A-NeSI: A Scalable Approximate Method for Probabilistic Neurosymbolic Inference

Emile van Krieken[1] **Thiviyan Thanapalasingam**[2]

Jakub M. Tomczak[3] **Frank van Harmelen**[1]

Annette ten Teije[1]
[1]Vrije Universiteit Amsterdam [2]University of Amsterdam [3]TU Eindhoven
{e.van.krieken,annette.ten.teije,frank.van.harmelen}@vu.nl
t.singam@uva.nl

Abstract

We study the problem of combining neural networks with symbolic reasoning. Frameworks for Probabilistic Neurosymbolic Learning (PNL) like DeepProbLog perform exponential-time exact inference that is limited in scalability. We introduce *Approximate Neurosymbolic Inference* (A-NeSI): a new framework for PNL that uses deep generative modelling for scalable approximate inference. A-NeSI 1) performs approximate inference in polynomial time; 2) is trained using data generated by background knowledge; 3) can generate symbolic explanations of predictions; and 4) can guarantee the satisfaction of logical constraints at test time. Our experiments show that A-NeSI is the first end-to-end method to scale the Multi-digit MNISTAdd benchmark to sums of 15 MNIST digits.

1 Introduction

Neurosymbolic methods using probabilistic logics, which we call *Probabilistic Neurosymbolic Learning (PNL)* methods, add probabilities over discrete truth values to maintain all logical equivalences expected from classical logic. However, computing the probability of a query, i.e., performing inference, requires solving the *weighted model counting (WMC)* problem. Exact inference of the WMC problem is exponential Chavira & Darwiche (2008), which significantly limits the kind of tasks we can solve with PNL. Probabilistic circuits Vergari & Van den Broeck (2020) significantly speed up exact inference, but do not escape exponential growth.

We study how to scale PNL to exponentially complex tasks using deep generative modeling Tomczak (2022). Our method *Approximate Neurosymbolic Inference* (A-NeSI) uses two neural networks that perform approximate inference over the WMC problem. The *prediction model* predicts the output of the system, while the *explanation model* computes which worlds best explain a prediction. We use a novel training algorithm to fit both models with data generated using background knowledge. For an overview, see Figure 1.

A-NeSI combines all benefits of neurosymbolic learning with scalability. Our experiments on the Multi-digit MNISTAdd problem Manhaeve et al. (2018) show that, unlike other approaches, A-NeSI scales almost linearly in the number of digits which allows solving the Multi-digit MNISTAdd problem for sums of 15 digits, up from 4 in competing systems. Furthermore, A-NeSI maintains the predictive performance of exact inference, produces explanations of predictions, and guarantees the satisfaction of logical constraints using a novel *symbolic pruner*.

2 Problem setup

First, we introduce our inference problem. We will use the MNISTAdd task from Manhaeve et al. (2018) as a running example to illustrate the introduced concepts of A-NeSI. In this problem, we

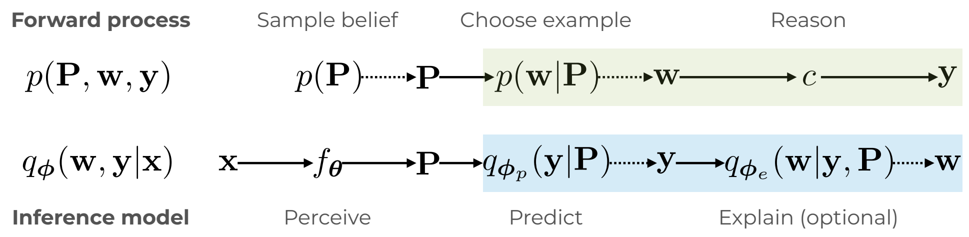

Forward process
Sample belief
Choose example
Reason
$p(\mathbf{P}, \mathbf{w}, \mathbf{y})$
$p(\mathbf{P})$
$\mathbf{P}$
$p(\mathbf{w}|\mathbf{P})$
$\mathbf{w}$
c
$\mathbf{y}$
$q_{\boldsymbol{\phi}}(\mathbf{w}, \mathbf{y}|\mathbf{x})$
$\mathbf{x}$
$f_{\boldsymbol{\theta}}$
$\mathbf{P}$
$q_{\boldsymbol{\phi}_p}(\mathbf{y}|\mathbf{P})$
$\mathbf{y}$
$q_{\boldsymbol{\phi}_e}(\mathbf{w}|\mathbf{y}, \mathbf{P})$
$\mathbf{w}$
Inference model
Perceive
Predict
Explain (optional)

Figure 1: Overview of A-NeSI. The forward process samples a belief $\mathbf{P}$ from a prior, then chooses a world $\mathbf{w}$ for that belief. The symbolic function c computes its output $\mathbf{y}$. The inference model uses the perception model $f_{\boldsymbol{\theta}}$ to find a belief $\mathbf{P}$, then uses the prediction model $q_{\boldsymbol{\phi}_p}$ to find the most likely output $\mathbf{y}$ for that belief. Optionally, the explanation $q_{\boldsymbol{\phi}_e}$ model explains the output.

have to learn to recognize the sum of two MNIST digits using only the sum as a training label. Importantly, we do not provide the labels of the individual MNIST digits.

We introduce four sets representing the spaces of the variables of interest:

1. X is an input space. In MNISTAdd, this is a pair of MNIST digits $\mathbf{x} =$ (,).
2. W is a structured space of k_W discrete choices. Its elements $\mathbf{w} \in W$ are *worlds*: symbolic representations of some $\mathbf{x} \in X$. For (,), the correct world is $\mathbf{w} = (5, 8)$.
3. Y is a structured space of k_Y discrete choices. Elements $\mathbf{y} \in Y$ represent the output of the neurosymbolic system: Given $\mathbf{w} = (5, 8)$, the sum is 13. We decompose $\mathbf{y}$ on the value of each digit, so $\mathbf{y} = (1, 3)$.
4. $\mathbf{P}$ is a *belief* that assigns probabilities to different worlds $\mathbf{w}$ with $p(\mathbf{w}|\mathbf{P}) = \prod_{i=1}^{k_W} \mathbf{P}_{i,w_i}$.

We define a *symbolic reasoning function* $c : W \to Y$ that computes the output $\mathbf{y}$ for some world $\mathbf{w}$. For MNISTAdd, c takes the digits $(5, 8)$, sums them, and decomposes by digit to form $(1, 3)$. These components form the *Weighted Model Counting (WMC)* problem Chavira & Darwiche (2008):

$$p(\mathbf{y}|\mathbf{P}) = \mathbb{E}_{p(\mathbf{w}|\mathbf{P})}[c(\mathbf{w}) = \mathbf{y}] \tag{1}$$

In PNL, we train a perception model $f_{\boldsymbol{\theta}}$ that computes a belief $\mathbf{P} = f_{\boldsymbol{\theta}}(x)$ for an input $\mathbf{x} \in X$. We want $f_{\boldsymbol{\theta}}$ to predict beliefs where the correct world is likely. Note that a possible world is not necessarily the correct world: $\mathbf{w} = (4, 9)$ also sums to 13, but is not a symbolic representation of $\mathbf{x} =$ (,). We are interested in efficiently computing the following quantities:

1. $p(\mathbf{y}|\mathbf{P} = f_{\boldsymbol{\theta}}(\mathbf{x}))$: What outputs are likely for the belief $\mathbf{P}$?
2. $\nabla_{\mathbf{P}} p(\mathbf{y}|\mathbf{P} = f_{\boldsymbol{\theta}}(\mathbf{x}))$: How do we backpropagate through this probability computation?
3. $p(\mathbf{w}|\mathbf{y}, \mathbf{P} = f_{\boldsymbol{\theta}}(\mathbf{x}))$: What are the most likely worlds?

