DeepPSL: End-to-End Perception and Reasoning

Sridhar Dasaratha[1] , **Sai Akhil Puranam**[1] , **Karmvir Singh Phogat**[1] , **Sunil Reddy Tiyyagura**[1] , **Nigel P. Duffy**[2]

[1] EY Global Delivery Services India LLP
[2] Ernst & Young (EY) LLP USA
{Sridhar.Dasaratha, Sai.Puranam, Karmvir.Phogat, Sunil.Tiyyagura}@gds.ey.com,
Nigel.P.Duffy@ey.com

Abstract

We introduce DeepPSL a variant of probabilistic soft logic (PSL) to produce an end-to-end trainable system that integrates reasoning and perception. PSL represents first-order logic in terms of a convex graphical model – hinge-loss Markov random fields (HL-MRFs). PSL stands out among probabilistic logic frameworks due to its tractability having been applied to systems of more than 1 billion ground rules. The key to our approach is to represent predicates in first-order logic using deep neural networks and then to approximately back-propagate through the HL-MRF and thus train every aspect of the first-order system being represented. We believe that this approach represents an interesting direction for the integration of deep learning and reasoning techniques with applications to knowledge base learning, multi-task learning, and explainability. Evaluation on three different tasks demonstrates that DeepPSL significantly outperforms state-of-the-art neuro-symbolic methods on scalability while achieving comparable or better accuracy.

1 Introduction

Many machine learning problems involve rich and structured domains with numerous dependencies between its elements. Statistical relational learning (SRL) [Richardson and Domingos, 2006; Koller *et al.*, 2007] methods seek to represent these dependencies and create graphical models using rule based representations. A fundamental challenge faced by SRL approaches is balancing scalability with the expressivity of the dependency structure.

[Bach *et al.*, 2017] introduced HL-MRFs a class of probabilistic graphical models that are both tractable and expressive enabling scalable modelling of rich structured data. In addition they provide a powerful formalism, probabilistic soft logic (PSL) that can define the HL-MRF using first order logic and introduce a scalable inference algorithm. The continuous nature of HL-MRFs enable PSL to scale beyond what was previously feasible for SRL frameworks [Augustine and Getoor, 2018]. Using PSL, problems with tens of millions of ground rules have been solved in minutes [Kouki *et al.*, 2017]. Recent advances using tandem inference make inference tractable even for extremely large systems (billions of random variables) [Srinivasan *et al.*, 2020]. The PSL technique has been successfully applied to problems from various domains ranging from knowledge extraction [Rospocher, 2018], cyberattack prediction [Perera *et al.*, 2018], enrichment of product graphs [Gandoura *et al.*, 2020] to hybrid recommender systems [Rodden *et al.*, 2020].

While PSL has significantly advanced SRL methods, there have also been remarkable advances in the field of perception driven by deep learning methods. It would be highly desirable to integrate these perception capabilities into the PSL framework: however currently there is no mechanism to provide this integration. We tackle this challenge with DeepPSL, an end-to-end integration of PSL with deep learning thus achieving a major enhancement of PSL capabilities. DeepPSL fully inherits the scalability of PSL both during inference and training.

The first order expressions in PSL are built from predicates that capture the truth of an assertion with soft truth values in [0,1]. For instance, "HasClaws" and "HasStripes" could be predicates that represent whether claws and stripes are detected in an image. Given some mechanism to compute these truth values and combined with knowledge of animal attributes, PSL can infer whether a specific animal is present in the image. On the other hand neural nets (NN) can learn to identify animals directly from an image, typically from large quantities of training data.

With DeepPSL we integrate these two approaches: some of the predicates are replaced by neural nets and the input data (for e.g., text or image) is processed through a NN to generate the predicate values which are then used by PSL for inference. End-to-end training of this architecture on training data for a given task then permits the NN to learn concepts without any data on the concept itself. In contrast to PSL where one must define the predicate, the training in DeepPSL directly learns predicates that are optimized for the task at hand. Further, these concepts can now be utilized in other tasks where we may have only limited or no task specific data.

Training of this architecture poses significant challenges as it requires back-propagation through an HL-MRF that does not have continuous derivatives. This optimization problem cannot be solved by adapting existing convex optimization methods but solving it is critical to integrating deep learning

and PSL. The key contribution of our work is a novel and non-obvious approach to back-propagate through the HL-MRF to learn the parameters of these deep networks. The proposed algorithm enables end-to-end training of DeepPSL which in turn helps fully realize the benefits of the proposed architecture.

We evaluate the efficacy and performance of DeepPSL on three different tasks: digit addition, semi-supervised classification and entity resolution. Experiment results demonstrate the superior scalability of DeepPSL over other state-of-the-art neuro-symbolic approaches while achieving similar or better accuracy.

2 Related Work

Relational Neural machine (RNM) [Marra *et al.*, 2020], an extension of Deep logic models (DLM) [Marra *et al.*, 2019], model reasoning using a Markov random field and back-propagate through that field to learn underlying neural models. Unlike DeepPSL, RNMs do not require backpropagation through an $\arg\min$ and do not allow any learned values to be used directly in logical rules. Rather they add potentials that couple the learned values to output variables which must be either observed or latent in the Markov random field potentially resulting in a large increase in the number of latent variables. DLM and RNM are related to semantic-based regularization (SBR) [Diligenti *et al.*, 2017], logic tensor networks (LTN) [Donadello *et al.*, 2017] and neural logic machines [Dong *et al.*, 2019] which allow logical constraints to constrain the learning of deep networks. SBR and LTN cannot learn rule weights, and require them to be specified a priori, while DeepPSL learns both neural net weights and rule weights in an end-to-end fashion.

Neural theorem prover [Rocktäschel and Riedel, 2016] is an end- to-end differentiable prover. TensorLog [Cohen *et al.*, 2020] is a recent framework to reuse the deep learning infrastructure of TensorFlow to perform probabilistic logical reasoning. Neither of these methods model predicates using deep learning. [Hu *et al.*, 2016] presents an iterative distillation method that transfers structured information of first-order logic rules into the weights of the NNs. [Gridach, 2020] generalized the approach to include rules built using PSL.

DeepProbLog [Manhaeve *et al.*, 2018] augments the probabilistic logic programming language ProbLog [Raedt *et al.*, 2007] by incorporating neural predicates, and makes predictions by employing marginal inference and sampling. DeepProbLog and other methods such as NeurASP [Yang *et al.*, 2020] do not scale well as they rely on the computationally expensive possible world semantics. DeepStochLog [Winters *et al.*, 2022] achieves better scalability as compared to DeepProbLog and NeurASP using stochastic definite clause grammars. In contrast, DeepPSL scales to significantly larger systems with millions of ground rules by leveraging the continuous nature of HL-MRFs that cast MAP inference as a convex optimization problem. [Pryor *et al.*, 2022] propose a neuro-symbolic approach based on an energy based modeling extension of the PSL framework. Their work uses a novel "energy loss" that doesn't require back propagating through a convex optimization problem. Our approach allows for arbitrary convex differentiable loss functions including all of the most commonly used losses. A detailed discussion on neuro-symbolic techniques may be found in [Raedt *et al.*, 2020].

