Neural Probabilistic Logic Programming in Discrete-Continuous Domains

Abstract

Neural-symbolic AI (NeSy) allows neural networks to exploit symbolic background knowledge in the form of logic. It has been shown to aid learning in the limited data regime and to facilitate inference on out-of-distribution data. Probabilistic NeSy focuses on integrating neural networks with both logic and probability theory, which additionally allows learning under uncertainty. A major limitation of current probabilistic NeSy systems, such as DeepProbLog, is their restriction to finite probability distributions, i.e., discrete random variables. In contrast, deep probabilistic programming (DPP) excels in modelling and optimising continuous probability distributions. Hence, we introduce DeepSeaProbLog, a neural probabilistic logic programming language that incorporates DPP techniques into NeSy. Doing so results in the support of inference and learning of both discrete and continuous probability distributions under logical constraints. Our main contributions are 1) the semantics of DeepSeaProbLog and its corresponding inference algorithm, 2) a proven asymptotically unbiased learning algorithm, and 3) a series of experiments that illustrate the versatility of our approach.

1 INTRODUCTION

Neural-symbolic AI (NeSy) [Garcez et al., 2002, De Raedt et al., 2021] focuses on the integration of symbolic and neural methods. The advantage of NeSy is that it combines the reasoning power of logical representations with the learning capabilities of neural networks. Additionally, it has been shown to converge faster during learning and to be more robust [Rocktäschel and Riedel, 2017, Xu et al., 2018, Evans and Grefenstette, 2018].

The challenge of NeSy lies in combining discrete symbols with continuous and differentiable neural representations. So far, such a combination has been realised for Boolean variables by interpreting the outputs of neural networks as the weights of these variables. These weights can then be given either a fuzzy semantics [Badreddine et al., 2022, Diligenti et al., 2017] or a probabilistic semantics [Manhaeve et al., 2021a, Yang et al., 2020]. The latter is also used in neural probabilistic logic programming (NPLP), where neural networks parametrise probabilistic logic programs.

A shortcoming of traditional probabilistic NeSy approaches is that they fail to capture models that integrate continuous random variables and neural networks – a feature already achieved with mixture density networks [Bishop, 1994] and more generally deep probabilistic programming (DPP) [Tran et al., 2017, Bingham et al., 2019]. However, it is unclear whether DPP can be generalised to enable logical and relational reasoning. Hence, a gap exists between DPP and NeSy as reasoning is, after all, a fundamental component of the latter. We contribute towards closing this DPP-NeSy gap by introducing DeepSeaProbLog[1], an NPLP language with support for discrete-continuous random variables that retains logical and relational reasoning capabilities. We achieve this integration by allowing arbitrary and differentiable probability distributions expressed in a modern DPP language while combining knowledge compilation [Darwiche and Marquis, 2002] with the reparametrisation trick [Ruiz et al., 2016] and continuous relaxations [Petersen et al., 2021].

Our main contributions are (1) the well-defined probabilistic semantics of DeepSeaProbLog (Section 3) with an inference algorithm based on weighted model integration (WMI) [Belle et al., 2015] (Section 4.1), (2) a proven asymptotically unbiased gradient estimate for WMI that turns DeepSeaProbLog into a differentiable, discrete-continuous NPLP language (Section 4.2), and (3) an experimental evaluation showing the versatility of discrete-continuous reasoning and the efficacy of our approach (Section 6).

[1] 'Sea' stands for the letter C, as in **c**ontinuous random variable.

Submitted to the 39th Conference on Uncertainty in Artificial Intelligence (UAI 2023). **To be used for reviewing only**.

2 LOGIC PROGRAMMING CONCEPTS

A term $\mathtt{t}$ is either a constant $\mathtt{c}$, a variable $\mathtt{V}$ or a structured term of the form $\mathtt{f(t_1,...,t_K)}$, where $\mathtt{f}$ is a functor and each $\mathtt{t}_i$ is a term. Atoms are expressions of the form $\mathtt{q(t_1,...,t_K)}$. Here, $\mathtt{q}/K$ is a predicate of arity K and each $\mathtt{t}_i$ is a term. A literal is an atom or the negation of an atom $\neg\mathtt{q(t_1,...,t_K)}$. A definite clause (also called a rule) is an expression of the form $\mathtt{h:-\ b_1,...,b_K}$ where $\mathtt{h}$ is an atom and each $\mathtt{b}_i$ is a literal. Within the context of a rule, $\mathtt{h}$ is called the head and the conjunction of $\mathtt{b}_i$'s is referred to as the body of the rule. Rules with an empty body are called facts. A logic program is a finite set of definite clauses. If an expression does not contain any variables, it is called ground. Ground expressions are obtained from non-ground ones by means of substitution. A substitution $\theta = \{\mathtt{V}_1 = \mathtt{t}_1, \ldots, \mathtt{V}_K = \mathtt{t}_K\}$ is a mapping from variables $\mathtt{V}_i$ to terms $\mathtt{t}_i$. Applying a substitution θ to an expression $\mathtt{e}$ (denoted $\mathtt{e}\theta$) replaces each occurrence of $\mathtt{V}_i$ in $\mathtt{e}$ with the corresponding $\mathtt{t}_i$.

While *pure* Prolog (or definite clause logic) is defined using the concepts above, practical implementations of Prolog extend definite clause logic with an external arithmetic engine [Sterling and Shapiro, 1994, Section 8]. Such engines enable the use of system specific routines in order to handle numeric data efficiently. Analogous to standard terms in definite clause logic, as defined above, we introduce numeric terms. A numeric term $\mathtt{n}_i$ is either a numeric constant (a real, an integer, a float, etc.), a numeric variable $\mathtt{N}_i$, or a numerical functional term, which is an expression of the form $\varphi\mathtt{(n_1,...,n_K)}$ where φ is an externally defined numerical function. The difference between a standard logical term and a numerical term is that *ground* numerical terms are evaluated and yield a numeric constant. For instance, if `add` is a function, then `add(3, add(5, 0))` evaluates to the numerical constant `8`.

