reaches 75.49%, Primal-Dual achieves 76.59%, and the combination of Primal-Dual and ILP performs best with 80.35% accuracy. More details are available in the Appendix. Following is a sample constraint defined for this task.

```
symmetric.has_a(arg1=question,
↪   arg2=question)
ifL(is_more('x'), is_less(path=('x',
↪   arg2))
```

5.3 Image Classification

We use CIFAR-10 benchmark (Krizhevsky et al.) to show image classification task in DomiKnowS. CIFAR-10 consists of 60,000 colourful images of 10 classes with 6,000 image for each class. To construct the graph, we defined the derived concepts, *airplane*, *dog*, *truck*, *automobile*, *bird*, *cat*, *deer*, *frog*, *horse* and *ship*, and the base concept *image*. We introduce the disjoint constraint between the labels of an image. We can also define hierarchical constraints between additional upper level concepts such as *Animal* and *Object* with the existing concepts such *dog* and *ship*.

```
disjoint(truck, dog, airplane,
↪   automobile, bird, cat, deer, frog,
↪   horse, ship)
```

Both the disjoint and hierarchical constraints do not affect the accuracy of the task by a large margin and ILP can only achieve near 0.5% improvement over the classification task.

5.4 Inference-Only Example

This example is to show that DomiKnowS can solve pure optimization problems as well. The task

is similar to the classic graph-coloring problem. A set of cities are given each of which can have a fire station or not. We want to allocate the fire stations to cities with respect to the following constraint.
Constraint: For each city x, it either has a fire station or there exists a city y which is a neighbor of city x and has a fire station.

To implement this, we define the basic concept *city*, the *neighbor* relationship between two cities and the derived concept *FirestationCity*.

```
neighbor.has_a(arg1=city, arg2=city)
orL(firestationCity('x'),
↪   existsL(firestationCity(path=('x',
↪   neighbor.arg2))
```

We have also included more showcases to solve sentiment analysis (Go et al., 2009) and email spam detection in our GitHub repository. [8] We will add models for procedural reasoning (Faghihi and Kordjamshidi, 2021) and spatial role labeling (Mirzaee et al., 2021) in future.

6 Conclusions and Future Work

DomiKnowS makes it effortless to integrate domain knowledge into deep neural network models using a unified framework. It allows users to switch between different algorithms and benefit from a rich source of abstracted functionalities and computational modules developed for multiple tasks. It allows naming concepts, defining their relationships symbolically and combining symbolic and sub-symbolic reasoning over the named concepts. DomiKnowS helps in interpretability of neural architectures by providing named layers and access to the neural computations at each stage of the training and evaluation process. As a future direction, we are looking to enrich our library with predefined functionalities and neural models and further extend its ability to support more techniques on integration of domain knowledge with deep neural models as well as seamless model composition. More information about technical details and the documentation of DomiKnowS is publicly available at our website[9] and on GitHub.[10]

Acknowledgements

This project is partially funded by the Office of Naval Research (ONR) grant #N00014-20-1-2005. We thank Drew Hayward and Yue Zhang for their work on parts of this framework and the anonymous reviewers for their helpful feedback towards improving the clarity of this manuscript.

[9]https://hlr.github.io/domiknows-nlp/
[10]https://github.com/HLR/DomiKnowS

References

Akari Asai and Hannaneh Hajishirzi. 2020. Logic-guided data augmentation and regularization for consistent question answering. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 5642–5650.

Stephen H. Bach, Matthias Broecheler, Bert Huang, and Lise Getoor. 2017. Hinge-loss markov random fields and probabilistic soft logic. *Journal of Machine Learning Research (JMLR)*, 18:1–67.

Matthias Broecheler, Lilyana Mihalkova, and Lise Getoor. 2010. Probabilistic similarity logic. In *Conference on Uncertainty in Artificial Intelligence*.

Bob Carpenter, Andrew Gelman, Matthew Hoffman, Daniel Lee, Ben Goodrich, Michael Betancourt, Marcus Brubaker, Jiqiang Guo, Peter Li, and Allen Riddell. 2017. Stan: A probabilistic programming language. *Journal of Statistical Software, Articles*, 76(1):1–32.

Luc De Raedt, Angelika Kimmig, and Hannu Toivonen. 2007. Problog: a probabilistic Prolog and its application in link discovery. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence*, pages 2468–2473. AAAI Press.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Perdo Domingos and Matthew Richardson. 2004. Markov logic: A unifying framework for statistical relational learning. In *ICML'04 Workshop on Statistical Relational Learning and its Connections to Other Fields*, pages 49–54.