End-to-end training of a DeepPSL model requires solving a bi-level optimization problem. The techniques discussed in [Sinha *et al.*, 2017; Ghadimi and Wang, 2018; Dempe, 2018] for solving a bi-level optimization computes gradient of the loss function which needs computation of inverse of the Hessian that is expensive to compute at each iteration. Other methods consider neural network layers consisting of a variety of optimization problems: $\arg\min$ and $\arg\max$ problems [Gould *et al.*, 2016], quadratic programming problems [Amos and Kolter, 2017; Lee *et al.*, 2019], convex problems [Agrawal *et al.*, 2019], cone programs [Agrawal *et al.*, 2020]. All of these methods require that the optimization functions have continuous derivatives. HL-MRFs do not have continuous derivatives and therefore are not amenable to these approaches.

3 Background

3.1 HL-MRFs: Hinge Loss Markov Random Fields

HL-MRFs are defined with k continuous potentials $\boldsymbol{\phi} = \{\phi_1, \ldots, \phi_k\}$ of the form:

$$\phi_j(\boldsymbol{x}, \boldsymbol{y}) = (max\{l_j(\boldsymbol{x}, \boldsymbol{y}), 0\})^{p_j} \tag{1}$$

where ϕ_j is a potential function of n free random variables $\boldsymbol{y} = \{y_1, \ldots, y_n\}$ conditioned on n' observed random variables $\boldsymbol{x} = \{x_1, \ldots, x_{n'}\}$, where each random variable can take soft values in $[0, 1]$. The function l_j is linear in $\boldsymbol{y}$ and $\boldsymbol{x}$, and $p_j \in \{1, 2\}$. [1] Collecting the definitions from above, a hinge-loss energy function f is defined as

$$f_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{y}) = \sum_{j=1}^{k} \theta_j \phi_j(\boldsymbol{x}, \boldsymbol{y}) \tag{2}$$

where $\boldsymbol{\theta} = (\theta_1, \ldots, \theta_k)$ and θ_j is a positive weight corresponding to the potential function ϕ_j. A HL-MRF over random variables $\boldsymbol{y}$ and conditioned on random variables $\boldsymbol{x}$ is a probability density defined as

$$P(\boldsymbol{y}|\boldsymbol{x}) = \frac{1}{Z(\boldsymbol{x})} \exp\left(-f_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{y})\right) \tag{3}$$

where $Z(\boldsymbol{x})$ is the partition co-efficient. Maximum a posteriori (MAP) inference finds the most probable assignment to the free variables $\boldsymbol{y}$ given the observed variables $\boldsymbol{x}$. MAP inference is done by maximizing the probability density $P(\boldsymbol{y}|\boldsymbol{x})$ while satisfying the constraint that the random variable $\boldsymbol{y} \in [0, 1]^n$. Since the normalizing function Z in (3) is not a function of $\boldsymbol{y}$, maximizing $P(\boldsymbol{y}|\boldsymbol{x})$ is equivalent to minimizing the energy function f, i.e.,

$$\arg\max_{\boldsymbol{y} \in [0,1]^n} P(\boldsymbol{y}|\boldsymbol{x}) \equiv \arg\min_{\boldsymbol{y} \in [0,1]^n} f_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{y}) \tag{4}$$

Critically the function f is convex in $\boldsymbol{y}$, for a given $\boldsymbol{x}$, allowing for tractable inference even for very large HL-MRFs.

[1] In this work, $p_j = 2$ is used for quadratic hinge-loss.

In this study, the inference problem is solved by employing stochastic gradient descent[2] (SGD) algorithm. However, one may employ other alternative algorithms such as distributed optimization algorithm, alternating direction method of multipliers (ADMM), as discussed in [Bach *et al.*, 2017].

3.2 PSL Rules

PSL uses first order logic as a template language for HL-MRFs. A PSL program defines a set of rules in first order logic. These rules in the PSL program are grounded over the base of ground atoms each of which represents an observation or an unknown of interest. These ground atoms are associated with random variables $(\boldsymbol{x}, \boldsymbol{y})$ and can take any value in $[0, 1]$. Each ground rule is then translated into a weighted hinge-loss potential. The sum of these potentials defines a HL-MRF, and minimizing the HL-MRF conditioned over $\boldsymbol{x}$ gives values for the inferred predicates $\boldsymbol{y}$.

It is beyond the scope of this paper to provide a detailed description of how first order logic rules are used as a template language for HL-MRFs, see [Bach *et al.*, 2017] for further details.

4 DeepPSL

4.1 Deep Learning Based Predicates

The key difference between PSL and DeepPSL is that some of the predicates are modeled with deep neural networks (DNNs). In PSL, the observed predicates $\boldsymbol{x}$ are available through a knowledge base, while in DeepPSL a feature vector $\boldsymbol{u}$ is processed through DNNs to compute some of the predicates. Typically, there is no data available on these predicates to learn their corresponding DNNs. Hence, end-to-end training of DeepPSL system is needed to learn these predicates.

4.2 Learning

The key problem that needs to be solved is to determine how to train this system end-to-end. In the proposed DeepPSL framework, the features $\boldsymbol{u}$ are first processed through a neural net with tunable weights $\boldsymbol{\omega}$ to generate estimates of $\boldsymbol{x}$ which are predicates for the PSL. The estimates of predicates in the DeepPSL are modeled by a deep neural network $\boldsymbol{p}(\boldsymbol{u}; \boldsymbol{\omega})$. These predicates then go through PSL inference to produce the final values of the random variables $\boldsymbol{y}$. For end-to-end training, we need to enable backpropagation through the PSL inference. Since PSL inference is a convex optimization problem, there is no direct way to backpropagate and update the weights of the predicate network. We now describe our solution to address this problem.

Optimization Objective

We restrict our analysis here to HL-MRFs in which the inferred variables $\boldsymbol{y}$ have their corresponding true values $\hat{\boldsymbol{y}}$ in the training data. The prime objective of training this end-to-end learning model is to determine weights $\boldsymbol{w} = (\boldsymbol{\omega}, \boldsymbol{\theta})$ such that the HL-MRFs inference yields variables $\boldsymbol{y}$ close to $\hat{\boldsymbol{y}}$.

[2]The SGD optimization in PSL suffers from a drawback that hard constraints cannot be strictly enforced. However, this limitation can be circumvented by employing ADMM in place of SGD.