The three quantities above require calculating or estimating the WMC problem of Equation 1. However, exact computation of the WMC is in complexity class #P-hard. This requires an exponential-time computation for each training iteration and test query. Existing PNL methods use probabilistic circuits (PCs) to speed up this computation Vergari & Van den Broeck (2020); Kisa & Van den Broeck (2014). PCs compile a logical formula into a circuit for which many inference queries are linear in the size of the circuit. However, they do not overcome the exponential complexity of the problem.

We can increase the complexity of MNISTAdd exponentially by considering not only the sum of two digits but the sum of two numbers consisting of multiple digits. Each complexity increase translates into a factor 10 additional options to consider for exact inference. While A-NeSI can solve this problem for the sum of two numbers with 15 digits ($N = 15$), exact inference would require enumerating around 10^{15} options for each query.

3 A-NeSI: Approximate Neurosymbolic Inference

To reduce the inference complexity of PNL, we introduce *Approximate Neurosymbolic Inference* (A-NeSI). A-NeSI approximates the three quantities of interest from Section 2, namely $p(\mathbf{y}|\mathbf{P})$, $\nabla_{\mathbf{P}} p(\mathbf{y}|\mathbf{P})$ and $p(\mathbf{w}|\mathbf{y}, \mathbf{P})$, using neural networks. We give an overview of our method in Figure 1.

3.1 Inference models

A-NeSI uses an *inference model* $q_{\boldsymbol{\phi}}$ defined as

$$q_{\boldsymbol{\phi}}(\mathbf{w}, \mathbf{y}|\mathbf{P}) = q_{\boldsymbol{\phi}_p}(\mathbf{y}|\mathbf{P}) q_{\boldsymbol{\phi}_e}(\mathbf{w}|\mathbf{y}, \mathbf{P}). \tag{2}$$

We call $q_{\boldsymbol{\phi}_p}(\mathbf{y}|\mathbf{P})$ the *prediction model* and $q_{\boldsymbol{\phi}_e}(\mathbf{w}|\mathbf{y}, \mathbf{P})$ the *explanation model.* The prediction model should approximate the WMC problem of Equation 1, while the explanation model should predict likely worlds $\mathbf{w}$ given outputs $\mathbf{y}$ and beliefs $\mathbf{P}$. One way to model $q_{\boldsymbol{\phi}}$ is by factorizing the distributions over the structures of W and Y:

$$q_{\boldsymbol{\phi}}(\mathbf{y}|\mathbf{P}) = \prod_{i=1}^{k_Y} q_{\boldsymbol{\phi}_p}(y_i|\mathbf{y}_{1:i-1}, \mathbf{P}) \quad q_{\boldsymbol{\phi}}(\mathbf{w}|\mathbf{y}, \mathbf{P}) = \prod_{i=1}^{k_W} q_{\boldsymbol{\phi}_e}(w_i|\mathbf{y}, \mathbf{w}_{1:i-1}, \mathbf{P}) \tag{3}$$

The full factorization ensures that the inference model is flexible enough to model the problem. We can use simpler models if we know of independences between variables in W and Y. The complete factorization is linear in $k_W + k_Y$. Therefore, a forward pass of the inference model is polynomial in problem size instead of exponential. We use a beam search to find the most likely prediction when testing our model.

We use the prediction model to train the perception model $f_{\boldsymbol{\theta}}$ given a dataset $\mathcal{D}$ of tuples $(\mathbf{x}, \mathbf{y})$. Our novel loss function trains the perception model by backpropagating through the prediction model:

$$\mathcal{L}_{Perc}(\mathcal{D}, \boldsymbol{\theta}) = -\log q_{\boldsymbol{\phi}_p}(\mathbf{y}|\mathbf{P} = f_{\boldsymbol{\theta}}(\mathbf{x})), \quad \mathbf{x}, \mathbf{y} \sim \mathcal{D} \tag{4}$$

This gives biased gradients due to the error in $q_{\boldsymbol{\phi}_p}$, but the only variance is in sampling from the training dataset.

3.2 Training the inference model

We define two variants for training the inference model: *Prediction-only* and *explainable*. We first define the forward process that uses the symbolic reasoning function c to generate training data:

$$p(\mathbf{w}, \mathbf{y}|\mathbf{P}) = p(\mathbf{w}|\mathbf{P}) p(\mathbf{y}|\mathbf{w}, \mathbf{P}) = p(\mathbf{w}|\mathbf{P})(c(\mathbf{w}) = \mathbf{y}) \tag{5}$$

We take some example world $\mathbf{w}$ and deterministically compute its associated output $\mathbf{y} = c(\mathbf{w})$. $p(\mathbf{w}, \mathbf{y}|\mathbf{P})$ is 0 if $c(\mathbf{w}) \neq \mathbf{y}$ (that is, $\mathbf{w}$ is not a possible world of $\mathbf{y}$).

The belief prior $p(\mathbf{P})$ allows us to generate beliefs $\mathbf{P}$ for the forward process. That is, we generate training data for the inference model using the joint $p(\mathbf{P}, \mathbf{w}) = p(\mathbf{P})p(\mathbf{w}|\mathbf{P})$: Sample $\mathbf{P}, \mathbf{w} \sim p(\mathbf{P}, \mathbf{w})$ and deterministically compute $\mathbf{y} = c(\mathbf{w})$. The belief prior allows us to train the inference model with synthetic data. The prior and the forward process define everything $q_{\boldsymbol{\phi}}$ needs to learn. In Section C.2, we discuss considerations on what an applicable prior might be.

Prediction-only variant: Only train the prediction model $q_{\boldsymbol{\phi}_p}(\mathbf{y}|\mathbf{P})$. We use the samples generated by the process described in the previous paragraph. We minimize the expected cross entropy between $p(\mathbf{y}|\mathbf{P})$ and $q_{\boldsymbol{\phi}_p}(\mathbf{y}|\mathbf{P})$ over the prior $p(\mathbf{P})$ (see Appendix A):

$$\mathcal{L}_{Pred}(\boldsymbol{\phi}_p) = -\log q_{\boldsymbol{\phi}_p}(c(\mathbf{w})|\mathbf{P}), \quad \mathbf{P}, \mathbf{w} \sim p(\mathbf{P}, \mathbf{w}) \tag{6}$$

We estimate the expected cross entropy using samples from $p(\mathbf{P}, \mathbf{w})$. We use the sampled world $\mathbf{w}$ to compute the output $\mathbf{y} = c(\mathbf{w})$ and increase its probability under $q_{\boldsymbol{\phi}_p}$. Importantly, we use generated data to evaluate this loss function.