Hossein Rajaby Faghihi and Parisa Kordjamshidi. 2021. Time-stamped language model: Teaching language models to understand the flow of events. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 4560–4570.

Alec Go, Lei Huang, and Richa Bhayani. 2009. Twitter sentiment analysis.

Quan Guo, Hossein Rajaby Faghihi, Yue Zhang, Andrzej Uszok, and Parisa Kordjamshidi. 2020. Inference-masked loss for deep structured output learning. In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI-20*, pages 2754–2761. International Joint Conferences on Artificial Intelligence Organization. Main track.

P. Kordjamshidi, D. Roth, and H. Wu. 2015. Saul: Towards declarative learning based programming. In *Proc. of the International Joint Conference on Artificial Intelligence (IJCAI)*.

Parisa Kordjamshidi, Daniel Khashabi, Christos Christodoulopoulos, Bhargav Mangipudi, Sameer Singh, and Dan Roth. 2016. Better call saul: Flexible programming for learning and inference in nlp. In *Proc. of the International Conference on Computational Linguistics (COLING)*.

Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. Cifar-10 (canadian institute for advanced research).

Tao Li and Vivek Srikumar. 2019a. Augmenting neural networks with first-order logic. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 292–302.

Tao Li and Vivek Srikumar. 2019b. Augmenting neural networks with first-order logic. In *Proceedings of the 57th Conference of the Association for Computational Linguistics, ACL 2019, Florence, Italy, July 28- August 2, 2019, Volume 1: Long Papers*, pages 292–302. Association for Computational Linguistics.

Robin Manhaeve, Sebastijan Dumancic, Angelika Kimmig, Thomas Demeester, and Luc De Raedt. 2018. Deepproblog: Neural probabilistic logic programming. In *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc.

Vikash K. Mansinghka, Daniel Selsam, and Yura N. Perov. 2014. Venture: a higher-order probabilistic programming platform with programmable inference. *CoRR*, abs/1404.0099.

Brian Milch, Bhaskara Marthi, Stuart Russell, David Sontag, Daniel L. Ong, and Andrey Kolobov. 2005. BLOG: Probabilistic models with unknown objects. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*.

Tom Minka, John M. Winn, John P. Guiver, and David A. Knowles. 2012. Infer.NET 2.5. Microsoft Research Cambridge. http://research.microsoft.com/infernet.

Roshanak Mirzaee, Hossein Rajaby Faghihi, Qiang Ning, and Parisa Kordjamshidi. 2021. Spartqa: A textual question answering benchmark for spatial reasoning. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 4582–4598.

Nikhil Muralidhar, Mohammad Raihanul Islam, Manish Marwah, Anuj Karpatne, and Naren Ramakrishnan. 2018. Incorporating prior domain knowledge into deep neural networks. In *2018 IEEE International Conference on Big Data (Big Data)*, pages 36–45. IEEE.

Yatin Nandwani, Abhishek Pathak, Mausam, and Parag Singla. 2019a. A primal dual formulation for deep learning with constraints. In *NeurIPS*.

Yatin Nandwani, Abhishek Pathak, Parag Singla, et al. 2019b. A primal dual formulation for deep learning with constraints. In *Advances in Neural Information Processing Systems*, pages 12157–12168.

Anh Nguyen, Jason Yosinski, and Jeff Clune. 2015. Deep neural networks are easily fooled: High confidence predictions for unrecognizable images. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 427–436.

Avi Pfeffer. 2016. *Practical Probabilistic Programming*. Manning Publications.

Alexander Ratner, Stephen H. Bach, Henry Ehrenberg, Jason Fries, Sen Wu, and Christopher Ré. 2017. Snorkel: Rapid training data creation with weak supervision. *Proc. VLDB Endow.*, 11(3):269–282.

N. Rizzolo and D. Roth. 2010. Learning based Java for rapid development of NLP systems. In *Proceedings of the Seventh Conference on International Language Resources and Evaluation*.

D. Roth and W. Yih. 2005. Integer linear programming inference for conditional random fields. In *Proc. of the International Conference on Machine Learning (ICML)*, pages 737–744.

Alexander Rush. 2020. Torch-struct: Deep structured prediction library. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pages 335–342, Online. Association for Computational Linguistics.

Erik Tjong Kim Sang and Fien De Meulder. 2003. Introduction to the conll-2003 shared task: Language-independent named entity recognition. In *Proceedings of the Seventh Conference on Natural Language Learning at HLT-NAACL 2003*, pages 142–147.