Lastly, numeric constants can be compared to each other using a built-in binary comparison operator $\bowtie \in \{<, =<, >, >=, =:=, =\backslash=\}$. Here we use Prolog syntax to write comparison operators, which correspond to $\{<, \leq, >, \geq, =, \neq\}$ in standard mathematical notation. Comparison operators appear in the body of a rule, have two arguments, and are generally written as $\varphi_l\mathtt{(n_{l,1},...,n_{n,K})} \bowtie \varphi_r\mathtt{(n_{r,1},...,n_{r,K})}$. They evaluate their left and right side and subsequently compare the results, assuming everything is ground. If the stated comparison holds, the comparison is interpreted by the logic program as true, else as false.

3 DEEPSEAPROBLOG

3.1 SYNTAX

While facts in pure Prolog are deterministically true, in probabilistic logic programs they are annotated with the probability with which they are true. These are the so-called probabilistic facts [De Raedt et al., 2007]. When working in discrete-continuous domains, we need to use the more general concept of distributional facts [Zuidberg Dos Martires, 2020], inspired by the distributional clauses of Gutmann et al. [2011].

Definition 3.1 (Distributional fact)**.** *Distributional facts* are expressions of the form $\mathtt{x} \sim \mathtt{distribution(n_1,...,n_K)}$, where $\mathtt{x}$ denotes a term, the $\mathtt{n}_i$'s are numerical terms and `distribution` expresses the probability distribution according to which $\mathtt{x}$ is distributed.

Example 3.1 (Distributional fact)**.** To declare a Poisson distributed variable $\mathtt{x}$ with rate parameter λ, one would write $\mathtt{x} \sim \mathtt{poisson}(\lambda)$.

The meaning of a distributional fact is that all ground instances $\mathtt{x}\theta$ serve as random variables that are distributed according to $\mathtt{distribution(n_1\theta,...,n_K\theta)}$. To obtain a neural-symbolic interface, we will allow neural networks to parametrise these distributions.

Definition 3.2 (Neural distributional fact)**.** A *neural distributional fact* (NDF) is a distributional fact $\mathtt{x} \sim \mathtt{distribution(n_1,...,n_K)}$ in which a subset of the set of numerical terms $\{\mathtt{n}_i\}_{i=1}^K$ is implemented by neural networks that depend on a set of parameters $\mathbf{\Lambda}$.

Random variables defined by NDFs can then be used in the logic in the form of comparisons, e.g., $x > y$, to reason about desired ranges of the variables.

Definition 3.3. (Probabilistic comparison formula) A *probabilistic comparison formula* (PCF) is an expression of the form $(g(\boldsymbol{x}) \bowtie 0)$, where g is a function applied to the set of random variables $\boldsymbol{x}$ and $\bowtie \in \{<, =<, >, >=, =:=, =\backslash=\}$ is a binary comparison operator. A PCF is called *valid* if $\{\boldsymbol{x} \mid g(\boldsymbol{x}) \bowtie 0\}$ is a *measurable* set.

Example 3.2 (Probabilistic comparison formula)**.** If $\mathtt{x}$ is Poisson distributed and represents the number of chocolate pieces put in a chocolate biscuit, then we can use a simple PCF to define when such a biscuit passes a quality test through the rule `passes_test :- (x > 11)`.

Note that the general form of a PCF in Definition 3.3 has a 0 on the right hand side, which can always be obtained by subtracting the right hand-side from both sides of the relation. With the definitions of NDFs and PCFs, a DeepSeaProbLog program can now be formally defined.

Definition 3.4 (DeepSeaProbLog program)**.** A *DeepSeaProbLog program* consists of a finite set of NDFs $\mathcal{F}_D$ (defining random variables), a finite set $\mathcal{C}_M$ of valid PCFs and a set of logical rules $\mathcal{R}_L$ that can use any of those valid PCFs in their bodies.

Example 3.3 (DeepSeaProbLog program)**.** `humid` denotes a Bernoulli random variable that takes the value 1 with a probability p given by the output of a neural network `humid_detector`. `temp` denotes a normally distributed variable whose parameters are predicted by a network `temperature_predictor`. The program further contains two rules that deduce whether we have good weather or not. The first one expresses the case of snowy weather, while the second holds for a rather temperate and dry situation. The atom `query(good_weather(🌍))` declares that we want to compute the probability of `good_weather` when evaluated on the data 🌍. It illustrates the neural-symbolic nature of DeepSeaProbLog, as its ground argument is a sub-symbolic representation (🌍) of the world.

```
humid(Data) ~
    bernoulli(humid_detector(Data)).
temp(Data, T) ~
    normal(temperature_predictor(Data)).

good_weather(Data):-
    humid(Data) =:= 1, temp(Data) < 0.

good_weather(Data):-
    humid(Data) =:= 0, temp(Data) > 15.

query(good_weather(🌍)).
```

Notice how the random variables `humid` and `temp` appear in the body of a logical rule with comparison operators. In our probabilistic setting, the truth value of a comparison depends on the value of its random variables and is thus random itself.

DeepSeaProbLog generalises a range of existing PLP languages. For instance, if we were to remove the distributional fact on `temp` and all the PCFs using them, we would obtain a DeepProbLog program [Manhaeve et al., 2021a]. If we additionally replace the neural network in `humid` with a fixed probability `p`, we end up with a probabilistic logic program [De Raedt et al., 2007]. Replacing that constant probability `p` by a constant `1` yields a non-probabilistic Prolog program. Alternatively, considering all rules and facts in Example 3.3 but replacing the neural parameters of the normal distribution with numeric constants results in a Distributional Clause program [Gutmann et al., 2011]. We further discuss these connections in Appendix A, where we also formally prove that DeepSeaProbLog strictly generalises DeepProbLog.

3.2 SEMANTICS

DeepSeaProbLog programs are used to compute the probability that a ground atom `q` is entailed. That probability follows from the semantics of a DeepSeaProbLog program. As is custom in (probabilistic) logic programming, we will define the semantics of DeepSeaProbLog with respect to ground programs. We will assume that each ground distributional fact $f \in \mathcal{F}_D$ defines a different random variable, as each random variable can only have one unique distribution. Also notice that any ground neural distributional facts will contain the inputs to their neural functions. In a sense, a DeepSeaProbLog program is conditioned on these neural network inputs.