Explainable variant: Use both the prediction model $q_{\boldsymbol{\phi}_p}(\mathbf{y}|\mathbf{P})$ and the explanation model $q_{\boldsymbol{\phi}_e}(\mathbf{w}|\mathbf{y}, \mathbf{P})$. Note that both factorizations of the joint should have the same probability mass: $p(\mathbf{w}, \mathbf{y}|\mathbf{P}) = q_{\boldsymbol{\phi}}(\mathbf{w}, \mathbf{y}|\mathbf{P})$. Therefore, we use a novel *joint matching* loss inspired by the theory of GFlowNets, and the trajectory balance loss Bengio et al. (2021; 2022); Malkin et al. (2022a). For an

	N=1	N=2	N=4	N=15
	Symbolic prediction			
DeepProbLog	97.20 ± 0.50	95.20 ± 1.70	T/O	T/O
DPLA*	88.90 ± 14.80	83.60 ± 23.70	T/O	T/O
DeepStochLog	**97.90 ± 0.10**	**96.40 ± 0.10**	**92.70 ± 0.60**	T/O
Embed2Sym	97.62 ± 0.29	93.81 ± 1.37	91.65 ± 0.57	60.46 ± 20.36
A-NeSI (predict)	97.66 ± 0.21	95.96 ± 0.38	92.56 ± 0.79	75.90 ± 2.21
A-NeSI (explain)	97.37 ± 0.32	96.04 ± 0.46	92.11 ± 1.06	**76.84 ± 2.82**
	Neural prediction			
Embed2Sym	97.34 ± 0.19	84.35 ± 6.16	0.81 ± 0.12	0.00
A-NeSI (predict)	**97.66 ± 0.21**	95.95 ± 0.38	**92.48 ± 0.76**	54.66 ± 1.87
A-NeSI (explain)	97.37 ± 0.32	**96.05 ± 0.47**	92.14 ± 1.05	**61.77 ± 2.37**
Reference	98.01	96.06	92.27	73.97

Table 1: Accuracy of predicting the correct sum for increased complexity on the Multi-digit MNISTAdd task. Reference accuracy approximates the accuracy when using a perception network with 0.99 accuracy using 0.99^{2N}. We provide a source for each baseline in Appendix G.1.

in-depth discussion, see Appendix E. The joint matching loss is a regression of q_{ϕ} onto the true joint p that we compute in closed form:

$$\mathcal{L}_{Expl}(\phi) = \left(\log \frac{q_{\phi}(\mathbf{w}, c(\mathbf{w})|\mathbf{P})}{p(\mathbf{w}|\mathbf{P})}\right)^2, \quad \mathbf{P}, \mathbf{w} \sim p(\mathbf{P}, \mathbf{w}) \tag{7}$$

The sampling process is the same as in the prediction-only variant. We minimize the loss function to *match* the joints p and q_{ϕ}. Unlike a cross-entropy loss, the joint matching loss ensures q_{ϕ} does not become overly confident in a single prediction (Appendix D). See B for pseudocode.

4 Experiments

We perform experiments on the Multi-digit MNISTAdd problem discussed in Section 2. We refer to Appendix G for a detailed description of the experimental setup.

Table 1 reports the accuracy of predicting the sum. For all N, A-NeSI is close to the reference accuracy, so there is no significant performance drop as N increases. For N in 1 to 4, it is slightly outperformed by DeepStochLog, which can not scale to $N = 15$. Accuracy improvements compared to DeepProbLog could come from hyperparameter tuning and longer training times, as A-NeSI approximates DeepProbLog's semantics. However, DeepProbLog does not scale beyond $N = 2$.

With symbolic prediction, A-NeSI remains close to the reference accuracy. With neural prediction, we get the same accuracy for $N = 1$ to $N = 4$, but there is a significant difference for $N = 15$, meaning the prediction network did not perfectly learn the problem. However, the prediction network is much more accurate than a neural network without background knowledge for $N \geq 2$, which shows that A-NeSI allows training a prediction network with high accuracy that can learn on large-scale problems. We present an ablation study and a comparison of inference times in Appendix I.

5 Related work

Probabilistic Neurosymbolic Learning (PNL) A-NeSI can approximate multiple PNL methods De Raedt et al. (2019). DeepProbLog Manhaeve et al. (2018) performs symbolic reasoning by interpreting $\mathbf{w}$ as the ground facts in a Prolog program. It enumerates all possible proofs of a query $\mathbf{y}$ and weights each proof by $p(\mathbf{w}|\mathbf{P})$. NeurASP Yang et al. (2020) is a PNL framework closely related to DeepProbLog, but is based on Answer Set Programming semantics Brewka et al. (2011).

Inference in PNL The approaches above perform exact inference using probabilistic circuits Vergari & Van den Broeck (2020), in particular using SDDs Kisa & Van den Broeck (2014). Probabilistic circuits are fast solutions for exact inference but cannot escape that computation time increases exponentially as the problem size increases. Manhaeve et al. (2021a) considers approximate inference for PNL by only considering the top-k proofs when computing the probabilistic circuit. However, finding those proofs is hard when beliefs have high entropy. A-NeSI ensures computation time is constant irrespective of the entropy of $\mathbf{P}$.

6 Conclusion

We introduced A-NeSI: A scalable method for approximate inference of probabilistic neurosymbolic learning. Our first experiments demonstrated that A-NeSI scales to exponentially challenging domains without a loss in accuracy. A-NeSI is highly flexible and can be extended to include explanations and hard constraints without loss of performance.

References

Yaniv Aspis, Krysia Broda, Jorge Lobo, and Alessandra Russo. Embed2Sym - Scalable Neuro-Symbolic Reasoning via Clustered Embeddings. In *Proceedings of the Nineteenth International Conference on Principles of Knowledge Representation and Reasoning*, pp. 421–431, Haifa, Israel, July 2022. International Joint Conferences on Artificial Intelligence Organization. ISBN 978-1-956792-01-0. doi: 10.24963/kr.2022/44.

Samy Badreddine, Artur d'Avila Garcez, Luciano Serafini, and Michael Spranger. Logic Tensor Networks. *Artificial Intelligence*, 303:103649, February 2022. ISSN 0004-3702. doi: 10.1016/j.artint.2021.103649.

Tomas Balyo, Marijn J. H. Heule, Markus Iser, Matti Järvisalo, Martin Suda, Helsinki Institute for Information Technology, Constraint Reasoning and Optimization research group / Matti Järvisalo, and Department of Computer Science. Proceedings of SAT Competition 2022 : Solver and Benchmark Descriptions. 2022.

Emmanuel Bengio, Moksh Jain, Maksym Korablyov, Doina Precup, and Yoshua Bengio. Flow network based generative models for non-iterative diverse candidate generation. *Advances in Neural Information Processing Systems*, 34:27381–27394, 2021.

Yoshua Bengio, Salem Lahlou, Tristan Deleu, Edward J. Hu, Mo Tiwari, and Emmanuel Bengio. GFlowNet Foundations, August 2022.

