Neuro-Symbolic Reasoning for Handwritten Expression Evaluation

Kangrui Ren
School of Computer Science
McGill University
Montreal, Quebec, Canada
kangrui.ren@mail.mcgill.ca

ABSTRACT

In this report, we explore the capabilities of DeepProbLog, a language designed to integrate logical reasoning with neural networks, in handling complex inference tasks. Our focus was on reverse logical reasoning, where we aimed to evaluate the system's ability to infer unknown variables within arithmetic expressions. Through a series of experiments, we discovered that while DeepProbLog performs well on basic arithmetic tasks, its ability to handle more complex non-basic arithmetic reasoning is hindered by limitations in the gradient semiring approach. Additionally, we observed that the introduction of interference elements, such as unknown shape images, significantly impacted the system's reasoning accuracy. These findings highlight both the strengths and limitations of the current model, providing insights into potential areas for improvement in future work, particularly regarding more advanced symbolic reasoning and probabilistic inference.

1 INTRODUCTION

1.1 Integration of perception and reasoning

In recent years, Deep Learning has rapidly evolved into a prominent research area and a mainstream direction within the broader field of Artificial Intelligence (AI). The model's success lies in its architecture, which comprises multiple layers that progressively extract higher-level abstractions from raw data. Deep Learning has led to transformative progress in fields such as machine learning and computer vision, allowing machines to excel at tasks that require recognizing patterns in large, complex datasets. These advancements have established deep learning as a key player in AI research and its various applications.

Despite the success of Deep Learning in low-level perception tasks, the integration of low-level perception with high-level reasoning remains an ongoing challenge. This is especially important as many real-world tasks require not only recognizing patterns but also understanding and reasoning about them. Several researchers [2, 11, 15] have pursued the modernization of early neural-symbolic integration techniques in hopes of effectively combining statistical learning with probabilistic logic programming. Their goal is to bridge the gap between perception (handled well by neural networks) and reasoning, which typically relies on symbolic logic.

While Deep Learning and Neural Networks have shown remarkable promise in solving low-level perception problems, there is still a clear limitation when it comes to handling high-level reasoning tasks. Although modern advances in logical

and probabilistic representations have made significant progress in addressing these high-level reasoning problems, the full integration of deep neural networks with high-level probabilistic reasoning remains an open problem. This challenge persists due to the inherent complexity of combining flexible, probabilistic reasoning with the representational power of deep networks.

To address this issue, our research utilizes a novel language called DeepProbLog [13], which extends the capabilities of the probabilistic logic programming language ProbLog [5]. DeepProbLog integrates neural networks into logical reasoning by allowing neural predicates to be treated as part of logical expressions. Unlike traditional approaches, which often rely on embedding reasoning capabilities within complex neural network architectures, DeepProbLog simplifies this integration by leveraging the strengths of both systems. Specifically, DeepProbLog encapsulates neural network outputs as probabilities in logical predicates. For instance, an atomic expression of the form $q(t_1, \ldots, t_n)$ can have a probability associated with it, and if the neural network's output can be interpreted as a probability, it can be integrated into the logical reasoning framework as a neural predicate.

As demonstrated by [13], DeepProbLog has proven to be particularly effective for additive reasoning tasks. For example, after training the model with appropriate logical rules, it can accurately infer the sum of two handwritten digits (Figure 8). This additive reasoning capability highlights the potential of DeepProbLog to address symbolic reasoning tasks that have traditionally been handled by rule-based systems.

```
nn(mnist_net,[X],Y,[0,1,2,3,4,5,6,7,8,9]) :: digit(X,Y).

addition(X,Y,Z) :- digit(X,X2), digit(Y,Y2), Z is X2+Y2.
```

Figure 1: Additive Reasoning Logical Rule

The example in Figure 8 illustrates a basic case where $digit(X, Y)$ passes an input image X to a neural network, which then outputs the recognized number Y. The second line shows the training predicates, where X and Y are the handwritten digit images, and Z is the sum of X and Y. This framework can be further extended to handle multi-digit numbers without requiring additional training, making it a flexible tool for symbolic reasoning tasks involving numerical data.

Building on this additive reasoning framework, we explored the system's ability to handle reverse reasoning tasks. Specifically, we aimed to determine whether DeepProbLog could solve equations, even if they were linear equations with unknown variables. Success in this domain would indicate that DeepProbLog could be extended to solve more complex equations through incremental expansions of the logical rule set. This approach aligns with the system's underlying philosophy, where logic-based rules are used as a foundation for handling more intricate mathematical or symbolic reasoning tasks.

By "solving equations," we refer to scenarios where predicates involve handwritten digits that are recognized by the neural network, along with unknowns represented by images that cannot be recognized directly by the network. For instance, consider the predicate $addition([|X|], [|3|], 9)$, which represents the equation $X + 3 = 9$. In this case, the model should infer that $X = 6$. Our experiments, detailed in Section 4, evaluate the system's ability to solve such equations. Notably, even when interference images (unrecognizable by the network) are present, DeepProbLog can still infer the correct sum, although the forward inference accuracy may decrease slightly. Further results are presented in Section 4.4.