In order to measure if the inferred values $\boldsymbol{y}$ are close enough to the true values $\hat{\boldsymbol{y}}$, let us consider a differentiable convex loss function:

$$\mathbb{R}^n \times \mathbb{R}^n \ni (\hat{\boldsymbol{y}}, \boldsymbol{y}) \mapsto L(\hat{\boldsymbol{y}}, \boldsymbol{y}) \in \mathbb{R} \tag{5}$$

The DeepPSL inference problem (4) is approximated with soft constraints [3] and $\boldsymbol{w} = (\boldsymbol{\omega}, \boldsymbol{\theta})$ as

$$\boldsymbol{y}^* = \arg\min_{\boldsymbol{y}} \ \tilde{f}(\boldsymbol{u}, \boldsymbol{w}, \boldsymbol{y}) \tag{6}$$

where

$$\begin{aligned}\tilde{f}(\boldsymbol{u}, \boldsymbol{w}, \boldsymbol{y}) =& f_{\boldsymbol{\theta}}(\boldsymbol{p}(\boldsymbol{u}; \boldsymbol{\omega}), \boldsymbol{y}) + \sum_{i=1}^{n} \underline{\gamma_i} (\max\{0, -y_i\})^2 \\ &+ \sum_{i=1}^{n} \overline{\gamma_i} (\max\{0, y_i - 1\})^2\end{aligned}$$

with fixed $\overline{\gamma_i}, \underline{\gamma_i} > 0$. Therefore, the weight training problem is set up as a nonlinear optimization

$$\begin{aligned} &\min_{\boldsymbol{w}, \boldsymbol{y}} \ L(\hat{\boldsymbol{y}}, \boldsymbol{y}) \\ &\text{subject to} \quad \boldsymbol{y} = \arg\min_{\bar{\boldsymbol{y}}} \ \tilde{f}(\boldsymbol{u}, \boldsymbol{w}, \bar{\boldsymbol{y}}) \end{aligned} \tag{7}$$

Gradient Following Algorithm

We develop a gradient descent procedure for solving the nonlinear optimization (7). This task is challenging because we need to back-propagate through the $\arg\min$. The most direct approach involves inverting the Hessian $\nabla_{\boldsymbol{yy}} \tilde{f}(\boldsymbol{u}, \boldsymbol{w}, \boldsymbol{y})$ which is not well-defined for HL-MRFs as they do not have continuous derivatives.

We take an alternative approach which avoids this pitfall. We assume that the functions L and f are differentiable. Furthermore, we assume that the gradient $\nabla_{\boldsymbol{y}} \tilde{f}$ and the neural network $\boldsymbol{p}$ are locally Lipschitz continuous. In general, DNNs are designed to be trained using gradient based techniques and that requires DNNs to be locally Lipschitz continuous, see [Rockafellar, 1981; Scaman and Virmaux, 2018; Jordan and Dimakis, 2020]. Therefore, these assumptions are general enough and the proposed technique is applicable to a wide class of problems.

Consider the neural network weights $\boldsymbol{w}_{t-1}$ such that $\boldsymbol{y}_t = \arg\min_{\boldsymbol{y}} \tilde{f}(\boldsymbol{u}, \boldsymbol{w}_{t-1}, \boldsymbol{y})$. The objective is to find a $\boldsymbol{w}_t$ such that

$$L(\hat{\boldsymbol{y}}, \boldsymbol{y}_{t+1}) < L(\hat{\boldsymbol{y}}, \boldsymbol{y}_t) \ \text{ where } \ \boldsymbol{y}_{t+1} = \arg\min_{\boldsymbol{y}} \tilde{f}(\boldsymbol{u}, \boldsymbol{w}_t, \boldsymbol{y})$$

To this end, we first linearly approximate the constraint $\tilde{\boldsymbol{y}}_{t+1} = \arg\min_{\boldsymbol{y}} \tilde{f}(\boldsymbol{u}, \boldsymbol{w}, \boldsymbol{y})$ in the neighborhood of $\boldsymbol{y}_t$ by using continuous dependence[4] of $\tilde{\boldsymbol{y}}_{t+1}$ on $\boldsymbol{w}$ and that is given by

$$\tilde{\boldsymbol{y}}_{t+1} = \boldsymbol{y}_t - \delta \nabla_{\boldsymbol{y}} \tilde{f}(\boldsymbol{u}, \boldsymbol{w}, \boldsymbol{y}_t) \tag{8}$$

[3]The constraints in (4) are incorporated using Lagrange multipliers in a fairly standard way.

[4]Please note that, in general, convexity of a differentiable function f is not sufficient to ensure continuous dependence of $\arg\min_{\boldsymbol{y}} \tilde{f}(\boldsymbol{u}, \boldsymbol{w}, \boldsymbol{y})$ on $\boldsymbol{w}$. However, $\arg\min_{\boldsymbol{y}} \tilde{f}(\boldsymbol{u}, \boldsymbol{w}, \boldsymbol{y}) + \nu ||\boldsymbol{y} - \boldsymbol{y}_t||^2$ for any $\nu > 0$ is augmented to ensure uniqueness of the solution and so continuous dependence on $\boldsymbol{w}$.

for sufficiently small $\delta > 0$. Therefore, using the approximation (8), the optimization (7), in the neighborhood of $\boldsymbol{y}_t$, translates to

$$\arg\min_{\boldsymbol{w}} L(\hat{\boldsymbol{y}}, \boldsymbol{y}_t - \delta \nabla_{\boldsymbol{y}} \tilde{f}(\boldsymbol{u}, \boldsymbol{w}, \boldsymbol{y}_t)) \tag{9}$$

and that is linearly approximated, in the neighborhood of $\boldsymbol{y}_t$, to

$$\boldsymbol{w}_t = \arg\max_{\boldsymbol{w}} \nabla_{\boldsymbol{y}} \tilde{f}(\boldsymbol{u}, \boldsymbol{w}, \boldsymbol{y}_t) \cdot \nabla_{\boldsymbol{y}} L(\hat{\boldsymbol{y}}, \boldsymbol{y}_t) \tag{10}$$

It is worth noting that if $\nabla_{\boldsymbol{y}} \tilde{f}(\boldsymbol{u}, \boldsymbol{w}_t, \boldsymbol{y}_t) \cdot \nabla_{\boldsymbol{y}} L(\hat{\boldsymbol{y}}, \boldsymbol{y}_t) > 0$ then $L(\hat{\boldsymbol{y}}, \tilde{\boldsymbol{y}}_{t+1}) < L(\hat{\boldsymbol{y}}, \boldsymbol{y}_t)$. On the other hand if $\nabla_{\boldsymbol{y}} \tilde{f}(\boldsymbol{u}, \boldsymbol{w}_t, \boldsymbol{y}_t) \cdot \nabla_{\boldsymbol{y}} L(\hat{\boldsymbol{y}}, \boldsymbol{y}_t) \leq 0$ then local optimality is attained at $\boldsymbol{w}_{t-1}$. Furthermore, to ensure constraint satisfaction at each iteration, the local linear approximation $\tilde{\boldsymbol{y}}_{t+1}$ is replaced with the inference optimization

$$\boldsymbol{y}_{t+1} = \arg\min_{\boldsymbol{y}} \tilde{f}(\boldsymbol{u}, \boldsymbol{w}_t, \boldsymbol{y}) \tag{11}$$

Recall that $\lim_{h \to 0} \frac{g(v+hz) - g(v)}{h} = \nabla g(v) \cdot z$ and therefore, (10) can be rewritten as

$$\boldsymbol{w}_t = \arg\min_{\boldsymbol{w}} \ \tilde{f}(\boldsymbol{u}, \boldsymbol{w}, \boldsymbol{y}_t - \alpha \nabla_{\boldsymbol{y}} L(\hat{\boldsymbol{y}}, \boldsymbol{y}_t)) - \tilde{f}(\boldsymbol{u}, \boldsymbol{w}, \boldsymbol{y}_t) \tag{12}$$

for sufficiently small $\alpha > 0$.