Gerhard Brewka, Thomas Eiter, and Mirosław Truszczyński. Answer set programming at a glance. *Communications of the ACM*, 54(12):92–103, 2011.

Mark Chavira and Adnan Darwiche. On probabilistic inference by weighted model counting. *Artificial Intelligence*, 172(6):772–799, April 2008. ISSN 0004-3702. doi: 10.1016/j.artint.2007.11.002.

Xi Chen, Diederik P. Kingma, Tim Salimans, Yan Duan, Prafulla Dhariwal, John Schulman, Ilya Sutskever, and Pieter Abbeel. Variational Lossy Autoencoder, March 2017.

Luc De Raedt, Robin Manhaeve, Sebastijan Dumancic, Thomas Demeester, and Angelika Kimmig. Neuro-symbolic= neural+ logical+ probabilistic. In *NeSy'19@ IJCAI, the 14th International Workshop on Neural-Symbolic Learning and Reasoning*, 2019.

Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. *arXiv:1412.6980 [cs]*, January 2017.

Doga Kisa and Guy Van den Broeck. Probabilistic sentential decision diagrams. *Proceedings of the 14th International Conference on Principles of Knowledge Representation and Reasoning (KR)*, pp. 558–567, 2014.

Yann LeCun and Corinna Cortes. MNIST handwritten digit database. 2010.

Nikolay Malkin, Moksh Jain, Emmanuel Bengio, Chen Sun, and Yoshua Bengio. Trajectory Balance: Improved Credit Assignment in GFlowNets. *arXiv preprint arXiv:2201.13259*, 2022a.

Nikolay Malkin, Salem Lahlou, Tristan Deleu, Xu Ji, Edward Hu, Katie Everett, Dinghuai Zhang, and Yoshua Bengio. GFlowNets and variational inference, October 2022b.

Robin Manhaeve, Sebastijan Dumančić, Angelika Kimmig, Thomas Demeester, and Luc De Raedt. DeepProbLog: Neural probabilistic logic programming. In Samy Bengio, Hanna M Wallach, Hugo Larochelle, Kristen Grauman, Nicolò Cesa-Bianchi, and Roman Garnett (eds.), *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, 3-8 December 2018, Montréal, Canada*, 2018.

Robin Manhaeve, Sebastijan Dumančić, Angelika Kimmig, Thomas Demeester, and Luc De Raedt. Neural probabilistic logic programming in DeepProbLog. *Artificial Intelligence*, 298:103504, 2021a. ISSN 0004-3702. doi: 10.1016/j.artint.2021.103504.

Robin Manhaeve, Giuseppe Marra, and Luc De Raedt. Approximate inference for neural probabilistic logic programming. In *Proceedings of the 18th International Conference on Principles of Knowledge Representation and Reasoning*, pp. 475–486, November 2021b. doi: 10.24963/kr.2021/45.

Thomas Minka. Estimating a dirichlet distribution, 2000.

Connor Pryor, Charles Dickens, Eriq Augustine, Alon Albalak, William Wang, and Lise Getoor. NeuPSL: Neural Probabilistic Soft Logic, June 2022.

Danilo Rezende and Shakir Mohamed. Variational Inference with Normalizing Flows. In *Proceedings of the 32nd International Conference on Machine Learning*, pp. 1530–1538. PMLR, June 2015.

Jakub M. Tomczak. *Deep Generative Modeling*. Springer International Publishing, Cham, 2022. ISBN 978-3-030-93157-5 978-3-030-93158-2. doi: 10.1007/978-3-030-93158-2.

Antonio Vergari and Guy Van den Broeck. Probabilistic circuits: A unifying framework for tractable probabilistic models. 2020.

Thomas Winters, Giuseppe Marra, Robin Manhaeve, and Luc De Raedt. DeepStochLog: Neural Stochastic Logic Programming. In *Proceedings of the First MiniCon Conference*, February 2022.

Zhun Yang, Adam Ishay, and Joohyung Lee. NeurASP: Embracing neural networks into answer set programming. In Christian Bessiere (ed.), *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI-20*, pp. 1755–1762. International Joint Conferences on Artificial Intelligence Organization, July 2020. doi: 10.24963/ijcai.2020/243.

A DERIVATION OF PREDICTION-ONLY VARIANT LOSS

$$\begin{aligned} &\mathbb{E}_{p(\mathbf{P})}\left[-\mathbb{E}_{p(\mathbf{y}|\mathbf{P})}[\log q_{\boldsymbol{\phi}_p}(\mathbf{y}|\mathbf{P})]\right] && (8)\\ =&-\mathbb{E}_{p(\mathbf{P})}\left[\mathbb{E}_{p(\mathbf{w},\mathbf{y}|\mathbf{P})}[\log q_{\boldsymbol{\phi}_p}(\mathbf{y}|\mathbf{P})]\right] && (9)\\ =&-\mathbb{E}_{p(\mathbf{P},\mathbf{w})}\left[\log q_{\boldsymbol{\phi}_p}(c(\mathbf{w})|\mathbf{P})\right] && (10)\\ \mathcal{L}_{Pred}(\boldsymbol{\phi}_p)=&-\log q_{\boldsymbol{\phi}_p}(c(\mathbf{w})|\mathbf{P}),\quad \mathbf{P},\mathbf{w}\sim p(\mathbf{P},\mathbf{w}) && (11)\end{aligned}$$

In line 9, we marginalize out $\mathbf{w}$, and use the fact that $\mathbf{y}$ is deterministic given $\mathbf{w}$.

B PSEUDOCODE FOR THE A-NESI TRAINING ALGORITHM

We describe how to compute this loss in Algorithm 1 and the complete training loop in Algorithm 2.

Algorithm 1 Computing joint matching loss

```
Input: observations $[\mathbf{P}_1, ..., \mathbf{P}_k]$, params $\boldsymbol{\phi}$
fit prior $p(\mathbf{P})$ on $\mathbf{P}_1, ..., \mathbf{P}_k$
$\mathbf{P} \sim p(\mathbf{P})$
$\mathbf{w} \sim p(\mathbf{w}|\mathbf{P})$
$\mathbf{y} \leftarrow c(\mathbf{w})$
return $\boldsymbol{\phi} - \alpha\nabla_{\boldsymbol{\phi}}\left(\log \frac{q_{\boldsymbol{\phi}}(\mathbf{w},\mathbf{y}|\mathbf{P})}{p(\mathbf{w}|\mathbf{P})}\right)^2$ {Equation 7}
```

Algorithm 2 A-NESI training loop

```
Input: dataset $\mathcal{D}$, params $\boldsymbol{\theta}$, params $\boldsymbol{\phi}$
beliefs ← []
while not converged do
    $(\mathbf{x}, \mathbf{y}) \sim \mathcal{D}$
    $\mathbf{P} \leftarrow f_{\boldsymbol{\theta}}(\mathbf{x})$
    update beliefs with $\mathbf{P}$
    $\boldsymbol{\phi} \leftarrow$ Algorithm 1(beliefs, $\boldsymbol{\phi}$)
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha\nabla_{\boldsymbol{\theta}} \log q_{\boldsymbol{\phi}}(\mathbf{y}|\mathbf{P})$
end while
```