1.2 LP Concepts

In this section, we will review some fundamental concepts in logical programming, which are crucial

for understanding the subsequent sections. Logical programming provides the backbone for symbolic reasoning and is essential for integrating neural networks with logical inference systems.

A term t can take one of three forms: - A **mutable variable** V, representing a placeholder that can assume different values throughout computation. - A **constant** c, which refers to a fixed, unchangeable value. - An n-ary **functor** $f(t_1, t_2, \ldots, t_n)$, where each t_i is itself a term. Functors allow us to construct more complex terms from simpler ones.

An **atom** is an expression of an n-ary predicate $p(t_1, t_2, \ldots, t_n)$, where each t_i is a term. Mathematically, we can write this as:

$$p(t_1, t_2, \ldots, t_n),$$

where p is the predicate symbol, and the terms $t_1, t_2, \ldots, t_n$ serve as its arguments. This predicate can be viewed as a fundamental logical statement or fact, representing relationships or properties of objects in the domain.

A **substitution** is the process of assigning specific terms to variables. Formally, it is represented as:

$$S = \{V_1 = t_1, V_2 = t_2, \ldots, V_n = t_n\},$$

where each variable V_i is substituted with a corresponding term t_i. This operation allows logical systems to instantiate variables with specific values during inference, enabling forward or backward reasoning.

A **literal** is either an atom or its negation. For example, given an atom $p(t_1, t_2, \ldots, t_n)$, its negation is denoted as $\neg p(t_1, t_2, \ldots, t_n)$, which expresses that the predicate does not hold for the given terms.

In logical programming, a **rule** is expressed in the form:

$$h \ :- \ b_1, b_2, \ldots, b_n,$$

where h (the head) is an atom, and $b_1, b_2, \ldots, b_n$ (the body) are literals. This rule implies that h holds true if all the literals $b_1, b_2, \ldots, b_n$ are true. In logical terms, this is represented as:

$$h \Leftrightarrow (b_1 \wedge b_2 \wedge \cdots \wedge b_n).$$

This form of logical equivalence is central to rule-based reasoning, as it allows systems to derive conclusions based on the satisfaction of multiple conditions.

The concept of **grounding** refers to an expression that contains no variables, meaning that all terms are constants. A ground expression is a fully instantiated fact, such as:

$$p(c_1, c_2, \ldots, c_n),$$

where $c_1, c_2, \ldots, c_n$ are constants. Grounding is crucial in logical inference, as it provides concrete facts from which more complex reasoning can be built.

These foundational concepts are essential for understanding how logical reasoning frameworks, such as DeepProbLog, operate in conjunction with neural network outputs. In Section 3, we will delve deeper into how these concepts are applied to enable neural-symbolic reasoning.

1.3 DeepProbLog Program

Now, we will briefly introduce the structure of a DeepProbLog program by first explaining the foundation it builds upon, namely, a ProbLog program. Understanding the components and mechanics of ProbLog is essential to fully grasping how DeepProbLog extends this framework to integrate neural networks into logical inference systems.

A **ProbLog program** consists of two primary components: a set of rules $\mathcal{R}$ and a set of ground probabilistic facts $\mathcal{F}$. Each fact in $\mathcal{F}$ takes the form $p :: f$, where p represents the probability associated with a ground atom f. The rules in $\mathcal{R}$ serve to define logical relationships between these probabilistic facts, facilitating the inference process based on both the structure of the logic program and the probabilities of the individual facts.

A **DeepProbLog program** extends the traditional ProbLog framework by incorporating neural predicates through a set of **ground neural ADs (nADs)**. An AD, or **annotated disjunction**, is an expression of the form:

$$p_1 :: h_1; \ldots; p_n :: h_n \ :- \ b_1, \ldots, b_m,$$

where p_i represents the probability that the atom h_i holds, subject to the constraint that $\sum p_i = 1$. The atoms h_i represent the possible outcomes, while the literals b_j form the body of the rule. The semantics of this expression are that, whenever all body literals b_j are true, one of the head atoms h_i will be true with probability p_i, while the other head atoms will be false. This allows for the expression of probabilistic choices within the logic program.

As an extension, **nADs (neural ADs)** integrate neural network outputs into this probabilistic reasoning. Neural networks can produce outputs that represent probabilistic interpretations, which are then encapsulated as part of the logical reasoning process in DeepProbLog. This combination allows DeepProbLog to perform probabilistic reasoning not only based on logical rules, but also based on the predictions of neural networks, thus enabling the framework to handle both symbolic and perceptual data.

For example, the following nAD is the core of our experiment involving single-digit addition problems. It showcases how neural predicates can be incorporated into DeepProbLog to reason about digit images:

$$nn(m_{\text{digit}}, \boxed{}, [0, \ldots, 9]) :: \texttt{digit}(\boxed{}, 0); \ldots; \texttt{digit}(\boxed{}, 9).$$

Figure 2: nADs for single digit images

This nAD demonstrates the use of neural networks to predict the digits in a single-digit addition problem. The probabilities p_i are generated from the neural network's output, which is then combined with logical reasoning to infer the sum of the digits. This process illustrates the seamless integration of neural network predictions into a logical reasoning system, which is the key innovation of DeepProbLog.