Regularization

A PSL potential ϕ_j corresponding to a rule j, defined in (1), translates in DeepPSL setup to

$$\phi_j(\boldsymbol{u}, \boldsymbol{\omega}, \boldsymbol{y}) = \max\left(l_j(\boldsymbol{p}(\boldsymbol{\omega}; \boldsymbol{u}), \boldsymbol{y}), 0\right)^{p_j}$$

where l_j is a linear function, $p_j \in \{1, 2\}$, $\boldsymbol{p}(\boldsymbol{\omega}; \boldsymbol{u})$ is a neural network with weights $\boldsymbol{\omega}$ and input $\boldsymbol{u}$. Note that, for a certain neural network weights $\tilde{\boldsymbol{\omega}}$, the potential ϕ_j does not trigger, i.e.,

$$\phi_j(\bar{\boldsymbol{u}}, \tilde{\boldsymbol{\omega}}, \bar{\boldsymbol{y}}) = 0 \quad \text{for any} \quad \bar{\boldsymbol{u}} \in \mathcal{D} \text{ and } \bar{\boldsymbol{y}} \in [0, 1]^n$$

where $\mathcal{D}$ is a collection of features from training dataset, and therefore, there will be no updates to the neural network weights $\tilde{\boldsymbol{\omega}}$ from the potential ϕ_j. The lack of weight updates might lead to locally optimal solutions to the training optimization (7), and that may be avoided by adding a penalty term to the optimization (12) for each potential ϕ_j as

$$\psi_j(\boldsymbol{u}, \boldsymbol{\omega}, \boldsymbol{y}) = \mu \left(l_j(\boldsymbol{p}(\boldsymbol{u}; \boldsymbol{\omega}), \boldsymbol{y})\right)^2 \quad \text{with} \quad \mu > 0 \tag{13}$$

The regularizer, defined in (13), ensures that whenever the hinge loss potential ϕ_j becomes zero the corresponding regularizer term ψ_j remains non-zero, thus ensuring neural weight updates and avoiding trivial solutions. The optimization (12) with the regularizer is given by

$$\begin{aligned} \boldsymbol{w}_t = \arg\min_{\boldsymbol{w}} \ & \tilde{f}(\boldsymbol{u}, \boldsymbol{w}, \boldsymbol{y}_t - \alpha \nabla_{\boldsymbol{y}} L(\hat{\boldsymbol{y}}, \boldsymbol{y}_t)) \\ & - \tilde{f}(\boldsymbol{u}, \boldsymbol{w}, \boldsymbol{y}_t) + \mu \Omega(\boldsymbol{u}, \boldsymbol{w}, \boldsymbol{y}_t) \end{aligned} \tag{14}$$

where

$$\Omega(\boldsymbol{u}, \boldsymbol{w}, \boldsymbol{y}_t) = \sum_{j=1}^{k} \psi_j(\boldsymbol{u}, \boldsymbol{\omega}, \boldsymbol{y}_t - \alpha \nabla_{\boldsymbol{y}} L(\hat{\boldsymbol{y}}, \boldsymbol{y}_t)) - \psi_j(\boldsymbol{u}, \boldsymbol{\omega}, \boldsymbol{y}_t)$$

for k potentials, and $\mu > 0$.

These two optimizations (11) and (14) are executed alternatively until convergence in Algorithm 1.

Algorithm 1 Joint optimization: backpropagating loss to the neural network

Initialization: $t = 1; \alpha, \eta, \mu > 0; N, T \geq 1$.

Neural network weights $\boldsymbol{w}_0$ are initialized using standard techniques.

while $t \leq T$ **do**

  $\boldsymbol{y}_t = \arg\min_{\boldsymbol{y}} \ \tilde{f}(\boldsymbol{u}, \boldsymbol{w}_{t-1}, \boldsymbol{y})$ {MAP inference}

  $\boldsymbol{w}_t = \boldsymbol{w}_{t-1}$

  for $i = 1, \ldots, N$ **do**

    $\boldsymbol{w}_t \mathrel{-}= \eta \nabla_{\boldsymbol{w}_t} \big[\tilde{f}(\boldsymbol{u}, \boldsymbol{w}_t, \boldsymbol{y}_t - \alpha \nabla_{\boldsymbol{y}} L(\hat{\boldsymbol{y}}, \boldsymbol{y}_t)) - \tilde{f}(\boldsymbol{u}, \boldsymbol{w}_t, \boldsymbol{y}_t) + \mu \Omega(\boldsymbol{u}, \boldsymbol{w}_t, \boldsymbol{y}_t)\big]$

  end for

  $t = t + 1$

end while

4.3 Scalability

DeepPSL fully inherits the scalability of PSL, that is, they have the same Big O behavior. The first step in DeepPSL inference consists of a forward pass through the NNs to compute the groundings, only adding time linear in number of ground terms. The second step is standard PSL inference, that is extremely efficient and scales linearly in number of potentials, see Section 3.1.

DeepPSL training requires alternate execution of gradient descent step for minimizing loss (14) and an inference step. The gradient descent step can become memory intensive. As the loss function can be rewritten as summation of losses for each of the grounding, we address the problem using gradient accumulation. Weight updates are made after accumulating gradients from all the grounded potentials. Preserving the inherent scalability of PSL inference and addressing the challenges during training makes DeepPSL highly scalable.

5 Experimental Evaluation

We evaluate DeepPSL on three tasks – T1: a digit addition task that shows that DeepPSL can learn predicates that are hard to specify in PSL, T2: a document classification relational problem that is of moderate scale and T3: a challenging large scale entity resolution problem. We compare our method with state-of-the-art neuro-symbolic methods and in some cases, with other methods that are more specific to the task. Each of these tasks contains train, validation and test splits in the corresponding dataset(s). We select the model and hyper-parameters, see Table 1, that give the highest performance on the validation set. The selected model is then evaluated on the test set to report the performance. The performance metric is reported with a 95% confidence interval, calculated by repeating each experiment multiple times. We report "timeout" when a single training run exceeds an hour as defined in [Winters *et al.*, 2022] or getting an out of memory error. The experiments are performed on a MacBook Pro with 2.6GHz Intel i7 processor having 6 cores.

Inference parameters	T1	T2	T3
Optimizer	SGD	SGD	SGD
Learning rate	$5e{-3}$	$5e{-3}$	$1e{-2}$
Loss change threshold	$1e{-6}$	$5e{-3}$	$1e{-1}$
Max iterations	5000	1000	5000
$\underline{\gamma_i}$, $\overline{\gamma_i}$	20	20	20
Training Parameters *			
Optimizer	Adagrad	Adam	Adagrad
α	$5e{-5}$	1	$5e{-3}$
μ	$1e{-3}$	$12e{-2}$	0
Update steps (N)	2	10	2
Epochs (T)	10	75	100

* Rule weights are initialized randomly by drawing samples from a normal distribution $\mathcal{N}(1.0, 0.1)$.