C CHALLENGES WHEN APPLYING A-NESI

We have shown that A-NESI is a promising method for approximate inference of the WMC problem. However, there is 'no free lunch': When is A-NESI a good approximation, and when is it not? We discuss three aspects of learning tasks that could make it difficult to learn a strong and efficient inference model:

- **Strong dependencies of variables.** When different variables in world $\mathbf{w}$ are interdependent, finding an informative prior is hard. We discuss potential solutions in Sections C.1 and C.2.
- **Structure in symbolic reasoning function.** MNISTAdd is a task with a simple structure. Learning the inference model will be much more difficult for symbolic reasoning functions c with much less structure. Studying the relation between (lack of) structure and learnability is interesting future work
- **Problem size.** While we did not observe a deviation from the reference accuracy when increasing N in our experiments, fitting the inference model took longer. For high N, we observed A-NESI could not perfectly train the prediction model. We expect the required size of the prediction model to increase as the problem size increases.

C.1 OUTPUT SPACE FACTORIZATION

The factorization of the output space Y introduced in Section 2 is one of the key ideas that allow efficient learning in A-NESI. We will illustrate this with the MNISTAdd example.

Consider increasing the number of digits N that make up a number in the Multi-digit MNISTAdd problem (Section 2). The number of possible outputs (i.e., sums) is $2 \cdot 10^N - 1$. Without factorization, we would need an exponentially large output layer. We solve this by predicting the individual digits of the output so that we need only $N \cdot 10 + 2$ outputs. Embed2Sym Aspis et al. (2022) first used this factorization.

There is an additional benefit to this, namely easier learning. Recognizing a single digit of the sum is easier than recognizing the entire sum. For instance, for the rightmost digit of the sum, only the rightmost digits of the input are relevant.

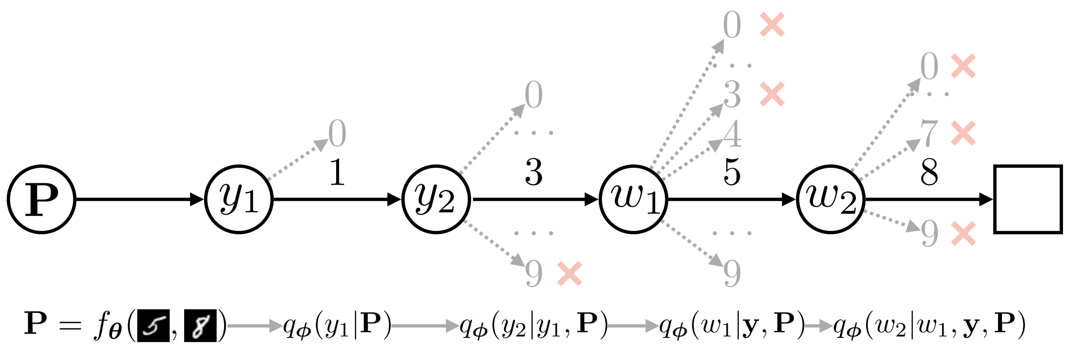
We believe choosing the right factorization will be crucial when applying A-NESI. A general approach is to take the CNF of the symbolic function and predict each clause's truth value. However, this requires grounding the formula, which can be exponential. Another option is to predict for what objects a universally quantified formula holds, which would be linear in the number of objects.

C.2 BELIEF PRIOR DESIGN

How should we choose the $\mathbf{P}$s for which we train $q_{\boldsymbol{\phi}}$? The naive method is to use the perception model $f_{\boldsymbol{\theta}}$, sample some training data $\mathbf{x}_1, ..., \mathbf{x}_k \sim \mathcal{D}$ and train the inference model over $\mathbf{P}_1 = f_{\boldsymbol{\theta}}(\mathbf{x}_1), ..., \mathbf{P}_k = f_{\boldsymbol{\theta}}(\mathbf{x}_k)$. However, this means the inference model is only trained on those $\mathbf{P}$ appearing empirically through the training data! Again, consider the Multi-digit MNISTAdd problem of Section 2. For $N = 15$ we have a dataset of 2000 sums to train on, while there are $2 \cdot 10^{15} - 1$ possible sums. By simulating many beliefs, the inference model sees a much richer set of inputs and outputs, allowing it to generalize.

A better approach is to fit a Dirichlet prior $p(\mathbf{P})$ on $\mathbf{P}_1, ..., \mathbf{P}_k$ that covers all possible combinations of numbers. We choose a Dirichlet prior since it is conjugate to the discrete distributions. For details, see Appendix F. During hyperparameter tuning, we found that the prior needs to be high entropy to prevent the inference model from ignoring the inputs $\mathbf{P}$. Therefore, we regularize the prior with an additional term encouraging high-entropy Dirichlet distributions.

Our current Dirichlet prior considers each dimension in W independent. This assumption works for the MNISTAdd task, but will be too simple for tasks where different variables $\mathbf{w}_i$ are highly dependent, such as in sudoku. For these tasks, we suggest using a prior that can incorporate dependencies between variables, such as a normalizing flow Rezende & Mohamed (2015); Chen et al. (2017) or other deep generative models Tomczak (2022) over a Dirichlet distribution.



P
y_1
y_2
w_1
w_2
0
1
3
5
8
9
. . .
0
3
4
7
$\mathbf{P} = f_{\boldsymbol{\theta}}(\,,\,)$
$q_{\boldsymbol{\phi}}(y_1|\mathbf{P})$
$q_{\boldsymbol{\phi}}(y_2|y_1, \mathbf{P})$
$q_{\boldsymbol{\phi}}(w_1|\mathbf{y}, \mathbf{P})$
$q_{\boldsymbol{\phi}}(w_2|w_1, \mathbf{y}, \mathbf{P})$

Figure 2: Example rollout when sampling an inference model on input $\mathbf{x} = (\,,\,)$. We first predict $\mathbf{y} = (1, 3)$. Note that for y_2, we prune option 9, as the highest attainable sum is $9 + 9 = 18$. For w_1, $\{0, ..., 3\}$ are pruned as there is no second digit to complete the sum. w_2 is deterministic given w_1, and prunes all branches but 8.