Taisuke Sato and Yoshitaka Kameya. 1997. Prism: A language for symbolic-statistical modeling. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1330–1339.

Russell Stewart and Stefano Ermon. 2017. Label-free supervision of neural networks with physics and domain knowledge. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 31.

Haitian Sun, Bhuwan Dhingra, Manzil Zaheer, Kathryn Mazaitis, Ruslan Salakhutdinov, and William Cohen. 2018. Open domain question answering using early fusion of knowledge bases and text. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 4231–4242.

Niket Tandon, Bhavana Dalvi, Keisuke Sakaguchi, Peter Clark, and Antoine Bosselut. 2019. WIQA: A dataset for "what if..." reasoning over procedural text. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 6076–6085, Hong Kong, China. Association for Computational Linguistics.

Jingyi Xu, Zilu Zhang, Tal Friedman, Yitao Liang, and Guy Van den Broeck. 2018. A semantic loss function for deep learning with symbolic knowledge. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 5502–5511. PMLR.

Bishan Yang and Tom Mitchell. 2017. Leveraging knowledge bases in lstms for improving machine reading. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1436–1446.

Xiao Zhang, Maria Leonor Pacheco, Chang Li, and Dan Goldwasser. 2016. Introducing DRAIL – a step towards declarative deep relational learning. In *Proceedings of the Workshop on Structured Prediction for NLP*, pages 54–62, Austin, TX. Association for Computational Linguistics.

Da Zheng, Minjie Wang, Quan Gan, Zheng Zhang, and George Karypis. 2020. Learning graph neural networks with deep graph library. WWW '20, New York, NY, USA. Association for Computing Machinery.

A Global Constraints and Mapping

Following is an example of the mapping between OWL constraint, graph structure and the logic python constraint. The ontology definition in OWL:

```
<owl:ObjectProperty rdf:ID="work_for">
    <rdfs:domain
    ↪   rdf:resource="#people"/>
    <rdfs:range
    ↪   rdf:resource="#organization"/>
</owl:ObjectProperty>
```

or equivalent graph structure definition:

```
work_for.has_a(arg1=people,
↪   arg2=organization)
```

DomiKnowS's constrain language representation:

```
ifL(work_for('x'), andL(people(path=
↪   ('x',arg1)),
↪   organization(path='x',arg2)))
```

All three above constraints represent the same knowledge that a *work_for* relationship only holds between *people* and *organization*.

In order to map this logical constrain to ILP, the solver collects sets of candidates for each used in the constrain concepts.

ILP inequalities are created for each of the combinations of candidates sets. The internal nested *andL* logical expression is translated to a set of three algebraic inequalities. The new variable *varAND*) is created to transfer the result of the internal expression into the external one.

```
varAND <= varPhraseIsPeople
varAND <= varPhraseIsOrganization
varPhraseIsPeople +
↪   varPhraseIsOrganization <= varAND
↪   + 1
```

External *ifL* expression is translated to a single algebraic inequality (refers to the variable *varAND*):

```
varPhraseIsWorkFor <= varAND
```

B Program Composition

The Program instances allow the user to define different training tasks without extra effort to change the underlying models. One can define end-to-end models, pipelines, and two step tuning paradigms just by defining different program instances and calling them one after another. For instance, we can seamlessly switch between the following variations of learning paradigms on the EMR task.
End-To-End training:

```
program = Program(graph, poi=(phrase,
↪   sentence, pair))
program.train()
```

Pre-train phrase then just train on the pairs:

```
program_1 = Program(graph,
↪   poi=(phrase, sentence))
program_2 = Program(graph,
↪   poi=(pair))
program_1.train(); program_2.train()
```

Pre-train phrase and use the result in the end-to-end training:

	Precision			Recall			F1		
	Entity	Relation	All	Entity	Relation	All	Entity	Relation	All
Baseline	0.898	0.960	0.933	0.7619	**0.807**	0.787	0.818	**0.872**	0.848
Baselin + ILP	0.896	**0.986**	**0.946**	**0.816**	0.77	**0.791**	**0.847**	0.86	**0.854**
Baseline + IML	0.8851	0.979	0.937	0.746	0.712	0.727	0.8	0.82	0.811
Baseline + PD	**0.9**	0.952	0.929	0.765	0.789	0.779	0.821	0.859	0.842

Table 1: The results on the 25% of the data on Conll benchmark.