To define the semantics of ground DeepSeaProbLog programs, we first introduce the possible worlds over the PCFs. Every subset C_M of a set of PCFs $\mathcal{C}_M$ defines a possible world $\omega_{C_M} = \{C_M \cup h\theta \mid \mathcal{R}_L \cup C_M \models h\theta \text{ and } h\theta \text{ is ground}\}$. Intuitively speaking, the comparisons in such a subset are considered to be true and all others false. A rule with a comparison in its body that is not in this subset can hence not be used to determine the truth value of atoms. The deterministic rules $\mathcal{R}_L$ and the subset C_M together define a set of all ground atoms $h\theta$ that are derivable, i.e., entailed by the program, and thus considered true. Such a set is called a *possible world*. We refer the reader to the paper of De Raedt and Kimmig [2015] for a detailed account of possible worlds in a PLP context. Following the distribution semantics of Sato [1995] and by taking inspiration from Gutmann et al. [2011], we define the probability of a possible world.

Definition 3.5 (Probability of a possible world)**.** Let $\mathbb{P}$ be a ground DeepSeaProbLog program and $C_M = \{c_1, \ldots, c_H\} \subseteq \mathcal{C}_M$ a set of PCFs that depend on the random variables declared in the set of distributional facts $\mathcal{F}_D$. The probability $P(\omega_{C_M})$ of a world ω_{C_M} is then defined as

$$\int \left[\Big(\prod_{c_i \in C_M} \mathbb{1}(c_i) \Big) \Big(\prod_{c_i \in \mathcal{C}_M \setminus C_M} \mathbb{1}(\bar{c}_i) \Big) \right] \mathrm{d}P_{\mathcal{F}_D}. \quad (1)$$

Here the symbol $\mathbb{1}$ denotes the indicator function, $\bar{c}_i$ is the complement of the comparison c_i and $\mathrm{d}P_{\mathcal{F}_D}$ represents the joint probability measure of the random variables defined in the set of distributional facts $\mathcal{F}_D$.

Example 3.4 (Probability of a possible world)**.** Given $\mathbb{P}$ as in Example 3.3, where `humid_detector(data1)` predicts $p(\texttt{data1})$ and `temperature_predictor(data1)` predicts the tuple $(\mu(\texttt{data1}), \sigma(\texttt{data1}))$, the probability of the possible world $\omega_{\{\texttt{temp(data1)>15},\ \texttt{humid(data1)=:=1}\}}$ is given by

$$p(\texttt{data1}) \cdot \int \mathbb{1}(x{>}15) \frac{\exp\left(-\frac{(x-\mu(\texttt{data1}))^2}{2\sigma^2(\texttt{data1})}\right)}{\sqrt{2\pi}\sigma(\texttt{data1})} \,\mathrm{d}x. \quad (2)$$

Indeed, the measure $\mathrm{d}P_{\mathcal{F}_D}$ decomposes into a counting measure and the product of a Gaussian density function with a differential. The counting measure leads to the factor $p(\texttt{data1})$, since that is the probability that $\texttt{humid(data1)=:=1}$. Hence, the products in Equation 1 reduce to a single indicator of the PCF $(x > 15)$.

Definition 3.6 (Probability of query atom)**.** The probability of a ground atom q is given by

$$P(q) = \sum_{C_M \subseteq \mathcal{C}_M : q \in \omega_{\mathcal{C}_M}} P(\omega_{C_M}). \tag{3}$$

Proposition 3.1 (Measureability of query atom)**.** Let $\mathbb{P}$ be a DeepSeaProbLog program, then $\mathbb{P}$ defines, for an arbitrary query atom q, the probability that q is true.

Proof. See Appendix B. □

4 INFERENCE AND LEARNING

4.1 INFERENCE VIA WEIGHTED LOGIC

A popular technique to perform inference in probabilistic logic programming uses a reduction to so-called *weighted model counting* (WMC); instead of computing the probability of a query, one computes the weight of a propositional logical formula [Chavira and Darwiche, 2008, Fierens et al., 2015]. For DeepSeaProbLog, the equivalent approach is to map a ground program onto a *satisfiability modulo theory* (SMT) formula [Barrett and Tinelli, 2018]. The analogous concept to WMC for these formulas is *weighted model integration* (WMI) [Belle et al., 2015], which can handle infinite sample spaces. In all that follows, for ease of exposition, we assume that all joint probability distributions are continuous.

Proposition 4.1 (Inference as WMI)**.** Assume that the measure $\mathrm{d}P_{\mathcal{F}_D}$ decomposes into a joint probability density function $w(\boldsymbol{x})$ and a differential $\mathrm{d}\boldsymbol{x}$, then the probability $P(q)$ of a query atom q can be expressed as the weighted model integration problem

$$\int \left[\sum_{C_M \subseteq \mathcal{C}_M : q \in \omega_{\mathcal{C}_M}} \prod_{c_i \in C_M \cup \overline{C}_M} \mathbb{1}(c_i(\boldsymbol{x})) \right] w(\boldsymbol{x})\, \mathrm{d}\boldsymbol{x}, \tag{4}$$

where $\overline{C}_M := \{\bar{c}_i \mid c_i \in \mathcal{C}_M \backslash C_M\}$.

Proof. See Appendix C. □

Being able to express the probability of a queried atom in DeepSeaProbLog as a weighted model integral allows us to adapt and deploy inference techniques developed in the weighted model integration literature for DeepSeaProbLog. We opt for the approximate inference algorithm 'Sampo' presented in Zuidberg Dos Martires et al. [2019] because of its more scalable nature. Sampo uses knowledge compilation [Darwiche and Marquis, 2002], a state-of-the-art technique for probabilistic logic inference [Chavira and Darwiche, 2008, Fierens et al., 2015]. Intuitively, knowledge compilation is a two-step procedure applied to a logical formula with PCFs, i.e., an SMT formula. First, it infers the exact probability of all PCFs containing discrete variables through symbolic inference. Then, it converts the remainder of the SMT formula into a polynomial in terms of those exact probabilities and the PCFs containing continuous random variables (Figure 1). This polynomial is the integrand of Equation 4. All that remains is to approximate the integration of this polynomial by sampling from the joint probability distribution $w(\boldsymbol{x})$ of the continuous random variables. In other words, Sampo computes the expression