Table 1: Hyperparameters used for DeepPSL

5.1 T1: Addition of Handwritten Digits

The goal of this task is to predict the sum of digits present in two MNIST images[5] [Manhaeve *et al.*, 2018]. Note that the training data provides only the sum of the digits, and the digit labels are not provided. Hence, it is not possible to directly learn or specify a predicate to classify individual images, making it difficult for PSL to solve this problem. We investigate whether DeepPSL can learn the image predicate and achieve performance comparable to other neuro-symbolic approaches.

The datasets for this problem are generated following the procedure described in [Winters *et al.*, 2022]. Rules provided in Figure 1 are used to add two digits in DeepPSL system. We use the predicate DIGIT, a convolutional neural network (CNN), to recognize the digits present in the input images (`Img1`, `Img2`). The inferred predicate SUM represents the sum of digits present in the images. DeepPSL is trained by minimizing the cross-entropy loss between the inferred predicates $\textsc{Sum}(\texttt{S})$ and the ground truth.

The CNN consists of two convolution (CONV) layers with 32 and 64 filters of size 3 and stride 1 with ELU activation. Max pool layer with size 2 is applied on the output of last CONV layer. This is followed by two fully connected layers with 128 and 10 nodes on which ELU and softmax activations are applied respectively. A batch size of 16 is used for training. The learning rate for rule weights and CNN are decayed for the 10 epochs according to $\eta = \eta_0/(E+1)$ where, η_0 ($5e{-4}$ for rule weights and $1e{-3}$ for CNN parameters) is the initial learning rate and E is the epoch number (starting with 0). A weight decay of $1e{-7}$ is used for CNN parameters from the second epoch of training. Weight decay is not used for rule weights. In the ruleset, the summation constraint is implemented as a soft constraint with a fixed weight 10.

The average classification accuracy of DeepPSL over 10 runs is shown in Table 2. The performance numbers for other methods are obtained from [Winters *et al.*, 2022]. We also evaluate the CNN corresponding to DIGIT predicate on the test split of MNIST data set. The CNN achieves an accuracy of 98.2% demonstrating that DeepPSL could successfully learn the predicate even without any explicit data and thus achieves performance comparable to other neuro-symbolic methods.

[5] A detailed explanation on digit addition problem may be found in [Manhaeve *et al.*, 2018].

 # Addition of handwritten digits rules

θ_1 :$\textsc{Digit}(\texttt{Img1}, \texttt{D1}) \wedge \textsc{Digit}(\texttt{Img2}, \texttt{D2}) \wedge (\texttt{Img1}\neg = \texttt{Img2})$
$\rightarrow \textsc{Sum}(\texttt{S})\hat{}2$

θ_2 :$\textsc{Digit}(\texttt{Img1}, \texttt{D1}) \wedge \neg\textsc{Digit}(\texttt{Img2}, \texttt{D2}) \wedge (\texttt{Img1}\neg = \texttt{Img2})$
$\rightarrow \neg\textsc{Sum}(\texttt{S})\hat{}2$

 # Summation constraint

$\textsc{Sum}(+\texttt{S}) = 1$

Figure 1: DeepPSL rule set for addition of handwritten digits

NeurASP	DeepProbLog	DeepStochLog	DeepPSL
97.3 ± 0.3	97.2 ± 0.5	97.9 ± 0.1	96.2 ± 0.2

Table 2: Test accuracy on addition of handwritten digits

To study the robustness of DeepPSL to varying sizes of training data, we evaluate the DeepPSL performance at three different sample sizes, see Table 3. Even with a ten-fold reduction in size of training set, the decrease in accuracy is only marginal.

Samples	30000	10000	3000
Test accuracy	96.2 ± 0.2	94.8 ± 0.5	90.4 ± 1.5

Table 3: Effect of training data size on DeepPSL performance

5.2 T2: Semi-Supervised Classification

The goal is to classify unlabeled documents in citation networks given some documents that are labeled. The problem is more challenging than task T1 as there is significantly more relational information available, which in turn poses a challenge to the scalability of neuro-symbolic methods. We use data from the Cora and Citeseer scientific datasets [Yang *et al.*, 2016]. The Cora dataset contains 2708 documents in 7 categories, with 5429 citation links, and each document is represented by indicating the absence or presence of the corresponding word from a dictionary of 1433 unique words. Similarly, the Citeseer dataset contains 3327 documents in 6 categories, with 4732 citation links, and each document is represented by indicating the absence or presence of the corresponding word from a dictionary of 3703 unique words.

The predicate $\textsc{Cite}(\texttt{A}, \texttt{B})$ defines a citation from node `A` to node `B`, and the number of neighbors of `A` are $n_{\texttt{A}} = |\{\texttt{B} \mid \textsc{Cite}(\texttt{B}, \texttt{A})\}|$. Furthermore, we define 2-hop neighbors and 3-hop neighbors as, for any $\texttt{A}\neg = \texttt{B}$,

$$\textsc{CiteP}(\texttt{A}, \texttt{B}) = \textsc{Cite}(\texttt{A}, \texttt{C}) \wedge \textsc{Cite}(\texttt{C}, \texttt{B}) \text{ for any node } \texttt{C},$$
$$\textsc{CiteQ}(\texttt{A}, \texttt{B}) = \textsc{CiteP}(\texttt{A}, \texttt{C}) \wedge \textsc{Cite}(\texttt{C}, \texttt{B}) \text{ for any node } \texttt{C}.$$

Dataset	Nodes	Split 1 (Train/ Val/ Test)	Split 2 (Train/ Val/ Test)
Cora	2708	140/ 500/ 1000	1708/ 500/ 500
Citeseer	3327	120/ 500/ 1000	2327/ 500/ 500

Table 4: Dataset splits for semi-supervised classification task

In the rule set shown in Figure 2, the predicates CITE, CITEP and CITEQ are directly observed from the data. The inferred predicate LABEL identifies the labels of the documents. The deep learning predicates NEURAL and SIMILAR represent the neural net classifier output and the similarity between documents, respectively.

 # Semi-supervised classification rules

θ_1 :NEURAL(A, Y) $\rightarrow$ LABEL(A, Y)^2

$\frac{\theta_2}{n_\text{A}}$:LABEL(B, Y) $\wedge$ SIMILAR(B, A) $\wedge$ CITE(B, A) $\rightarrow$ LABEL(A, Y)^2

$\frac{\theta_3}{n_\text{A}}$:LABEL(B, Y) $\wedge \neg$SIMILAR(B, A) $\wedge$ CITE(B, A) $\rightarrow \neg$LABEL(A, Y)^2

θ_4 : $\frac{1}{|\text{B}|}$NEURAL(+B, Y) $\rightarrow$ LABEL(A, Y){B : CITE(A, B)}^2

θ_5 : $\frac{1}{|\text{B}|}$NEURAL(+B, Y) $\rightarrow$ LABEL(A, Y){B : CITEP(A, B)}^2

θ_6 : $\frac{1}{|\text{B}|}$NEURAL(+B, Y) $\rightarrow$ LABEL(A, Y){B : CITEQ(A, B)}^2

 # Constraints

LABEL(A, +Y) = 1

Figure 2: DeepPSL rule set for semi-supervised classification

The predicate NEURAL is represented by a feedforward neural network with an input layer that takes the document features, followed by a hidden layer with 16 nodes (RELU activation), a drop out layer with a rate of 0.2, and a final softmax layer with 7 nodes corresponding to the output classes. The predicate SIMILAR is represented by a Siamese network composed of two identical networks that share the first layer of the feedforward network, and a distance layer with 16 nodes. We minimize cross entropy loss and use learning rate $2\text{e}{-3}$ for rule weights and $2\text{e}{-2}$ for NN parameters. The weight decay parameter of Adam optimizer is set to $3\text{e}{-4}$ (Split 1) and $3\text{e}{-5}$ (Split 2), and is only used for NN parameters. The summation constraint is implemented as a soft constraint with a fixed weight 20.