C.3 Symbolic pruner

An attractive option is to use symbolic knowledge to ensure the inference model only generates valid outputs. We can compute each factor $q_{\boldsymbol{\phi}}(w_i|\mathbf{y}, \mathbf{w}_{1:i-1}, \mathbf{P})$ (both for world and output variables) using a neural network $\hat{q}_{\boldsymbol{\phi},i}$ and a *symbolic pruner* s_i:

$$\begin{aligned} q_{\boldsymbol{\phi}_e}(w_i|\mathbf{y}, \mathbf{w}_{1:i-1}, \mathbf{P}) &= \frac{q_{w_i} s_{w_i}}{\mathbf{q} \cdot \mathbf{s}} \\ \mathbf{q} &= \hat{q}_{\boldsymbol{\phi}_e,i}(\mathbf{w}_{1:i-1}, \mathbf{y}, \mathbf{P}) \\ \mathbf{s} &= s_i(\mathbf{w}_{1:i-1}, \mathbf{y}). \end{aligned} \tag{12}$$

The symbolic pruner sets the probability mass of certain branches w_i to zero. Then, $q_{\boldsymbol{\phi}}$ is computed by renormalizing. If we know that by expanding $\mathbf{w}_{1:i-1}$ it will be impossible to produce a possible world for $\mathbf{y}$, we can set the probability mass under that branch to 0: we will know that $p(\mathbf{w}, \mathbf{y}) = 0$ for all such branches. In Figure 2 we give an example for single-digit MNISTAdd, and in Appendix J we present a symbolic pruner that also works for Multi-digit MNISTAdd. Symbolic pruning significantly reduces the number of branches our algorithm needs to explore during training. Moreover, symbolic pruning is appealing in settings where verifiability and safety play crucial roles, such as medicine.

We next discuss two high-level approaches for designing the symbolic pruner:

1. **Use SAT-solvers.** Add the sampled symbols $\mathbf{y}$ and $\mathbf{w}_{1:i}$ to a CNF-formula, and ask a SAT-solver if there is an extension $\mathbf{w}_{i+1:k_W}$ that satisfies the CNF-formula. SAT-solvers are a general approach that will work with every function c, but using them comes at the cost of having to solve a linear amount of NP-hard problems. However, competitive SAT solvers can deal with substantial problems due to years of advances in their design Balyo et al. (2022). Furthermore, a linear amount of NP-hard calls is a lower complexity class than #P hard. However, in the case of MNISTAdd, we can get a perfect pruner in linear time; see Appendix J. Using SAT-solvers will be particularly attractive in problem settings where safety and verifiability are critical.
2. **Prune with local constraints.** In many structured prediction tasks, we can use local constraints of the symbolic problem to prune paths that are guaranteed to lead to branches that can never create possible worlds. However, local constraints do not guarantee that each non-pruned path contains a possible world.

D A-NeSI and GFlowNets

A-NeSI is inspired by the theory of GFlowNets Bengio et al. (2021; 2022), and we use this theory to derive our loss function. In the current section, we discuss these connections and the potential for future research by taking inspiration from the GFlowNet literature. In this section, we will assume

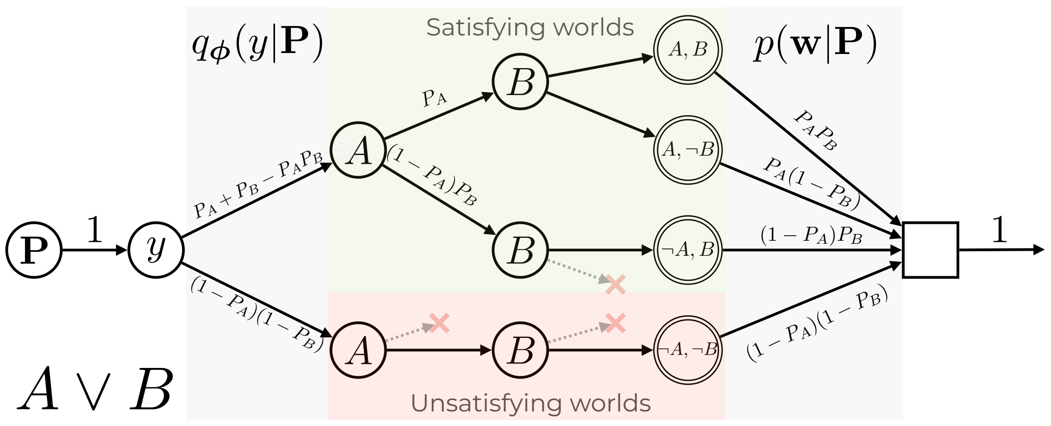


Figure 3: The tree flow network corresponding to weighted model counting on the formula $A \vee B$. Following edges upwards means setting the corresponding binary variable to true (and to false by following edges downwards). We first choose probabilities for the propositions A and B, then choose whether we want to sample a world that satisfies the formula $A \vee B$. $y = 1$ is the WMC of $A \vee B$, and equals its outgoing flow $P_A + P_B - P_A P_B$. Terminal states (with two circles) represent choices of the binary variables A and B. These are connected to a final sink node, corresponding to the prior over worlds $p(\mathbf{w}|\mathbf{P})$. The total ingoing and outgoing flow to this network is 1, as we deal with normalized probability distributions p and q_ϕ.

the reader is familiar with the notation introduced in Bengio et al. (2022) and refer to this paper for the relevant background.

D.1 Tree GFlowNet representation

The main intuition is that we can treat the inference model q_ϕ in Equation 3 as a 'trivial' GFlowNet. We refer to Figure 3 for an intuitive example. It shows what a flow network would look like for the formula $A \vee B$. We take the reward function $R(\mathbf{w}, \mathbf{y}) = p(\mathbf{w}, \mathbf{y})$. We represent states s by $s = (\mathbf{P}, \mathbf{y}_{1:i}, \mathbf{w}_{1:j})$, that is, the belief $\mathbf{P}$, a list of some dimensions of the output instantiated with a value, and a list of some dimensions of the world assigned to some value. Actions a set some value to the next output or world variable, i.e., $A(s) = Y_{i+1}$ or $A(s) = W_{j+1}$.

Note that this corresponds to a flow network that is a tree everywhere but at the sink, since the state representation conditions on the whole trajectory observed so far. We demonstrate this in Figure 3. We assume there is some fixed ordering on the different variables in the world, which we generate the value of one by one. Given this setup, Figure 3 shows that the branch going up from the node y corresponds to the regular weighted model count (WMC) introduced in Equation 1.

The GFlowNet forward distribution P_F is q_ϕ as defined in Equation 3. The backward distribution P_B is $p(\mathbf{w}, \mathbf{y}|\mathbf{P})$ at the sink node, which chooses a terminal node. Then, since we have a tree, this determines the complete trajectory from the terminal node to the source node. Thus, at all other states, the backward distribution is deterministic. Since our reward function $R(\mathbf{w}, \mathbf{y}, \mathbf{P}) = p(\mathbf{w}, \mathbf{y}|\mathbf{P})$ is normalized, we trivially know the partition function $Z(\mathbf{P}) = \sum_{\mathbf{w}} \sum_{\mathbf{y}} R(\mathbf{w}, \mathbf{y}|\mathbf{P}) = 1$.

D.2 Lattice GFlowNet representation

Our setup of the generative process assumes we are generating each variable in the world in some order. This is fine for some problems like MNISTAdd, where we can see the generative process as 'reading left to right'. For other problems, such as Sudoku, the order in which we would like to generate the symbols is less obvious. Would we generate block by block? Row by row? Column by column? Or is the assumption that it needs to be generated in some fixed order flawed by itself?