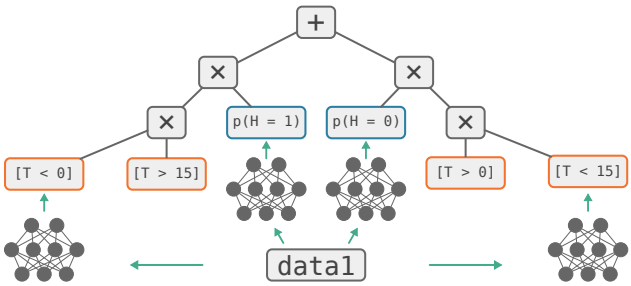

Figure 1: Diagrammatic representation of the result of knowledge compilation for the query in Example 3.3. The blue boxes originate from PCFs over discrete variables, while the orange ones are PCFs over continuous variables. Note how the discrete variable PCFs are reduced to their exact probabilities while the continuous PCFs still need to be inferred.

$$P(q) = \int \mathrm{SP}(\boldsymbol{x}) \cdot w(\boldsymbol{x})\, \mathrm{d}\boldsymbol{x} \approx \frac{1}{|\mathcal{X}|} \sum_{\boldsymbol{x} \in \mathcal{X}} \mathrm{SP}(\boldsymbol{x}), \tag{5}$$

where $\mathcal{X}$ denotes a set of samples drawn from $w(\boldsymbol{x})$ and $\mathrm{SP}(\boldsymbol{x})$ is the result of knowledge compilation, i.e., the sum of products of indicator functions in Equation 4.

We stress that the Sampo algorithm only samples random variables whose expected value with respect to the function $\mathrm{SP}(\boldsymbol{x})$ can not be computed exactly. Hence, in the absence of continuous random variables, our implementation of DeepSeaProbLog using Sampo coincides with DeepProbLog on both a semantics level (Proposition A.1) and inference level.

4.2 LEARNING VIA DIFFERENTIATION

A DeepSeaProbLog program depend on a set of (neural) parameters $\boldsymbol{\Lambda}$ (Definition 3.2). In order to optimise these parameters, we need to take their gradients of a loss function that compares the probability $P(q)$ to a training signal. More precisely, we need to compute the derivative

$$\partial_\lambda \mathcal{L}(P_{\boldsymbol{\Lambda}}(q)) = \partial_{P_{\boldsymbol{\Lambda}}(q)} \mathcal{L}(P_{\boldsymbol{\Lambda}}(q)) \cdot \partial_\lambda P_{\boldsymbol{\Lambda}}(q), \tag{6}$$

where we explicitly indicate the dependency of the probability on $\mathbf{\Lambda}$ and $\lambda \in \mathbf{\Lambda}$. Differentiating $P_{\mathbf{\Lambda}}(q)$ with respect to λ presents two obstacles. First, the question of differentiating through the sampling process of Equation 5 and second, the non-differentiability of the indicator functions in $\text{SP}(\boldsymbol{x})$.

The non-differentiability of sampling is tackled using the reparametrisation trick [Ruiz et al., 2016]. Reparametrisation offers better estimates than other approaches, such as REINFORCE [Williams, 1992] and is readily utilised in modern probabilistic programming languages such as Tensorflow Probability [Tran et al., 2017] and Pyro [Bingham et al., 2019]. Conversely, the non-differentiability of the indicator functions prevents swapping the order of differentiation and integration [Flanders, 1973], which we resolve by applying continuous relaxations following the work of Petersen et al. [2021]. Together, we obtain the gradient estimate

$$\partial_\lambda P_{\mathbf{\Lambda}}(q) = \partial_\lambda \int \text{SP}(\boldsymbol{x}) \cdot w_{\mathbf{\Lambda}}(\boldsymbol{x}) \, \mathrm{d}\boldsymbol{x} \tag{7}$$

$$\approx \int [\partial_\lambda \text{SP}_s(r(\boldsymbol{u}, \mathbf{\Lambda}))] \cdot p(\boldsymbol{u}) \, \mathrm{d}\boldsymbol{u}, \tag{8}$$

where the subscript s in $\text{SP}_s(\boldsymbol{x})$ denotes the continuously relaxed or 'softened' version of $\text{SP}(\boldsymbol{x})$ and $r(\boldsymbol{u}, \mathbf{\Lambda})$ is the reparametrisation function.

Our gradient estimate using relaxations is asymptotically unbiased. As an example of these relaxations, consider the indicator of a PCF $(g(\boldsymbol{x}) > 0)$, which is relaxed into the sigmoid $\sigma(\beta \cdot g(\boldsymbol{x}))$. Appendix D provides more details on relaxations of general PCFs. The *coolness* parameter $\beta \in \mathbb{R}_+^0$ determines the strictness of the relaxation. Hence, we recover the hard indicator function when $\beta \to +\infty$. Note that relaxing indicator functions introduces bias. Petersen et al. [2021] already stated in their work that, in the infinite coolness limit, a relaxed function coincides with the non-relaxed one. Proposition 4.2 extends this result to the derivatives of relaxed and non-relaxed functions, proving that our gradient estimate is asymptotically unbiased.

Proposition 4.2 (Unbiased in the infinite coolness limit)**.** Let $\mathbb{P}$ be a DeepSeaProbLog program with PCFs $(g_i(\boldsymbol{x}) \bowtie 0)$ and corresponding coolness parameters β_i.
If all $\partial_\lambda(g_i \circ r)$ are locally integrable over $\mathbb{R}^k$ and every $\beta_i \to +\infty$, then we have, for any query atom q, that

$$\partial_\lambda P(q) = \int \partial_\lambda \text{SP}_s(r(\boldsymbol{u}, \mathbf{\Lambda})) \cdot p(\boldsymbol{u}) \, \mathrm{d}\boldsymbol{u}. \tag{9}$$

Proof. The proof makes use of the mathematical theory of distributions [Schwartz, 1957], which generalise the concept of functions, and is given in Appendix E. □

Finally, we obtain a practical and unbiased estimate of $\partial_\lambda P_{\mathbf{\Lambda}}(q)$ using a set of samples $\mathcal{U}$ drawn from $p(\boldsymbol{u})$.

$$\partial_\lambda P(q) \approx \int [\partial_\lambda \text{SP}_s(r(\boldsymbol{u}, \mathbf{\Lambda}))] \cdot p(\boldsymbol{u}) \, \mathrm{d}\boldsymbol{u} \tag{10}$$

$$\approx \frac{1}{|\mathcal{U}|} \sum_{\boldsymbol{u} \in \mathcal{U}} \partial_\lambda \text{SP}_s(r(\boldsymbol{u}, \mathbf{\Lambda})). \tag{11}$$

Computing this gradient estimate does not require drawing new samples. Implementing the relaxations of PCFs in a 'straight-through' manner allows us to directly apply automatic differentiation on the inferred probability.