We compare the performance of DeepPSL against various baselines on the Cora and Citeseer datasets, using randomly generated data splits described in Table 4. The results, presented in Table 5, show classification class averaged accuracy on the test sets. The results for GCN[6] [Kipf and Welling, 2017], PSL and DeepPSL are generated by running each method 100 times, while the results for ManiReg [Belkin *et al.*, 2006], SemiEmb [Weston *et al.*, 2012], LP [Zhu *et al.*, 2003], DeepWalk [Perozzi *et al.*, 2014], ICA [Lu and Getoor, 2003], Planetoid [Yang *et al.*, 2016] are taken from [Kipf and Welling, 2017]. As DeepStochLog training was slow, we ran it only 5 times and don't report error bars. All the models are evaluated on Split 1, while GCN, PSL and DeepPSL are evaluated on both Split 1 and Split 2.

Algorithm	Cora	Citeseer
ManiReg	59.5	60.1
SemiEmb	59.0	59.6
LP	68.0	45.3
DeepWalk	67.2	43.2
ICA	75.1	69.1
Planetoid	75.7	64.7
DeepStochLog	69.4	65
DeepProbLog	timeout	timeout
GCN	80.08 ± 0.34	67.96 ± 0.32
PSL	62.97 ± 0.52	64.88 ± 0.38
DeepPSL	$\mathbf{81.31 \pm 0.28}$	$\mathbf{69.11 \pm 0.27}$
GCN(Split 2)	87.46 ± 0.30	75.96 ± 0.34
PSL(Split 2)	85.94 ± 0.28	75.66 ± 0.32
DeepPSL(Split 2)	$\mathbf{88.94 \pm 0.25}$	$\mathbf{76.01 \pm 0.31}$

Table 5: Classification accuracy on test nodes for Task T2

DeepPSL achieves the highest accuracy for both data sets and both splits, outperforming DeepStochLog by a large margin and surpassing even methods that are specialized for semi-supervised learning. The expressivity of PSL for relational systems enables DeepPSL to leverage the labels and features of its neighbors, while the end-to-end training helps optimize the weights of the neural predicates to maximize classification accuracy. When compared to other neuro-symbolic methods, DeepPSL demonstrates excellent scalability. DeepProbLog timed out for both Cora and Citeseer datasets while DeepStochLog training time was $\sim$ 50x that of DeepPSL.

5.3 T3: Entity Resolution

We perform entity resolution on Citeseer database [Bhattacharya and Getoor, 2007] using DeepPSL to identify duplicate references to authors and published papers. As the task requires predicting the edges between every pair of author nodes and paper nodes in a large network, the problem results in millions of ground rules posing a major challenge for existing neuro-symbolic approaches.

The data consists of author names, paper titles and relational information such as authorship of papers. There are around 3000 author references and 1500 paper references. We use the train and test splits provided in PSL entity resolution example.[7] We extract a third of data in the provided train set to create a validation set.

For this problem, pair-wise similarity of author names and pair-wise similarity of paper titles provide key information to identify duplicates. Traditionally, this similarity is computed with hand crafted metrics and is used for inference. It has been shown that the performance of different string similarity metrics vary greatly based on the application domain

[6]https://github.com/tkipf/gcn

[7]https://github.com/linqs/psl-examples/tree/master/entity-resolution

 # Entity resolution rules

θ_1 :AUTHORNAME(A1, N1) ∧ AUTHORNAME(A2, N2) ∧ SIMNAME(N1, N2) → SAMEAUTHOR(A1, A2)^2

θ_2 :PAPERTITLE(P1, T1) ∧ PAPERTITLE(P2, T2) ∧ SIMTITLE(T1, T2) → SAMEPAPER(P1, P2)^2

θ_3 :AUTHORBLK(A1, B1) ∧ AUTHORBLK(A2, B1) ∧ AUTHOROF(A1, P1)∧ AUTHOROF(A2, P2) ∧ SAMEPAPER(P1, P2) → SAMEAUTHOR(A1, A2)^2

θ_4 :AUTHORBLK(A1, B) ∧ AUTHORBLK(A2, B) ∧ AUTHORBLK(A3, B)∧ SAMEAUTHOR(A1, A2) ∧ SAMEAUTHOR(A2, A3) ∧ (A1¬ = A3)∧ (A2¬ = A3) ∧ (A1¬ = A2) → SAMEAUTHOR(A1, A3)^2

θ_5 :¬SAMEAUTHOR(A1, A2)^2

θ_6 :¬SAMEPAPER(P1, P2)^2

 # Identity constraints

SAMEAUTHOR(A, A) = 1

SAMEPAPER(P, P) = 1

Figure 3: DeepPSL rule set for entity resolution

[Cheatham and Hitzler, 2013]. Instead of relying on predefined string similarity metrics, DeepPSL learns the optimal similarity function for the task.

Rules[7] in Figure 3 incorporate the author name and paper title similarity values in conjunction with other relational information to identify duplicates. The inferred predicates SAMEAUTHOR and SAMEPAPER identify if authors and papers are same respectively. AUTHORNAME, PAPERTITLE, AUTHOROF and AUTHORBLK are directly observed from the data. In the DeepPSL system, we associate author name similarity predicate SIMNAME and title similarity predicate SIMTITLE with siamese networks [Koch *et al.*, 2015]. The NN to compute author name similarity takes 56 dimensional character ([A-Za-z .,']) based one hot encoding of the names as input. NN for title similarity operates on mean of 300 dimensional vectors derived from GloVe embeddings of the words in the title. We use the embeddings that have been pre-trained on Wikipedia 2014 and Gigaword5 corpus.[8] Each twin in the architecture has a hidden layer of 50 nodes and distance layer of 50 nodes for both NNs. We minimize cross entropy loss using learning rates of $5e{-}2$ for NN weights and $25e{-}3$ for rule weights. During training of the DeepPSL system, we do not perform any sampling over the graph. A weight decay of $1e{-}3$ is used only for the NN weights. The identity constraints are implemented with a fixed weight 10. The model which gives the highest average of F1-scores for same author and same paper identification is selected based on validation set.