Further details about the operation and mechanics of a DeepProbLog program, including its applications and use in complex reasoning tasks, will be explored in depth in Section 3. The upcoming sections will also address the formal semantics and the probabilistic inference algorithms used in DeepProbLog, providing a comprehensive understanding of its capabilities.

2 RELATED WORK

The neuro-symbolic reasoning literature has made significant strides in exploring methods for combining logical reasoning with neural networks [8, 9]. Traditional approaches typically attempt to encode logical terms in Euclidean space, allowing neural networks to approximate logical reasoning processes. However, these methods face limitations in several key areas. Firstly, they are unable to natively support probabilistic reasoning or perception, which are crucial for handling uncertainty and sensory data. Secondly, these methods are often constrained to non-recursive and acyclic logical programs, which limits their applicability to more complex, recursive tasks [10]. In contrast, DeepProbLog takes a fundamentally different approach, integrating neural networks into a probabilistic logic framework. This integration retains the full expressiveness of both logical and probabilistic reasoning, while also harnessing the powerful representational capabilities of deep learning.

Recent advancements have focused on developing differentiable frameworks for logical reasoning, aiming to bridge the gap between symbolic logic and neural learning. For example, [2] introduced a differentiable framework by implementing Prolog's theorem-proving procedure in a differentiable manner. This was further extended with the learning of sub-symbolic representations of

existing symbols, enabling the framework to handle noisy data. Unlike their approach, which primarily uses logic to construct a neural network and focuses on learning sub-symbolic representations, DeepProbLog emphasizes a tighter integration between neural and logical components. In DeepProbLog, parameter learning occurs simultaneously for both neural networks and logical rules, creating a more unified framework for reasoning and perception.

Another notable approach is introduced by [3], who developed a framework to compile a tractable subset of logic programs into differentiable functions. These functions are then executed within a neural network. This method provides an alternative to probabilistic logic programming, but its semantics are less developed compared to frameworks like DeepProbLog, which offer a more comprehensive probabilistic logic integration. While TensorLog represents an important step towards combining logic and neural networks, it lacks the flexibility of DeepProbLog in handling probabilistic reasoning and real-world uncertainties.

A different line of research has focused on incorporating background knowledge as a regularization mechanism during training. For instance, [6] and [7] used first-order logic to specify constraints on neural network outputs. They employed fuzzy logic to measure the degree to which neural network outputs violated these constraints, introducing an additional loss term that acts as a regularizer. This regularization encourages the neural network to produce outputs that align with the specified logical constraints. Recent advancements have extended these methods by replacing fuzzy logic with probabilistic logic, making the approach more similar to DeepProbLog. Furthermore, these methods often compile logical formulas into Sentential Decision Diagrams (SDDs) to enhance computational efficiency, similar to DeepProbLog's optimization techniques.

The work of [4] takes a different approach to combining perception with reasoning. Like DeepProbLog, they integrate domain knowledge in the form of purely logical Prolog rules with neural network outputs. However, the key distinction is that while DeepProbLog handles uncertainty in neural network outputs using probabilistic reasoning, their approach revises the hypothesis iteratively. They replace the neural network's output with anonymous variables until a consistent hypothesis is formed. This iterative hypothesis revision is computationally intensive and less suited to scenarios where real-time decision-making is critical, as compared to the probabilistic reasoning employed by DeepProbLog, which directly incorporates uncertainty into the reasoning process.

A concept closely aligned with DeepProbLog is presented in [1], who introduced a neural network architecture for visual question answering (VQA). This architecture is composed of smaller modules, each responsible for a specific sub-task, such as object detection or counting. The composition of these modules is guided by the linguistic structure of the questions being answered. In contrast, DeepProbLog utilizes logic programs to connect neural network outputs, allowing for more flexible and expressive combinations of reasoning and perception. While the modular VQA approach has inspired various works in the domain of neural-symbolic reasoning, DeepProbLog's use of probabilistic logic provides a more robust framework for handling uncertainty and integrating symbolic and sub-symbolic reasoning in a unified manner. These successes have inspired a series of developments aimed at incorporating probabilistic logic into fundamental deep learning primitives, thereby enhancing the capacity of neural-symbolic systems to reason about the real world under uncertainty.

In summary, the neuro-symbolic reasoning literature has evolved significantly, with various approaches addressing the challenges of integrating symbolic logic and neural networks. While many of these approaches have contributed important

insights, DeepProbLog stands out by combining probabilistic reasoning, logical inference, and deep learning into a coherent and flexible framework. Its ability to simultaneously handle perception, reasoning, and uncertainty makes it a powerful tool for tackling complex real-world tasks. Future research directions may focus on further optimizing this integration, especially in areas such as recursive reasoning and the efficient handling of larger-scale datasets.

3 FRAMEWORK

3.1 DeepProbLog Inference with SDD

In this section, we will explain how the DeepProbLog model is used to handle specific queries for inference tasks. To begin, it is essential to first introduce how inference works in ProbLog, as the inference process in DeepProbLog closely follows that of ProbLog, with some additional steps for neural predicates.