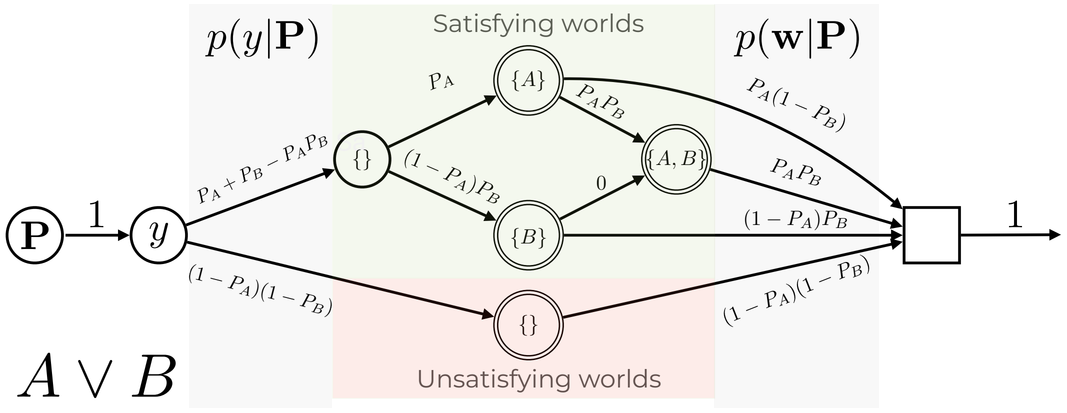


Figure 4: The lattice flow network corresponding to weighted model counting on the formula $A \vee B$. In this representation, nodes represent sets of included propositions. Terminal states represent sets of random variables such that $A \vee B$ is true given $y = 1$, or false otherwise.

In this section, we consider a second GFlowNet representation for the inference model that represents states using sets instead of lists. We again refer to Figure 4 for the resulting flow network of this representation for $A \vee B$. We represent states using $s = (\mathbf{P}, \{y_i\}_{i\in I_Y}, \{w_i\}_{i\in I_W})$, where $I_Y \subseteq \{1, ..., k_Y\}$ and $I_W \subseteq \{1, ..., k_W\}$ denote the set of variables for which a value is chosen. The possible actions from some state correspond to $A(s) = \bigcup_{i\notin I_W} W_i$ (and analogous for when $\mathbf{y}$ is not yet completely generated). For each variable in W for which we do not have the value yet, we add its possible values to the action space.

With this representation, the resulting flow network is no longer a tree but a DAG, as the order in which we generate the different variables is now different for every trajectory. What do we gain from this? When we are primarily dealing with categorical variables, the two gains are 1) we no longer need to impose an ordering on the generative process, and 2) it might be easier to implement parameter sharing in the neural network that predicts the forward distributions, as we only need a single set encoder that can be reused throughout the generative process.

However, the main gain of the set-based approach is when worlds are all (or mostly) binary random variables. We illustrate this in Figure 4. Assume $W = \{0, 1\}^{k_W}$. Then we can have the following state and action representations: $s = (\mathbf{P}, \mathbf{y}, I_W)$, where $I_W \subseteq \{1, ..., k_W\}$ and $A(s) = \{1, ..., k_W\} \setminus I_W$. The intuition is that I_W contains the set of all binary random variables that are set to 1 (i.e., true), and $\{1, ..., k_W\} \setminus I_W$ is the set of variables set to 0 (i.e., false). The resulting flow network represents a *partial order* over the set of all subsets of $\{1, ..., k_W\}$, which is a *lattice*, hence the name of this representation.

With this representation, we can significantly reduce the size and computation of the flow network required to express the WMC problem. As an example, compare Figures 3 and 4, which both represent the WMC of the formula $A \vee B$. We no longer need two nodes in the branch $y = 0$ to represent that we generate A and B to be false, as the initial empty set $\{\}$ already implies they are. This will save us two nodes. Similarly, we can immediately stop generating at $\{A\}$ and $\{B\}$, and no longer need to generate the other variable as false, which also saves a computation step.

While this is theoretically appealing, the three main downsides are 1) P_B is no longer trivial to compute; 2) we have to handle the fact that we no longer have a tree, meaning there is no longer a unique optimal P_F and P_B; and 3) parallelization becomes much trickier. We leave exploring this direction in practice for future work.

E Analyzing the Joint Matching Loss

This section discusses the loss function we use to train the joint variant in Equation 7. We recommend interested readers first read Appendix D.1. Throughout this section, we will refer to $p := p(\mathbf{w}, \mathbf{y}|\mathbf{P})$

(Equation 5) and $q := q_{\boldsymbol{\phi}}(\mathbf{w}, \mathbf{y}|\mathbf{P})$ (Equation 2). We again refer to Bengio et al. (2022) for notational background.

E.1 Trajectory Balance

We derive our loss function from the recently introduced Trajectory Balance loss for GFlowNets, which is proven to approximate the true Markovian flow when minimized. This means sampling from the GFlowNet allows sampling in proportion to reward $R(s_n) = p$. The Trajectory Balance loss is given by

$$\mathcal{L}(\tau) = \left(\log \frac{F(s_0) \prod_{t=1}^{n} P_F(s_t|s_{t-1})}{R(s_n) \prod_{t=1}^{n} P_B(s_{t-1}|s_t)}\right)^2, \tag{13}$$

where s_0 is the source state, in our case $\mathbf{P}$, and s_n is some terminal state that represents a full generation of $\mathbf{y}$ and $\mathbf{w}$. In the tree representation of GFlowNets for inference models (see Appendix D.1), this computation becomes quite simple:

1. $F(s_0) = 1$, as $R(s_n) = p$ is normalized;
2. $\prod_{t=1}^{n} P_F(s_t|s_{t-1}) = q$: The forward distribution corresponds to the inference model $q_{\boldsymbol{\phi}}(\mathbf{w}, \mathbf{y}|\mathbf{P})$;
3. $R(s_n) = p$, as we define the reward to be the true joint probability distribution $p(\mathbf{w}, \mathbf{y}|\mathbf{P})$;
4. $\prod_{t=1}^{n} P_B(s_{t-1}|s_t) = 1$, since the backward distribution is deterministic in a tree.

Therefore, the trajectory balance loss for (tree) inference models is

$$\mathcal{L}(\mathbf{P}, \mathbf{y}, \mathbf{w}) = \left(\log \frac{q}{p}\right)^2 = (\log q - \log p)^2, \tag{14}$$

i.e., the term inside the expectation of the joint matching loss in Equation 7. This loss function is stable because we can sum the individual probabilities in log-space.

A second question might then be how we obtain ‘trajectories’ $\tau = (\mathbf{P}, \mathbf{y}, \mathbf{w})$ to minimize this loss over. The paper on trajectory balance Malkin et al. (2022a) picks τ *on-policy*, that is, it samples τ from the forward distribution (in our case, the inference model $q_{\boldsymbol{\phi}}$). The joint matching loss as defined in Equation 7 is *off-policy*, as we sample from p and not from $q_{\boldsymbol{\phi}}$.