After trivial potential removal, there are 6.5 million and 3.1 million ground rules during test and train respectively. The scale of this relational problem is out of scope for neural probabilistic logic programming approaches such as DeepProbLog. We attempted a comparison with DeepStochLog[9]. However, DeepStochLog timed out even when using only a subset of the rules. We compare performance of PSL and DeepPSL using the same ruleset. For PSL, the author name and paper title similarities are computed using metrics mentioned in [Bhattacharya and Getoor, 2007]. PSL-A uses Soft TFIDF with Jaro-Winkler [Cohen *et al.*, 2003] for both author name and paper title similarity. PSL-B uses Soft TFIDF for author names and cosine similarity of GloVe based vectors for paper titles. PSL-C, PSL-J and PSL-M use cosine [Cheatham and Hitzler, 2013], Jaccard [Gali *et al.*, 2016], and Monge Elkan (with Levenshtein) [Yih and Meek, 2007] respectively, for both author name and paper title similarity. For computing these metrics, strings are not pre-processed apart from tokenization when necessary.

Algorithm	Author F1 score	Paper F1 score
PSL-A	0.9271	0.8276
PSL-B	0.8958	0.8501
PSL-C	0.7852	0.7656
PSL-J	0.8073	0.7631
PSL-M	0.8008	0.8884
DeepStochLog	timeout	timeout
DeepPSL	$\mathbf{0.9468 \pm 0.0061}$	$\mathbf{0.9228 \pm 0.0036}$

Table 6: Performance on Task T3

The comparison of PSL and DeepPSL over 10 runs is shown in Table 6. Error bars are not reported for PSL as there is insignificant variation across runs. DeepPSL outperforms all the PSL setups for both author and paper entity resolution. The PSL performance depends on the pre-determined similarity metric used and it is hard to find a similarity metric that yields the highest accuracy. DeepPSL achieves excellent performance by directly learning optimal similarity functions for the entity resolution task. Moreover, these results highlight that DeepPSL scales to a difficult problem which proves to be out of reach for competing methods.

6 Conclusions and Future Work

We introduced DeepPSL an end-to-end trainable system that integrates reasoning and perception. We proposed a novel algorithm to enable end-to-end training. Experimental results on three different tasks demonstrated the broad applicability of the method. DeepPSL scales to relational problems that prove to be challenging for competing methods. DeepPSL has some limitations. Firstly, DeepPSL learning is not convex and therefore subject to local minima. While we did not observe any significant sensitivity to local minima in our experiments, further research is needed to understand this better. Secondly, the work described in this article does not address latent variables. Future work will report on extensions of DeepPSL to the latent variable case.

Disclaimer

The views reflected in this article are the views of the authors and do not necessarily reflect the views of the global EY organization or its member firms.

References

[Agrawal *et al.*, 2019] Akshay Agrawal, Brandon Amos, Shane Barratt, Stephen Boyd, Steven Diamond, and

[8] https://nlp.stanford.edu/projects/glove

[9] https://github.com/MLKULeuven/deepstochlog/tree/main/examples.

J. Zico Kolter. Differentiable Convex Optimization Layers. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates Inc., 2019.

[Agrawal *et al.*, 2020] Akshay Agrawal, Shane Barratt, Stephen Boyd, Enzo Busseti, and Walaa M. Moursi. Differentiating Through a Cone Program. *arXiv; 1904.09043*, 2020.

[Amos and Kolter, 2017] Brandon Amos and J Zico Kolter. Optnet: Differentiable Optimization as a Layer in Neural Networks. In *International Conference on Machine Learning*, pages 136–145. PMLR, 2017.

[Augustine and Getoor, 2018] E. Augustine and L. Getoor. A Comparison of Bottom-Up Approaches to Grounding for Templated Markov Random Fields. In *SysML*, 2018.

[Bach *et al.*, 2017] Stephen H Bach, Matthias Broecheler, Bert Huang, and Lise Getoor. Hinge-Loss Markov Random Fields and Probabilistic Soft Logic. *Journal of Machine Learning Research*, 18:1–67, 2017.

[Belkin *et al.*, 2006] Mikhail Belkin, Partha Niyogi, and Vikas Sindhwani. Manifold regularization: A geometric framework for learning from labeled and unlabeled examples. *Journal of Machine Learning Research*, 7(11), 2006.

[Bhattacharya and Getoor, 2007] Indrajit Bhattacharya and Lise Getoor. Collective Entity Resolution In Relational Data. *ACM Transactions on Knowledge Discovery from Data*, pages 1–36, 2007.

[Cheatham and Hitzler, 2013] Michelle Cheatham and Pascal Hitzler. String Similarity Metrics for Ontology Alignment. In *The Semantic Web – ISWC 2013*, pages 294–309, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[Cohen *et al.*, 2003] W. Cohen, P. Ravikumar, and S. Fienberg. A Comparison of String Distance Metrics for Name-Matching Tasks. In *The IJCAI Workshop on Information Integration on the Web (IIWeb)*, 2003.

[Cohen *et al.*, 2020] William W. Cohen, Fan Yang, and Kathryn Mazaitis. TensorLog: A Probabilistic Database Implemented Using Deep-Learning Infrastructure. *Journal of Artificial Intelligence Research*, 67:285–325, 2020.

[Dempe, 2018] Stephan Dempe. *Bilevel Optimization: Theory, Algorithms and Applications*. TU Bergakademie Freiberg, Fakultät für Mathematik und Informatik, 2018.

[Diligenti *et al.*, 2017] Michelangelo Diligenti, Marco Gori, and Claudio Saccà. Semantic-based regularization for learning and inference. *Artificial Intelligence*, 244:143–165, 2017.

[Donadello *et al.*, 2017] Ivan Donadello, Luciano Serafini, and Artur D'Avila Garcez. Logic Tensor Networks for Semantic Image Interpretation. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence*, page 1596–1602. AAAI Press, 2017.

[Dong *et al.*, 2019] Honghua Dong, Jiayuan Mao, Tian Lin, Chong Wang, Lihong Li, and Denny Zhou. Neural Logic Machines. In *International Conference on Learning Representations*, 2019.

[Gali *et al.*, 2016] Najlah Gali, Radu Mariescu-Istodor, and Pasi Franti. Similarity Measures for Title Matching. In *International Conference on Pattern Recognition (ICPR)*, 2016.

[Gandoura *et al.*, 2020] Marouene Sfar Gandoura, Zografoula Vagena, and Nikolaos Vasiloglou. Human in the Loop Enrichment of Product Graphs with Probabilistic Soft Logic. In *Proceedings of Knowledge Graphs and E-commerce, KDD 20*, 2020.

[Ghadimi and Wang, 2018] Saeed Ghadimi and Mengdi Wang. Approximation Methods for Bilevel Programming. *arXiv preprint arXiv:1802.02246*, 2018.

[Gould *et al.*, 2016] Stephen Gould, Basura Fernando, Anoop Cherian, Peter Anderson, Rodrigo Santa Cruz, and Edison Guo. On Differentiating Parameterized Argmin and Argmax Problems with Application to Bi-level Optimization. *arXiv; 1607.05447*, 2016.

[Gridach, 2020] Mourad Gridach. A framework based on (probabilistic) soft logic and neural network for NLP. *Applied Soft Computing*, 93:106232, 2020.

[Hu *et al.*, 2016] Zhiting Hu, Xuezhe Ma, Zhengzhong Liu, Eduard Hovy, and Eric Xing. Harnessing Deep Neural Networks with Logic Rules. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2410–2420. Association for Computational Linguistics, August 2016.