The inference process generally proceeds in four main steps. To illustrate this, we use a simple query example that shows the Sentential Decision Diagram (SDD) structure (Figure 8). The steps are as follows:

(1) **Grounding**: The first step is to ground the logic program with respect to the query. This involves generating all ground instances of clauses in the program that are relevant to the query. These grounded instances form the foundation of the logical inference process.
(2) **Logic Rewriting**: Next, we rewrite the grounded logic program into a propositional logic formula. This formula expresses the truth value of the query in terms of the truth values of the probabilistic facts involved in the program. The logic formula represents the relationships between different facts and how they contribute to the overall query.
(3) **SDD Compilation**: The logic formula is then compiled into a Sentential Decision Diagram (SDD). An SDD is a structured representation that allows for efficient evaluation of the query by organizing the computation of probabilities in a hierarchical manner.
(4) **Evaluation**: Finally, the SDD is evaluated from the bottom up. The evaluation starts at the leaves of the SDD, where the probability labels provided by the program are located. The final success probability of the query is calculated by performing:
    - **Addition** at every OR-node, and
    - **Multiplication** at every AND-node.

To give a more concrete example, consider the following query:

addition(train(711), train(10783), 2)

In this query, `train(711)` and `train(10783)` represent handwritten digit images. The query asks whether the sum of these two images equals 2. The Sentential Decision Diagram (SDD) structure for this query is shown in Figure 8.

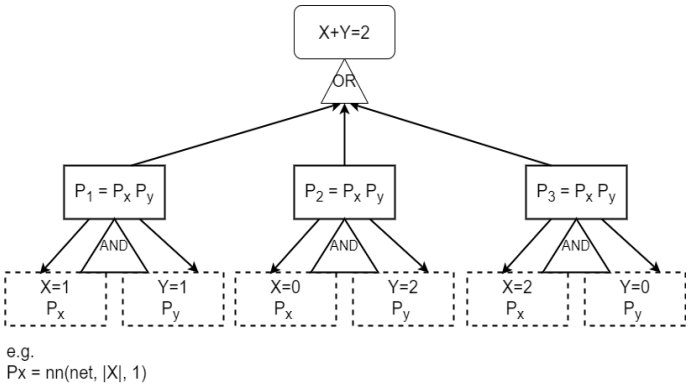

Figure 3: SDD structure

As shown in Figure 8, the neural network provides the probability distribution for each possible digit represented by the image. In this particular example, the sum of 2 can be grounded into three possible instances, each associated with a corresponding probability. These grounded instances are represented as conjunctions for each clause in

the logic program and are combined using disjunctions for the final calculation.

The original SDD structures for this inference are displayed in the diagram. These SDDs provide an efficient way to compute the overall probability of the query by reducing the complexity of evaluating multiple probabilistic facts.

Inference in DeepProbLog follows the same procedure as described above, with the exception of an additional step required for neural predicates. Specifically, whenever a neural predicate is encountered during grounding, a forward pass through the neural network is performed. The input images are fed into the neural network, and the resulting scores from the softmax output layer are used as the probabilities for the ground annotated disjunction (AD). This seamless integration of neural network predictions into the logical inference process is one of the key innovations of DeepProbLog, allowing it to handle both symbolic and perceptual data in a unified framework.

3.2 Learning Process

3.2.1 Learning Pipeline:

To begin, we introduce the learning pipeline of the entire DeepProbLog program. As presented in [14], the learning objective in DeepProbLog is formulated as a parameter optimization problem. The goal is to find the optimal parameters $\vec{x}$ that minimize the error between the predicted and target probabilities across a set of queries Q, defined by the following objective function:

$$\arg\min_{\vec{x}} \frac{1}{|Q|} \sum_{(q,p)\in Q} L(P_{\mathcal{X}=\vec{x}}(q), p),$$

where $\mathcal{X}$ represents the set of parameters being optimized, Q is a set of query-probability pairs (q, p), with q representing a query and p representing its corresponding probability. The loss function L measures the difference between the predicted probability $P_{\mathcal{X}=\vec{x}}(q)$ and the target probability p, typically using a measure such as cross-entropy or mean squared error.

Figure 8 provides a visual representation of the learning pipeline. The process involves several steps:

(1) **Grounding**: First, the DeepProbLog program is grounded with respect to the given query. Grounding generates all ground instances of the logic program that are relevant to the query.
(2) **Parameter Initialization**: During grounding, parameters for neural Annotated Disjunctions (nADs) are retrieved. The parameters include the initial weights of the neural network and any additional parameters required for the probabilistic logic.
(3) **Rewriting and Compilation**: The program is then rewritten and compiled into a Sentential Decision Diagram (SDD), which allows for efficient probabilistic reasoning.
(4) **Loss and Gradient Computation**: The loss function is computed based on the prediction error, and the gradients of the loss with respect to the neural network parameters are calculated.
(5) **Parameter Update**: Finally, the parameters of the neural network are updated based on the computed gradients using gradient descent.

3.2.2 Gradient Semiring:

The use of a **gradient semiring** is central to the learning process in DeepProbLog. Instead of relying on traditional algorithms such as Expectation Maximization (EM), we apply gradient descent to update the parameters. To handle the probabilistic logic components, DeepProbLog builds on the concept of **Algebraic ProbLog (aProbLog)** [12], which extends ProbLog by introducing the ability to process arbitrary commutative semirings. The gradient semiring is a specific type of semiring that tracks both the value of a probability and its derivative with respect to the parameters.