4.3 PROBABILISTIC PROGRAMMING CONNECTIONS

Since knowledge compilation symbolically infers discrete random variables, we only have to sample from a continuous joint probability distribution. To sample such distributions, we can fully exploit the advanced inference and learning techniques [Hoffman et al., 2014] of modern probabilistic programming languages [Tran et al., 2017, Bingham et al., 2019]. Our implementation of DeepSeaProbLog utilises Tensorflow Probability for this task, effectively using knowledge compilation as a differentiable bridge between logical and probabilistic reasoning. While this bridge is limited to sampling techniques for now, it presents an interesting direction for future work to completely unify NeSy with DPP.

4.4 LIMITATIONS

While the use of relaxations is well-known and used in recent gradient estimators [Tucker et al., 2017, Grathwohl et al., 2018], the bias they introduce is often hard to deal with in practice. In our case, this bias only reduces to zero in the infinite coolness limit (Proposition 4.2), meaning the use of annealing can be necessary. Finding a good annealing scheme for any problem is non-trivial and effectively introduces another component in need of optimisation. However, as relaxations allow the use of the reparametrisation trick, the resulting lower variance estimates together with our theoretical guarantees support our choice. A more detailed discussion of the current limitations of DeepSeaProbLog can be found in Appendix H.

5 RELATED WORK

From a NeSy perspective the formalism most closely related to DeepSeaProbLog is that of *Logic Tensor Networks* (LTNs) [Badreddine et al., 2022]. The main difference between LTNs and DeepSeaProbLog is the fuzzy logic semantics of the former and the probabilistic semantics of the latter. Interestingly, LTNs and other NeSy approaches based on fuzzy logic also require relaxations to incorporate continuous values. However, fuzzy-based approaches require

these relaxations at the semantics level, in contrast to DeepSeaProbLog. Even more, they can only compare continuous point values instead of more general continuous random variables. LTNs' fuzzy semantics also exhibit drawbacks on a more practical level. Unlike DeepSeaProbLog with its probabilistic semantics, LTNs are not inherently capable of neural-symbolic generative modelling (Section 6.3). For a broader overview of the field of neural-symbolic AI, we refer the reader to a series of survey papers that have been published in recent years [Garcez et al., 2019, Marra et al., 2021, Garcez et al., 2022].

From a probabilistic programming perspective, DeepSeaProbLog is related to languages that handle discrete and continuous random variables such as *BLOG* [Milch, 2006], *Distributional Clauses* [Gutmann et al., 2011] and *Anglican* [Tolpin et al., 2016], which have all been given declarative semantics, i.e., the meaning of the program does not depend on the underlying inference algorithm. However, these languages have the drawback of non-differentiability. This drawback stands in stark contrast to end-to-end (deep) probabilistic programming languages such as Pyro [Bingham et al., 2019] or Tensorflow Probability [Dillon et al., 2017], but these have only been equipped with operational semantics and do not support logical constraints. DeepSeaProbLog not only introduces the ability to express such logical constraints in the form of PCFs to construct challenging posterior distributions, but does so in an end-to-end differentiable fashion.

Finally, our gradient estimate can be related to relaxation-based methods like REBAR [Tucker et al., 2017] or RELAX [Grathwohl et al., 2018], but without the REINFORCE [Williams, 1992] inspired component. Instead, we utilise the differentiability of knowledge compilation to obtain exact gradients of discrete variables. Since our inference scheme innately requires knowledge compilation, the use of other discrete gradient estimators like [Niepert et al., 2021] does not directly apply to DeepSeaProbLog. Moreover, we exploit the structure of our problem by directly relaxing comparison formulae in a sound manner [Petersen et al., 2021], in contrast to introducing an artificial relaxation of the whole problem [Grathwohl et al., 2018].

6 EXPERIMENTAL EVALUATION

We illustrate the versatility of DeepSeaProbLog by tackling three different problems. Section 6.1 discusses the detection of handwritten dates without location supervision. In Section 6.2 a hybrid Bayesian network with conditional probabilities dependent on the satisfaction of certain logical constraints will be optimised. Finally, Section 6.3 introduces neural-symbolic variational auto-encoders, inspired by Misino et al. [2022].

The details of our experimental setup, including the precise DeepSeaProbLog programs, coolness annealing schemes and hyperparameters used for the neural networks are given in Appendix F.

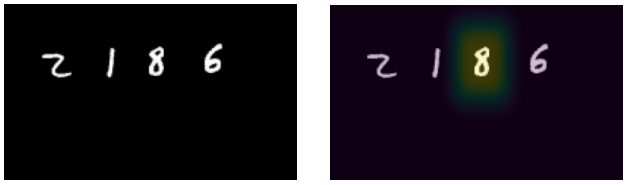

Figure 2: On the left, an example of a handwritten year. On the right, the attention map for the digit '8' as a generalised normal distribution. Intuitively, we can view generalised normal distributions as differentiable bounding boxes. This allows gradients to flow from a downstream classification network to the regression component.

6.1 NEURAL-SYMBOLIC ATTENTION

A problem that cannot yet be solved to a satisfactory degree by purely neural or other neural-symbolic systems is detecting handwritten years. Given a single image with a handwritten year, the task is to predict the correct year as a sequence of 4 digits together with the location of these digits (Figure 2, left). This year can be anywhere in the image and the only supervision is in the digits of the year, *not* where these digits are in the image. In other words, the problem is equivalent to object detection *without* bounding box supervision.

Solving such a problem seems to be out of scope for current methods. On the one hand, existing neural approaches are often complex pipelines of neural components that break end-to-end differentiability [Seker and Ahn, 2022]. On the other hand, current neural-symbolic methods lack sufficient spatial reasoning capabilities in order to perform the necessary image segmentation.