	Precision			Recall			F1		
	Entity	Relation	All	Entity	Relation	All	Entity	Relation	All
Baseline	0.909	0.954	0.934	0.824	0.914	0.874	0.86	0.934	0.901
Baselin + ILP	**0.911**	**0.989**	**0.954**	**0.855**	0.903	**0.882**	**0.877**	**0.944**	**0.914**
Baseline + IML	0.904	**0.989**	0.951	0.831	0.884	0.861	0.86	0.933	0.901
Baseline + PD	0.910	0.934	0.923	0.827	**0.915**	0.876	0.862	0.924	0.897

Table 2: The results on 100% of data on Conll benchmark.

```
program_1 = POIProgram(graph, poi=(phrase, sentence), ...)
program_2 = POIProgram(graph, poi=(phrase, sentence, pair), ...)
program_1.train(...)
program_2.train(...)
```

C Experiments

C.1 EMR

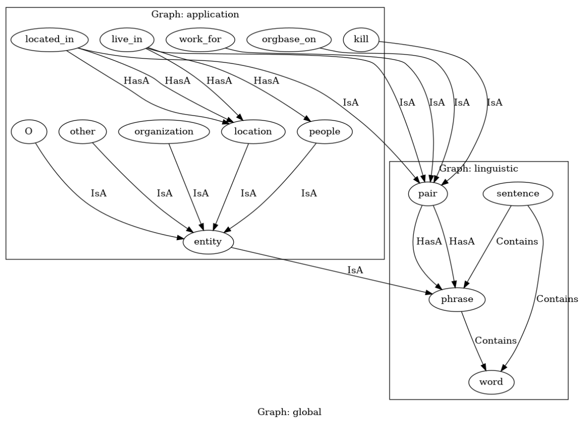

Figure 1: The domain knowledge used for the named entity and relation extraction task expressed as a graph in DomiKnowS

Figure 1 shows the prior structural domain knowledge expressed as a graph and used in this example. It contains the basic concepts such as 'sentence', 'phrase', 'word', and 'pair' and the existing relationships between them alongside the possible output concepts such as 'people' and 'work_for'. Figure 2 also represents a sample DataNode graph populated for a single phrase, alongside its properties and decisions.

Tables 1 and 2 summarize the results of the model applied on only 25% of the training data and the whole 100% of the training data based on the CONLL dataset respectively.

C.2 Question Answering

WIQA dataset contains 39,705 multiple choice questions regarding cause and effects in the context of a procedural paragraph. The answer is always either *is less*, *is more*, or *no effect*. To model this task in DomiKnowS, we define *paragraph*, *question*, *symmetric*, and *transitive* concepts. Each *paragraph* comes with a set of *questions*. As explained by Asai and Hajishirzi, enforcing constraints between different question answers can be beneficial to the models' performance. Here, two questions can be the opposite of each other with a symmetric relationship between their answers, or three questions may introduce a chain of reasoning of cause and effects leading to a transitivity property among their answers. Following is a sample constraint defined in DomiKnowS to represent the symmetric property between questions.

```
symmetric.has_a(arg1=question, arg2=question)
ifL(is_more('x'), is_less(path=('x', arg2))
```

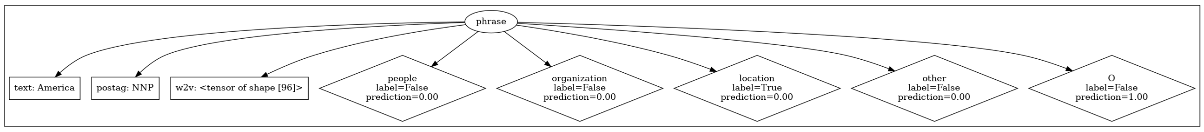
phrase
text: America
postag: NNP
w2v: <tensor of shape [96]>
people label=False prediction=0.00
organization label=False prediction=0.00
location label=True prediction=0.00
other label=False prediction=0.00
O label=False prediction=1.00

Figure 2: Sample DataNode graph populated for one single phrase from the Named Entity and Relation Extraction task.

Model	Test Accuracy
Baseline	74.22%
Baseline + IML	75.49%
Baseline + PD	76.59%
Baseline + ILP	79.05%
Baseline + PD + ILP	**80.35%**

Table 3: Results of accuracy on WIQA dataset

The results for the WIQA dataset are shown in table 3. Using IML method with this task results in 1.27% improvement while Primal Dual does a better job by improving the accuracy with 2.37%. The best result is achieved with the combination of ILP and Primal Dual with 6.1% improvement over the baseline.