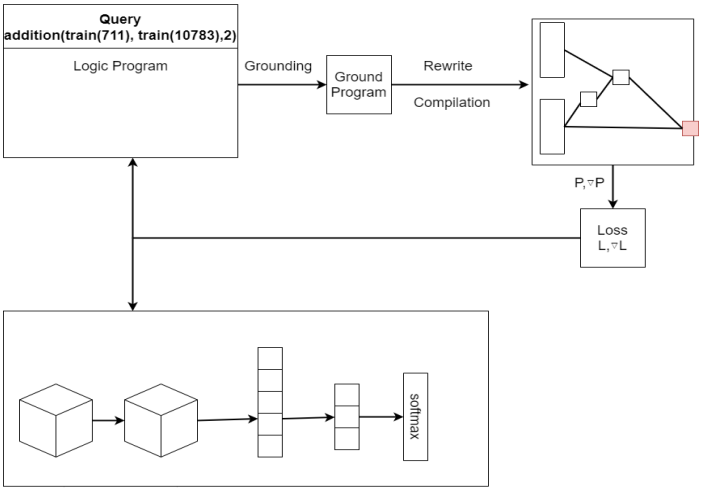

Figure 4: DeepProbLog Learning Pipeline

3.2.3 Definition of Semirings:
A **commutative semiring** is defined by the tuple:

$$(\mathcal{A}, \bigoplus, \bigotimes, e^{\oplus}, e^{\otimes}),$$

where: - $\mathcal{A}$ is the set of elements in the semiring (in our case, probabilities and gradients), - $\bigoplus$ is the addition operation, - $\bigotimes$ is the multiplication operation, - $e^{\oplus}$ is the additive identity (neutral element for addition), - $e^{\otimes}$ is the multiplicative identity (neutral element for multiplication).
In the context of DeepProbLog, we work with **gradient semirings**, where each element in the semiring represents both a value and its gradient. The operations $\bigoplus$ and $\bigotimes$ for the gradient semiring are defined as follows:

$$(a_1, a_2) \bigoplus (b_1, b_2) = (a_1 + b_1, a_2 + b_2),$$

$$(a_1, a_2) \bigotimes (b_1, b_2) = (a_1 b_1, a_2 b_1 + a_1 b_2),$$

where: - (a_1, a_2) and (b_1, b_2) represent pairs consisting of a value and its gradient, respectively. - Addition ($\bigoplus$) combines the values and gradients independently. - Multiplication ($\bigotimes$) follows the product rule from calculus, combining both the values and their gradients in a manner consistent with the chain rule.
The neutral elements for these operations are:

$$e^{\oplus} = (0, 0), \quad e^{\otimes} = (1, 0).$$

Thus, the gradient semiring allows for the simultaneous computation of the probabilities and their corresponding gradients as we propagate through the program's logical and probabilistic structure. This makes it possible to efficiently compute both the value of a query and the gradient of that value with respect to the parameters.

3.2.4 Gradient Descent in DeepProbLog:
To apply gradient descent in this framework, we first convert the DeepProbLog program into an aProbLog program. Each probabilistic fact in the program, represented as $p :: f$, is extended to include both the probability p and its gradient $\frac{\partial p}{\partial x_i}$ with respect to all parameters x_i. The objective is to update the parameters $\vec{x}$ such that the overall loss is minimized.
The gradients are computed as follows: 1. For each probabilistic fact, the gradient of the loss with respect to the output of the neural network is calculated. 2. The computed gradient is then propagated back through the neural network using **backpropagation**, which updates the internal parameters of the network (weights, biases, etc.).
In DeepProbLog, neural predicates serve as the connection point between the logic and neural components. While the logic side of the model computes gradients with respect to the neural network's output, it remains agnostic to the internal structure of the neural network. However, these computed gradients are sufficient to initiate backpropagation in the neural network, allowing the parameters to be updated accordingly.
The **Adam optimizer** is used to perform the gradient-based updates. Adam is an adaptive learning rate optimizer that combines the advantages of **momentum** and **RMSProp**. The update rule for each parameter x_i in Adam is given by:

$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t, \quad v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2,$$

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t}, \quad \hat{v}_t = \frac{v_t}{1-\beta_2^t},$$
$$x_i = x_i - \alpha \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon},$$

where: - m_t and v_t are estimates of the first and second moments of the gradient g_t at timestep t, - α is the learning rate, - β_1 and β_2 control the decay rates for the moment estimates, and - ϵ is a small constant to prevent division by zero.
By utilizing Adam, DeepProbLog ensures that the learning process is both efficient and stable, particularly when handling the complex interplay between the neural and logical components. The use of the softmax output layer in the neural network ensures that the probabilities remain normalized during learning, which simplifies the overall parameter update process.

4 EXPERIMENT

4.1 Experiment Set-up

We performed all of our experiments on a MacBook Pro with a 2.3GHz Dual-Core Intel i5 processor and 8 GB of RAM. The operating system was macOS, and the program, along with related data, is posted on GitHub: https://github.com/KangruiRen0102/Neural-Symbolic-Reasoning-for-Handwritten-Expression-Evaluation.