[Jordan and Dimakis, 2020] Matt Jordan and Alexandros G Dimakis. Exactly Computing the Local Lipschitz Constant of ReLU Networks. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 7344–7353, 2020.

[Kipf and Welling, 2017] Thomas N. Kipf and Max Welling. Semi-Supervised Classification with Graph Convolutional Networks. In *5th International Conference on Learning Representations, ICLR 2017*. OpenReview.net, 2017.

[Koch *et al.*, 2015] Gregory Koch, Richard Zemel, and Ruslan Salakhutdinov. Siamese Neural Networks for One-shot Image Recognition. In *ICML 2015 Deep Learning Worshop*, 2015.

[Koller *et al.*, 2007] Daphne Koller, Nir Friedman, Lise Getoor, and Ben Taskar. Graphical Models in a Nutshell. *Introduction to statistical relational learning*, 43, 2007.

[Kouki *et al.*, 2017] Pigi Kouki, Jay Pujara, Christopher Marcum, Laura Koehly, and Lise Getoor. Collective entity resolution in familial networks. In *2017 IEEE International Conference on Data Mining (ICDM)*, pages 227–236. IEEE, 2017.

[Lee *et al.*, 2019] Kwonjoon Lee, Subhransu Maji, Avinash Ravichandran, and Stefano Soatto. Meta-Learning with Differentiable Convex Optimization. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10657–10665, 2019.

[Lu and Getoor, 2003] Qing Lu and Lise Getoor. Link-based classification. In *Proceedings of the 20th International*

Conference on Machine Learning, pages 496–503. AAAI Press, 2003.

[Manhaeve *et al.*, 2018] Robin Manhaeve, Sebastijan Dumancic, Angelika Kimmig, Thomas Demeester, and Luc De Raedt. DeepProbLog: Neural Probabilistic Logic Programming. In *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

[Marra *et al.*, 2019] Giuseppe Marra, Francesco Giannini, Michelangelo Diligenti, and Marco Gori. Integrating Learning and Reasoning with Deep Logic Models. In *Machine Learning and Knowledge Discovery in Databases - European Conference, 2019, Proceedings, Part II*, volume 11907, pages 517–532. Springer, 2019.

[Marra *et al.*, 2020] Giuseppe Marra, Michelangelo Diligenti, Francesco Giannini, Marco Gori, and Marco Maggini. Relational Neural Machines. In *24th European Conference on Artificial Intelligence*, 2020.

[Perera *et al.*, 2018] Ian Perera, Jena Hwang, Kevin Bayas, Bonnie Dorr, and Yorick Wilks. Cyberattack Prediction Through Public Text Analysis and Mini-Theories. In *2018 IEEE International Conference on Big Data (Big Data)*, pages 3001–3010. IEEE, 2018.

[Perozzi *et al.*, 2014] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 701–710, 2014.

[Pryor *et al.*, 2022] Connor Pryor, Charles Dickens, Eriq Augustine, Alon Albalak, William Wang, and Lise Getoor. NeuPSL: Neural Probabilistic Soft Logic. *arXiv preprint arXiv:2205.14268*, 2022.

[Raedt *et al.*, 2007] Luc De Raedt, Angelika Kimmig, and Hannu Toivonen. ProbLog: A Probabilistic Prolog and Its Application in Link Discovery. In *30th International Joint Conference on Artificial Intelligence*, pages 2462–2467, 2007.

[Raedt *et al.*, 2020] Luc de Raedt, Sebastijan Dumančić, Robin Manhaeve, and Giuseppe Marra. From Statistical Relational to Neuro-Symbolic Artificial Intelligence. In *29th International Joint Conference on Artificial Intelligence*, pages 4943–4950, 2020.

[Richardson and Domingos, 2006] Matthew Richardson and Pedro Domingos. Markov Logic Networks. *Machine Learning*, 62(1–2):107–136, February 2006.

[Rockafellar, 1981] RT Rockafellar. Favorable Classes of Lipschitz Continuous Functions in Subgradient Optimization. 1981.

[Rocktäschel and Riedel, 2016] Tim Rocktäschel and Sebastian Riedel. Learning Knowledge Base Inference with Neural Theorem Provers. In *Proceedings of the 5th Workshop on Automated Knowledge Base Construction*, pages 45–50. Association for Computational Linguistics, June 2016.

[Rodden *et al.*, 2020] Aaron Rodden, Tarun Salh, Eriq Augustine, and Lise Getoor. VMI-PSL: Visual Model Inspector for Probabilistic Soft Logic. In *Fourteenth ACM Conference on Recommender Systems*, pages 604–606, 2020.

[Rospocher, 2018] Marco Rospocher. An Ontology-Driven Probabilistic Soft Logic Approach to Improve NLP Entity Annotations. In *International Semantic Web Conference*, pages 144–161. Springer, 2018.

[Scaman and Virmaux, 2018] Kevin Scaman and Aladin Virmaux. Lipschitz regularity of deep neural networks: analysis and efficient estimation. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, pages 3839–3848, 2018.

[Sinha *et al.*, 2017] Ankur Sinha, Pekka Malo, and Kalyanmoy Deb. A Review on Bilevel Optimization: From Classical to Evolutionary Approaches and Applications. *IEEE Transactions on Evolutionary Computation*, 22(2):276–295, 2017.

[Srinivasan *et al.*, 2020] Sriram Srinivasan, Eriq Augustine, and Lise Getoor. Tandem Inference: An Out-of-Core Streaming Algorithm for Very Large-Scale Relational Inference. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(06):10259–10266, 2020.

[Weston *et al.*, 2012] Jason Weston, Frédéric Ratle, Hossein Mobahi, and Ronan Collobert. Deep learning via semi-supervised embedding. *In Neural Networks: Tricks of the Trade*, pages 639–655, 2012.

[Winters *et al.*, 2022] Thomas Winters, Giuseppe Marra, Robin Manhaeve, and Luc De Raedt. Deepstochlog: Neural stochastic logic programming. *Proceedings of the AAAI Conference on Artificial Intelligence*, 36(9):10090–10100, Jun. 2022.

[Yang *et al.*, 2016] Zhilin Yang, William Cohen, and Ruslan Salakhudinov. Revisiting semi-supervised learning with graph embeddings. In *International Conference on Machine Learning*, pages 40–48. PMLR, 2016.

[Yang *et al.*, 2020] Zhun Yang, Adam Ishay, and Joohyung Lee. NeurASP: Embracing Neural networks into Answer Set Programming. In *Proceedings of the 29th International Joint Conference on Artificial Intelligence*, pages 1755–1762. International Joint Conferences on Artificial Intelligence Organization, 7 2020.

[Yih and Meek, 2007] Wen-Tau Yih and Christopher Meek. Improving Similarity Measures for Short Segments of Text. In *Proceedings of the 22nd National Conference on Artificial Intelligence - Volume 2*, AAAI'07, page 1489–1494. AAAI Press, 2007.

[Zhu *et al.*, 2003] Xiaojin Zhu, Zoubin Ghahramani, and John D Lafferty. Semi-supervised learning using gaussian fields and harmonic functions. In *Proceedings of the 20th International Conference on Machine Learning*, pages 912–919. AAAI Press, 2003.