We exploit probabilistic programming by modelling the location of a digit as a deep generalised normal distribution [Nadarajah, 2005]. That is, we use a convolutional neural network to regress the parameters of four generalised normal distributions, one for each digit of a year. Then, we take inspiration from the spatial transformer literature [Carion et al., 2020] and convert the distribution of each location to an attention map (Figure 2, right).

In our experimental validation we compare DeepSeaProbLog to a neural baseline and logic tensor networks. The neural baseline applies the four probabilistic attention maps, one for the location of each of the four digits, to the input image. The resulting four attenuated images are then passed on to a classification network without additional reasoning. The network is simply required to predict the digits in the right order. With DeepSeaProbLog, we encode that a year

Table 1: Mean accuracy and IoU with standard error for classifying the correct year, taken over 10 runs.

Method	Results	
	acc.	IoU
DeepSeaProbLog	93.77 ± 0.57	17.69 ± 0.23
LTN	76.50 ± 12.10	10.73 ± 1.69
Neural Baseline	54.71 ± 14.33	6.26 ± 1.77

is a sequence of digits, i.e., the order matters, by enforcing an explicit order on the digit locations. Doing so requires spatial reasoning, i.e., reasoning which digit is at which location. For LTNs, we encode the same information. However, as LTNs lack a proper distribution semantics, they can only reason on the level of the expected values of the generalised normal distributions.

In our experiment, the sets of years appearing in the training, validation and test data are all disjoint. Moreover, the sets of handwritten digits used to generate those years are also disjoint. Partitioning the data in such a way leads to a challenging learning problem; the difficulty lies in out-of-distribution inference, as the years and handwritten digits in the validation and test set have never been seen during training.

We evaluate all methods in terms of accuracy and Intersection-over-Union (IoU). For the accuracy, we compare the sequence of predicted digits to the correct sequence of digits constituting a year. A prediction is correct if *all* digits are correctly predicted in the right order. For the IoU, we map each predicted generalised normal distribution to a bounding box by using the mean as the centre and the scale parameter as the width of the box. The IoU is then given by the overlap between this box and the true location of the handwritten digit.

We present our results in Table 1. The most striking observation is the poor performance and large variance of the neural baseline. It fails to predict the location of the digits in the right order, as can be seen from the lower IoU values. Since classification depends on the predicted locations, these lower values also explain the lack in accuracy. We can conclude that the neural baseline struggles to generalise to out-of-distribution data. While LTNs fare better, the high standard error on the accuracy indicates that their continuous reasoning capabilities are insufficient to guarantee consistent solutions. DeepSeaProbLog distinguishes itself by a higher and more consistent accuracy. The reason is also clear; DeepSeaProbLog exploits the entire domain of the distribution of each location. This then leads to a higher IoU value that in turn results in a higher accuracy.

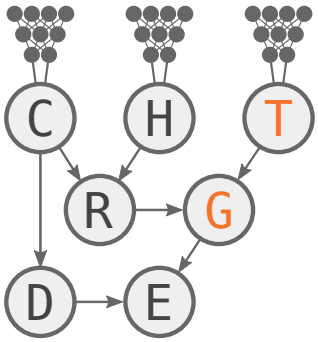

Figure 3: Graphical model of **E**njoying the weather (**E**).(**E**) holds when **D**epressed (**D**) is not true and there is **G**ood weather (**G**). A person has a higher probability of being depressed when it is **C**loudy (**C**), while the degree of good weather is beta distributed depending on various logical constraints on **T**emperature (**T**) and **R**ain (**R**). Finally, rain is probable when it is both **C**loudy and **H**umid (**H**).

6.2 NEURAL HYBRID BAYESIAN NETWORKS

Hybrid Bayesian networks [Lerner, 2003] are probabilistic graphical models that combine discrete and continuous random variables. DeepSeaProbLog allows for the introduction of optimisable neural components and logical constraints to such models, as shown in Example 3.3. We further extend this example (Figure 3) and specify the datasets that form the input to the various neural networks. The temperature is predicted from a real meteorological dataset [Cho et al., 2020] and we use CIFAR-10 images as proxies for observing clouds and humidity. Moreover, dependencies on a number of constraints are added, which goes beyond the capabilities of traditional probabilistic programming.

Our neural Bayesian model was optimised by only giving probabilistic supervision on whether **E** was true or false, i.e., the weather was enjoyed or not. Given our model, such distant supervision only translates into a learning signal on different *ranges* of temperature values that satisfy different PCFs. We will see that DeepSeaProbLog's reasoning over the full domain of the temperature distribution allows it to perform meaningful density estimation from such a signal.

The optimised Bayesian model can be evaluated in two ways. First, the accuracy on CIFAR-10 of the networks utilised in **C**loudy and **H**umid, which were 95.24 ± 3.32 and 98.96 ± 0.11, respectively. Second, we measure the quality of the density estimation on **T**emperature by looking at the MSE between the true and predicted mean values, which was 0.1799 ± 0.0139 . Importantly, DeepSeaProbLog was able to approximate the standard deviation of **T**emperature from just the distant supervision, deviating by only 0.60 ± 0.22.

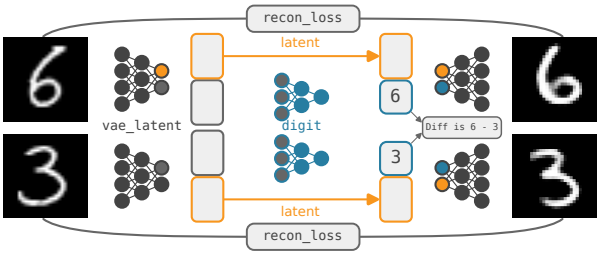

Figure 4: Given example pairs of images and the value of their subtraction, e.g., (6, 3) and 3, the CVAE encoder `vae_latent` first encodes each image into a multivariate normal NDF (`latent`) and a latent vector. The latter is the input of a categorical NDF `digit`, completing the CVAE latent space. Supervision is dual; generated images are compared to the original ones in a probabilistic reconstruction loss, while both digits need to subtract to the given value.