Given the resource limitations of the hardware, training efficiency and computation time were carefully monitored. The deep learning model was integrated with DeepProbLog to handle both symbolic reasoning and neural network inference.

For the training of the neural networks, we used the Adam optimizer with an initial learning rate of 0.001 and a batch size of 32. The training process spanned 30,000 iterations, and accuracy and loss were recorded at every 100th iteration. For the digit classification, we used a simple neural network with two fully connected layers trained on the MNIST dataset. For the shape classification, a similar network structure was employed, but fine-tuned to distinguish between circles, triangles, and squares from the shape dataset.

The purpose of these experiments is to evaluate the system's ability to perform probabilistic reasoning in the presence of unknown symbols (represented by shape images). We investigate whether the system can infer the correct sums when both digits and shape images are provided as input, and whether it can accurately deduce the numerical values assigned to unknown shapes.

4.2 Problem Statement

As stated in Section 2, the experiment is divided into three consecutive tasks to test the system's reasoning capabilities. The overall goal is to verify whether DeepProbLog can infer sums accurately when interference images (shape images representing unknown variables) are introduced. Additionally, we aim to test whether DeepProbLog can identify the numerical values of the unknown shapes after sufficient training.
To simplify the process, we use a set of shape images to represent unknowns. The three types of shapes used in the experiments are circles, triangles, and squares, and these shapes serve as the placeholders for unknown numbers in the arithmetic expressions.

Task1: Single-shape Inference. In this task, we test the system's ability to perform forward reasoning when a single interference image (shape image) is introduced. Rather than using labeled digit images, we train the system on pairs of images, where each pair is labeled with the sum of the two images' values. One input image is a handwritten digit, and the other is a shape image representing an unknown number. For example, in the query **addition([|Square|],[|3|],6)**, we infer that the square represents the number 3, as the equation $X + 3 = 6$ implies $X = 3$.
The key objective in this task is to study how the accuracy of image recognition influences the final

sum result, as well as to compare these results with the original single-digit addition tasks. We expect interference from shape images to introduce more variance in the results.

Task2: Multi-shape Inference. In this task, we expand the experiment to test the system's ability to reason with multiple interference images. For example, given the query **addition([|Circle|,|2|],[|3|,|Triangle|],48)**, the system should infer that the circle represents 1 and the triangle represents 6. This task is designed to test the system's ability to generalize from single-shape tasks to more complex multi-shape reasoning tasks.
We also aim to study how the size of the training dataset affects performance. Specifically, we test whether DeepProbLog can still maintain high accuracy in multi-shape tasks after being trained primarily on single-shape data.

Task3: Multi-shape Reverse Inference. In this task, we perform similar operations to those in Task 2, but instead of verifying the accuracy of the sum, we aim to deduce the exact values of multiple unknowns. This task presented more challenges during our experiments. The ProbLog engine encountered difficulties in processing additional inputs with additive reasoning, especially when attempting to perform back inference with multiple unknowns.
For example, in the query **addition([|Circle|,|2|],[|3|,|Triangle|],48, 1, 6)**, the engine was unable to process this input due to limitations in the logical rules. We discovered that the gradient semiring also required modifications to allow for back inference. This task will remain a topic for future work.

4.3 Data Collection

The dataset used for the experiments consisted of two distinct sets: one for training on handwritten digits, and the other for training on shape images.

- **Handwritten Digit Data**: The MNIST dataset was used as the source for handwritten digit images. This dataset contains 60,000 training images and 10,000 testing images, each a 28x28 grayscale image of a digit between 0 and 9. All images were normalized and preprocessed before being input into the neural network.
- **Shape Data**: The shape images were collected from a Kaggle dataset containing 300 images of circles, squares, and triangles. Each shape was assigned a numeric value from a uniform distribution and used as the unknown number in the arithmetic expressions. These images were preprocessed similarly to the MNIST images, ensuring consistency across datasets. During the training data generation process, a number was randomly assigned to each shape, which remained consistent throughout the dataset.

The data was split into training and test sets for each task:

- **Task1 Single-shape Inference**: Training and test data were structured as 3-ary tuples, where the first element is a single image array containing a shape, the second element is a digit image, and the third element is the sum of the two numbers. For example, **addition([|Circle(20)|],[|7|],9)**.
- **Task2 Multi-shape Inference**: Two datasets were generated for training. The first dataset contained only single-shape data (similar to Task 1), while the second dataset included multiple shape images in each input array. The test data was consistent across both datasets. An example from the multi-shape dataset is **addition([|Square(2)|,|9|],[|3|,|Circle(8)|],68)**. This setup allows us to test the system's ability to generalize from single-shape tasks to more complex scenarios.
- **Task3 Multi-shape Reverse Inference**: We reused the training data from Task 2, but modified the test data to include additional information about the numerical values of the unknown shapes. These 5-ary tuples included the exact value each shape represented in the arithmetic

expressions. For example, **addition([|Circle(20)|, |Square(2)|],[|3|,|Triangle(5)|],68, 1, 6)**.

4.4 Evaluation

Figure 5 compares the accuracy of the original single-digit addition task with that of the single-shape addition task (Task 1) after 30,000 iterations.