6.3 NEURAL-SYMBOLIC VARIATIONAL AUTO-ENCODER

Probabilistic programming is well-suited to generative tasks, but it can not perform generation conditioned on logical constraints. Inspired by the work of Misino et al. [2022], we showcase how DeepSeaProbLog extends the generative power of probabilistic programming to such constraints. To this end, we will consider the task of learning to generate 2 images of digits given the value of their subtraction.

A diagrammatic overview of our DeepSeaProbLog program is given in Figure 4. It uses a conditional variational auto-encoder (CVAE) [Sohn et al., 2015] to generate images conditioned on a digit value. DeepSeaProbLog finds those digit values from a given subtraction result by logical reasoning. It can also condition generation on other variables in the CVAE latent space as this space is an integral part of DeepSeaProbLog's deep, relational model. We will exploit this property later on when we extend the task to generating digits in the same writing style as a given image without *any* additional optimisation.

Both the CVAE and digit classifier are successfully trained jointly. Example generations of images that satisfy the subtraction result $\boxed{?} - \boxed{?} = 5$ can be seen below. In general, DeepSeaProbLog finds all possible digits that subtract to a given value and generates images for each correct combination. Below, we left out 2 such combinations for clarity of exposition.

While our program is inspired by the VAEL architecture of Misino et al. [2022], conceptual differences exist. Most notably, for VAEL, the image generation resides outside the probabilistic logic program. Conversely, the CVAE, including its latent space, is explicitly declared and accessible in DeepSeaProbLog. This difference allows DeepSeaProbLog to generalise to conditional generative queries that differ significantly from the original optimisation task. For example, we can *zero-shot* query the program to fill in the blank in $\boxed{1} - \boxed{?} = \texttt{Diff}$ instead of the two blanks of the learning task $\boxed{?} - \boxed{?} = \texttt{Diff}$. Even more, we can enforce that the generated digit is in the same writing style as the given digit by conditioning the generation on the latent space of the given image (Figure 5).

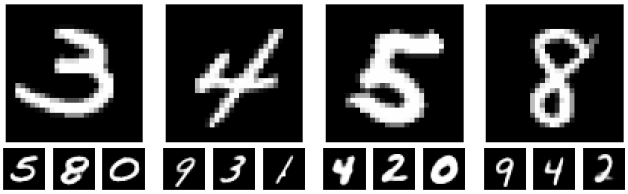

Figure 5: Five random images of right digits (top row) and their generated left digits for 3 given random difference values (bottom row). Note the preservation of the style of the given minuends.

7 CONCLUSION

We presented DeepSeaProbLog, a novel neural-symbolic probabilistic logic programming language that integrates hybrid probabilistic logic and neural networks. Inference is dealt with efficiently through approximate weighted model integration while learning is facilitated by reparametrisation and continuous relaxations of non-differentiable logic components. Our experiments illustrate how DeepSeaProbLog is capable of intricate probabilistic modelling allowing for meaningful weak supervision while maintaining strong out-of-distribution performance. Moreover, they show how hybrid probabilistic logic can be used as a flexible structuring formalism for the neural paradigm that can effectively optimise and reuse neural components in different tasks.

References

Martín Abadi. Tensorflow: learning functions at scale. In *International Conference on Functional Programming*, 2016.

Samy Badreddine, Artur d'Avila Garcez, Luciano Serafini, and Michael Spranger. Logic tensor networks. *Artificial Intelligence*, 2022.

Clark Barrett and Cesare Tinelli. Satisfiability modulo theories. In *Handbook of model checking*. Springer, 2018.

Vaishak Belle, Andrea Passerini, and Guy Van den Broeck. Probabilistic inference in hybrid domains by weighted model integration. In *IJCAI*, 2015.

Eli Bingham, Jonathan P Chen, Martin Jankowiak, Fritz Obermeyer, Neeraj Pradhan, Theofanis Karaletsos, Rohit Singh, Paul Szerlip, Paul Horsfall, and Noah D Goodman. Pyro: Deep universal probabilistic programming. *The Journal of Machine Learning Research*, 2019.

Christopher M Bishop. Mixture density networks. Technical report, Aston University, 1994.

Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. End-to-end object detection with transformers. In *Computer Vision–ECCV*. Springer, 2020.

Mark Chavira and Adnan Darwiche. On probabilistic inference by weighted model counting. *Artificial Intelligence*, 2008.

Dongjin Cho, Cheolhee Yoo, Jungho Im, and Dong-Hyun Cha. Comparative assessment of various machine learning-based bias correction methods for numerical weather prediction model forecasts of extreme air temperatures in urban areas. *Earth and Space Science*, 2020.

Adnan Darwiche and Pierre Marquis. A knowledge compilation map. *Journal of Artificial Intelligence Research*, 2002.

Luc De Raedt and Angelika Kimmig. Probabilistic (logic) programming concepts. *Machine Learning*, 2015.

Luc De Raedt, Angelika Kimmig, and Hannu Toivonen. Problog: A probabilistic prolog and its application in link discovery. In *IJCAI*. Hyderabad, 2007.

Luc De Raedt, Sebastijan Dumančić, Robin Manhaeve, and Giuseppe Marra. From statistical relational to neural-symbolic artificial intelligence. In *IJCAI*, 2021.

Michelangelo Diligenti, Marco Gori, and Claudio Saccà. Semantic-based regularization for learning and inference. *Artif. Intell.*, 2017.

Joshua V Dillon, Ian Langmore, Dustin Tran, Eugene Brevdo, Srinivas Vasudevan, Dave Moore, Brian Patton, Alex Alemi, Matt Hoffman, and Rif A Saurous. Tensorflow distributions. *arXiv*, 2017.

Richard Evans and Edward Grefenstette. Learning explanatory rules from noisy data. *Journal of Artificial Intelligence Research*, 2018.

Daan Fierens, Guy Van den Broeck, Joris Renkens, Dimitar Shterionov, Bernd Gutmann, Ingo Thon, Gerda Janssens, and Luc De Raedt. Inference and learning in probabilistic logic programs using weighted boolean formulas. *Theory and Practice of Logic Programming*, 2015.

Harley Flanders. Differentiation under the integral sign. *The American Mathematical Monthly*, 1973.

Artur d'Avila Garcez, Marco Gori, Luis C Lamb, Luciano Serafini, Michael Spranger, and Son N Tran. Neural-symbolic computing: An effective methodology for principled integration of machine learning and reasoning. *arXiv*, 2019.