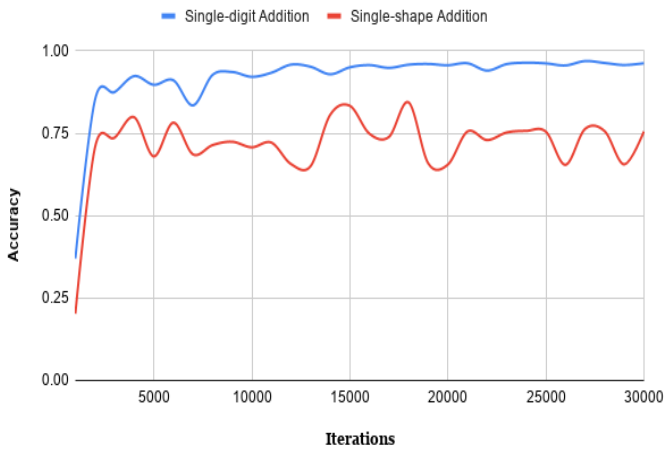

Figure 5: Single-shape Accuracy Comparison

As shown in Figure 5, the original single-digit addition task achieved an accuracy of approximately 96%, while the single-shape addition task only reached 79 %. The fluctuation in accuracy for single-shape addition suggests that interference images introduce instability into the neural network's probability distribution, making it harder for backpropagation to improve the model's accuracy on shape images.

Figure 6 compares the loss for both tasks. The loss for the single-shape addition task is consistently higher than that of the single-digit addition task, indicating that interference images lead to both lower accuracy and higher loss. The model struggled to generalize well from the shape images, as indicated by the greater instability in the loss curve.

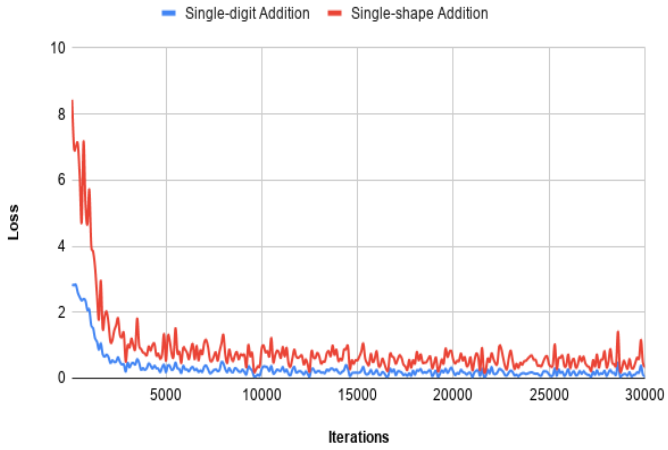

Figure 6: Single-shape Loss Comparison

Next, we examine the results of Task 2. Figures 7 and 8 compare the performance of the model on multi-shape tasks when trained with single-shape data versus multi-shape data. As shown in Figure 8, training on multi-shape data yields higher accuracy, indicating that DeepProbLog struggles to generalize from single-shape data. However, training with multi-shape data significantly increases computation time—by a factor of 10—due to the increased complexity in grounding and calculating the loss and gradients. This phenomenon is understandable as the grounding and loss calculation process, as described in Section 3.2, becomes more computationally intensive with larger input sizes, causing the overall process to slow down significantly.

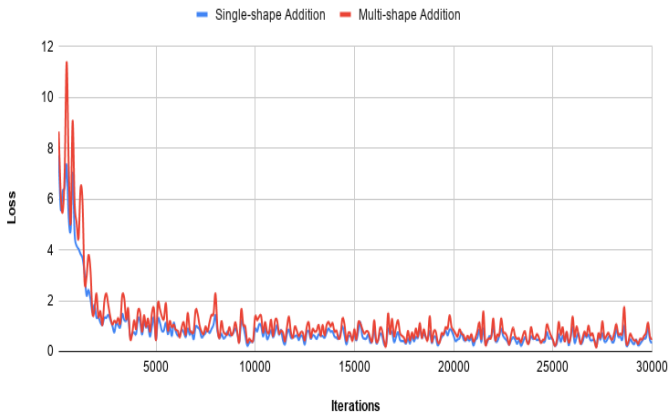

Figure 7: Loss for Multi-shape Addition and Single-shape Addition

Figure 7 shows the loss curves for both single-shape addition and multi-shape addition when trained using the single-shape dataset. As we can see, the loss remains similar throughout the training process, indicating that the model's performance is highly dependent on the number of interference images in the input set. Even when the same training set is used, the multi-shape test data produces non-ideal results, highlighting the limitations of the model in handling multiple unknowns.

Figure 8: Accuracy for two types of training in Multi-shape Addition

In Figure 8, we compare the accuracy of testing the same dataset with two different training sets: one trained on single-shape data and the other on multi-shape data. It is clear that training on multi-shape data provides higher accuracy in handling interference images. However, it also comes with the cost of increased training time and computational complexity. This shows that while DeepProbLog can perform well when explicitly trained for multi-shape tasks, it lacks extensibility when it only has single-shape training data.