Artur d'Avila Garcez, Sebastian Bader, Howard Bowman, Luis C Lamb, Leo de Penning, BV Illuminoo, Hoifung Poon, and Gerson Zaverucha. Neural-symbolic learning and reasoning: A survey and interpretation. *Neuro-Symbolic Artificial Intelligence: The State of the Art*, 2022.

Artur S d'Avila Garcez, Krysia Broda, Dov M Gabbay, et al. *Neural-symbolic learning systems: foundations and applications*. Springer Science & Business Media, 2002.

Will Grathwohl, Dami Choi, Yuhuai Wu, Geoff Roeder, and David Duvenaud. Backpropagation through the void: Optimizing control variates for black-box gradient estimation. In *ICLR*, 2018.

Andreas Griewank and Andrea Walther. *Evaluating derivatives: principles and techniques of algorithmic differentiation*. SIAM, 2008.

Bernd Gutmann, Ingo Thon, Angelika Kimmig, Maurice Bruynooghe, and Luc De Raedt. The magic of logical inference in probabilistic programming. *Theory and Practice of Logic Programming*, 2011.

Matthew D Hoffman, Andrew Gelman, et al. The no-u-turn sampler: adaptively setting path lengths in hamiltonian monte carlo. *J. Mach. Learn. Res.*, 2014.

Jiani Huang, Ziyang Li, Binghong Chen, Karan Samel, Mayur Naik, Le Song, and Xujie Si. Scallop: From probabilistic deductive databases to scalable differentiable reasoning. *NeurIPS*, 2021.

Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparametrization with gumble-softmax. In *ICLR*, 2017.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015.

Uri Nahum Lerner. *Hybrid Bayesian networks for reasoning about complex systems*. stanford university, 2003.

Chris J Maddison, Andriy Mnih, and Yee Whye Teh. The concrete distribution: A continuous relaxation of discrete random variables. In *ICLR*, 2017.

Robin Manhaeve, Sebastijan Dumančić, Angelika Kimmig, Thomas Demeester, and Luc De Raedt. Neural probabilistic logic programming in deepproblog. *Artificial Intelligence*, 2021a.

Robin Manhaeve, Giuseppe Marra, and Luc De Raedt. Approximate inference for neural probabilistic logic programming. In *KR*, 2021b.

Giuseppe Marra, Sebastijan Dumančić, Robin Manhaeve, and Luc De Raedt. From statistical relational to neural symbolic artificial intelligence: a survey. *arXiv*, 2021.

Brian Christopher Milch. *Probabilistic models with unknown objects*. PhD thesis, University of California, Berkeley, 2006.

Eleonora Misino, Giuseppe Marra, and Emanuele Sansone. Vael: Bridging variational autoencoders and probabilistic logic programming. In *NeurIPS*, 2022.

Paolo Morettin, Pedro Zuidberg Dos Martires, Samuel Kolb, and Andrea Passerini. Hybrid probabilistic inference with logical and algebraic constraints: a survey. In *IJCAI*, 2021.

Saralees Nadarajah. A generalized normal distribution. *Journal of Applied statistics*, 2005.

Mathias Niepert, Pasquale Minervini, and Luca Franceschi. Implicit mle: backpropagating through discrete exponential family distributions. *NeurIPS*, 2021.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *NeurIPS*, 2019.

Felix Petersen, Christian Borgelt, Hilde Kuehne, and Oliver Deussen. Learning with algorithmic supervision via continuous relaxations. *NeurIPS*, 2021.

Fabrizio Riguzzi and Terrance Swift. Well–definedness and efficient inference for probabilistic logic programming under the distribution semantics. *Theory and practice of logic programming*, 2013.

Tim Rocktäschel and Sebastian Riedel. End-to-end differentiable proving. *NeurIPS*, 2017.

Francisco J. R. Ruiz, Michalis K. Titsias, and David M. Blei. The generalized reparameterization gradient. *NeurIPS*, 2016.

Taisuke Sato. A statistical learning method for logic programs with distribution semantics. In *ICLP*, 1995.

Laurent Schwartz. Théorie des distributions à valeurs vectorielles. i. In *Annales de l'institut Fourier*, 1957.

Ahmet Cagatay Seker and Sang Chul Ahn. A generalized framework for recognition of expiration dates on product packages using fully convolutional networks. *Expert Systems with Applications*, 2022.

Kihyuk Sohn, Honglak Lee, and Xinchen Yan. Learning structured output representation using deep conditional generative models. *NeurIPS*, 2015.

Leon Sterling and Ehud Y Shapiro. *The art of Prolog: advanced programming techniques*. MIT press, 1994.

David Tolpin, Jan-Willem van de Meent, Hongseok Yang, and Frank Wood. Design and implementation of probabilistic programming language anglican. In *Symposium on the Implementation and Application of Functional programming Languages*, 2016.

Dustin Tran, Matthew D. Hoffman, Rif A. Saurous, Eugene Brevdo, Kevin Murphy, and David M. Blei. Deep probabilistic programming. In *ICLR*, 2017.

George Tucker, Andriy Mnih, Chris J Maddison, John Lawson, and Jascha Sohl-Dickstein. Rebar: Low-variance, unbiased gradient estimates for discrete latent variable models. *NeurIPS*, 2017.

Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 1992.

Thomas Winters, Giuseppe Marra, Robin Manhaeve, and Luc De Raedt. Deepstochlog: Neural stochastic logic programming. In *AAAI*, 2022.

Jingyi Xu, Zilu Zhang, Tal Friedman, Yitao Liang, and Guy Van den Broeck. A semantic loss function for deep learning with symbolic knowledge. In *ICML*, 2018.

Zhun Yang, Adam Ishay, and Joohyung Lee. Neurasp: Embracing neural networks into answer set programming. In *IJCAI*, 2020.

Pedro Zuidberg Dos Martires. *From Atoms to Possible Worlds: Probabilistic Inference in the Discrete-Continuous Domain*. PhD thesis, KU Leuven, 2020.

Pedro Zuidberg Dos Martires, Anton Dries, and Luc De Raedt. Exact and approximate weighted model integration with probability density functions using knowledge compilation. In *AAAI*, 2019.