4.4.1 Analysis and Future Work. While the current experiment setup allowed us to evaluate DeepProbLog's performance in reasoning under interference, there are clear areas for improvement, particularly in Task 3 (multi-shape reverse inference). The limitations of the ProbLog engine in handling additional input complexity and the constraints of the gradient semiring suggest that further work is needed to fully enable back inference with unknown variables. Future work will focus on:
- **Improving the ProbLog engine's handling of multi-variable additive reasoning**: Modifying the logical rules and semiring calculations to support more complex inference scenarios. - **Extending the dataset**: Increasing the variety and number of shapes and their associated numeric values to better simulate real-world scenarios. - **Optimizing the computational efficiency of DeepProbLog**: Introducing techniques such as batch processing and parallelization to reduce the time complexity for multi-shape tasks.
These improvements will be essential for developing a more robust framework capable of handling more complex reasoning tasks with unknown variables in symbolic and neural contexts.

5 CONCLUSION

This report provides an analysis of the advantages and capabilities of DeepProbLog as a language that effectively integrates logical reasoning with neural networks. We explored its operation through a comprehensive examination of the reasoning

pipeline, with a specific focus on reverse inference tasks. Our primary objective during the experimental phase was to assess DeepProbLog's ability to handle reverse reasoning, and while the third task encountered challenges, it revealed critical limitations in the gradient semiring approach. The ProbLog engine also requires further development to accommodate more complex logical rules and multi-variable scenarios.

In contrast, the first two tasks demonstrated DeepProbLog's robust reasoning capabilities, particularly in handling inference even when interference from unknown elements, such as shape images, was introduced. The system maintained a high level of accuracy, although a notable decrease in performance was observed as the interference increased. One potential solution to mitigate this decline is to expand the size and diversity of the training dataset, particularly by increasing the digit length, which could enhance the model's generalization and resistance to noisy inputs.

Overall, this study underscores both the strengths and current limitations of DeepProbLog. While the language excels in basic and moderately complex reasoning tasks, future work must address its scalability and adaptability to more advanced and intricate symbolic reasoning challenges, particularly in reverse inference and complex arithmetic scenarios.

ACKNOWLEDGEMENT

This work is supervised by Prof. Xujie Si at McGill University.

REFERENCES

[1] Jacob Andreas, Marcus Rohrbach, Trevor Darrell, and Dan Klein. 2016. Neural module networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 39–48.

[2] Matko Bošnjak, Tim Rocktäschel, Jason Naradowsky, and Sebastian Riedel. 2017. Programming with a differentiable forth interpreter. In *International conference on machine learning*. PMLR, 547–556.

[3] William W Cohen, Fan Yang, and Kathryn Rivard Mazaitis. 2017. Tensorlog: Deep learning meets probabilistic dbs. *arXiv preprint arXiv:1707.05390* (2017).

[4] Wang-Zhou Dai, Qiu-Ling Xu, Yang Yu, and Zhi-Hua Zhou. 2018. Tunneling neural perception and logic reasoning through abductive learning. *arXiv preprint arXiv:1802.01173* (2018).

[5] Luc De Raedt, Angelika Kimmig, and Hannu Toivonen. 2007. ProbLog: A Probabilistic Prolog and Its Application in Link Discovery.. In *IJCAI*, Vol. 7. Hyderabad, 2462–2467.

[6] Michelangelo Diligenti, Marco Gori, and Claudio Sacca. 2017. Semantic-based regularization for learning and inference. *Artificial Intelligence* 244 (2017), 143–165.

[7] Ivan Donadello, Luciano Serafini, and Artur D'Avila Garcez. 2017. Logic tensor networks for semantic image interpretation. *arXiv preprint arXiv:1705.08968* (2017).

[8] Artur S d'Avila Garcez, Krysia B Broda, and Dov M Gabbay. 2012. *Neural-symbolic learning systems: foundations and applications*. Springer Science & Business Media.

[9] Barbara Hammer and Pascal Hitzler. 2007. *Perspectives of neural-symbolic integration*. Vol. 77. Springer.

[10] Steffen Hölldobler, Yvonne Kalinke, and Hans-Peter Störr. 1999. Approximating the semantics of logic programs by recurrent neural networks. *Applied Intelligence* 11, 1 (1999), 45–58.

[11] William W Cohen Fan Yang Kathryn and R Mazaitis. 2018. Tensorlog: Deep learning meets probabilistic databases. *Journal of Artificial Intelligence Research* 1 (2018), 1–15.

[12] Angelika Kimmig, Guy Van den Broeck, and Luc De Raedt. 2011. An algebraic Prolog for reasoning about possible worlds. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 25.

[13] Robin Manhaeve, Sebastijan Dumancic, Angelika Kimmig, Thomas Demeester, and Luc De Raedt. 2018. Deepproblog: Neural probabilistic logic programming. *Advances in Neural Information Processing Systems* 31 (2018), 3749–3759.

[14] Luc De Raedt, Kristian Kersting, Sriraam Natarajan, and David Poole. 2016. Statistical relational artificial intelligence: Logic, probability, and computation. *Synthesis Lectures on Artificial Intelligence and Machine Learning* 10, 2 (2016), 1–189.

[15] Tim Rocktäschel and Sebastian Riedel. 2017. End-to-end differentiable proving. *arXiv preprint arXiv:1705.11040* (2017).