E.2 Relation to common divergences

These questions open quite some design space, as was recently noted when comparing the trajectory balance loss to divergences commonly used in variational inference Malkin et al. (2022b). Redefining $P_F = q$ and $P_B = p$, the authors compare the trajectory balance loss with the KL-divergence and the reverse KL-divergence and prove that

$$\nabla_{\boldsymbol{\phi}} D_{KL}(q||p) = \frac{1}{2}\mathbb{E}_{\tau\sim q}[\nabla_{\boldsymbol{\phi}} \mathcal{L}(\tau)]. \tag{15}$$

That is, the *on*-policy objective minimizes the *reverse* KL-divergence between p and q. We do not quite find such a result for the *off*-policy version we use for the joint matching loss in Equation 7:

$$\nabla_{\boldsymbol{\phi}} D_{KL}(p||q) = -\mathbb{E}_{\tau\sim p}[\nabla_{\boldsymbol{\phi}} \log q] \tag{16}$$

$$\mathbb{E}_{\tau\sim p}[\nabla_{\boldsymbol{\phi}} \mathcal{L}(\tau)] = -2\mathbb{E}_{\tau\sim p}[(\log p - \log q)\nabla_{\boldsymbol{\phi}} \log q] \tag{17}$$

So why do we choose to minimize the joint matching loss rather than the (forward) KL divergence directly? This is because, as is clear from the above equations, it takes into account how far the ‘predicted’ log-probability $\log q$ currently is from $\log p$. That is, given a sample τ, if $\log p < \log q$, the joint matching loss will actually *decrease* $\log q$. Instead, the forward KL will increase the probability for every sample it sees, and whether this particular sample will be too likely under q can only be derived through sampling many trajectories.

Furthermore, we note that the joint matching loss is a ‘pseudo’ f-divergence with $f(t) = t \log^2 t$ Malkin et al. (2022b). It is not a true f-divergence since $t \log^2 t$ is not convex. A related well-known f-divergence is the Hellinger distance given by

$$H^2(p||q) = \frac{1}{2}\mathbb{E}_{\tau\sim p}[(\sqrt{p} - \sqrt{q})^2]. \tag{18}$$

This divergence similarly considers the distance between p and q in its derivatives through squaring. However, it is much less stable than the joint matching loss since both p and q are computed by taking the product over many small numbers. Computing the square root over this will be much less numerically stable than taking the logarithm of each individual probability and summing.

Finally, we note that we minimize the on-policy joint matching $\mathbb{E}_{q_{\boldsymbol{\phi}}}[(\log p - \log q)^2]$ by taking derivatives $\mathbb{E}_{q_{\boldsymbol{\phi}}}[\nabla_{\boldsymbol{\phi}}(\log p - \log q)^2]$. This is technically not minimizing the joint matching, since it ignores the gradient coming from sampling from $q_{\boldsymbol{\phi}}$.

F Dirichlet prior

This section describes how we fit the Dirichlet prior $p(\mathbf{P})$ used to train the inference model. During training, we keep a dataset of the last 2500 observations of $\mathbf{P} = f_{\boldsymbol{\theta}}(\mathbf{x})$. We have to drop observations frequently because $\boldsymbol{\theta}$ changes during training, meaning that the empirical distribution over $\mathbf{P}$s changes as well.

We perform an MLE fit on k_W independent Dirichlet priors to get parameters $\boldsymbol{\alpha}$ for each. The log-likelihood of the Dirichlet distribution cannot be found in closed form Minka (2000). However, since its log-likelihood is convex, we run ADAM Kingma & Ba (2017) for 50 iterations with a learning rate of 0.01 to minimize the negative log-likelihood. We refer to Minka (2000) for details on computing the log-likelihood and alternative options. Since the Dirichlet distribution accepts positive parameters, we apply the softplus function on an unconstrained parameter during training. We initialize all parameters at 0.1.

We added L2 regularization on the parameters. This is needed because at the beginning of training, all observations $\mathbf{P} = f_{\boldsymbol{\theta}}(\mathbf{x})$ represent uniform beliefs over digits, which will all be nearly equal. Therefore, fitting the Dirichlet on the data will give increasingly higher parameter values, as high parameter values represent low-entropy Dirichlet distributions that produce uniform beliefs. When the Dirichlet is low-entropy, the inference models learn to ignore the input belief $\mathbf{P}$, as it never changes. The L2 regularization encourages low parameter values, which correspond to high-entropy Dirichlet distributions.

G MNISTAdd Experimental Setup

Like Manhaeve et al. (2021a;b), we take the MNIST LeCun & Cortes (2010) dataset and use each digit exactly once to create data. We follow Manhaeve et al. (2021a) and require more unique digits for increasing N. Therefore, the training dataset will be of size $60000/2N$ and the test dataset of size $10000/2N$. For the perception model, we use the same CNN as in DeepProbLog Manhaeve et al. (2018).

We chose to use a simple neural network architecture for the inference model. The prediction model has $N+1$ factors $q_{\boldsymbol{\phi}_p}(y_i|\mathbf{y}_{1:i-1}, \mathbf{P})$, while the explanation model has $2N$ factors $q_{\boldsymbol{\phi}_e}(w_i|\mathbf{y}, \mathbf{w}_{1,i-1}, \mathbf{P})$ (see Equation 3). We model each factor with an MLP and have no parameter sharing between the factors. Both y_i and w_i represent digits and are one-hot encoded, except for the first digit of the sum y_1: it can only be 0 or 1.

We performed hyperparameter tuning on a held-out validation set. We refer to Appendix H for the final set of hyperparameters used. We ran each experiment 10 times to estimate average accuracy.

G.1 Baselines

We compare with multiple neurosymbolic frameworks that previously tackled the MNISTAdd task. Several of those are probabilistic neurosymbolic methods: DeepProbLog Manhaeve et al. (2018), DPLA* Manhaeve et al. (2021b), NeurASP Yang et al. (2020) and NeuPSL Pryor et al. (2022). We also compare with the fuzzy logic-based method LTN Badreddine et al. (2022) and with Embed2Sym Aspis et al. (2022) and DeepStochLog Winters et al. (2022). We take results from the corresponding papers, except for DeepProbLog and NeurASP, which are from Manhaeve et al. (2021b), and LTN from Pryor et al. (2022)[1]. We reran Embed2Sym, averaging over 10 runs since its paper did not report standard deviations. We do not compare with DPLA* with pre-training because it tackles an easier problem where part of the digits are labeled.

Embed2Sym Aspis et al. (2022) uses three steps to solve Multi-digit MNISTAdd: First, it trains a neural network to embed each digit and to predict the sum from these embeddings. It then clusters the embeddings and uses symbolic reasoning to assign clusters to labels. A-NESI has a similar neural network architecture, but we train the prediction network on an objective that does not require data. Furthermore, we train A-NESI end-to-end, unlike Embed2Sym.

A-NESI and Embed2Sym have two methods for evaluating the accuracy of predicting the sum:

1. **Symbolic prediction** This uses symbolic reasoning on predicted digit labels to compute the sum. For A-NESI, this corresponds to $\hat{\mathbf{y}} = c(\arg\max_{\mathbf{w}} p(\mathbf{w}|\mathbf{P} = f_{\boldsymbol{\theta}}(\mathbf{x})))$, while for Embed2Sym this corresponds to Embed2Sym-NS.
2. **Neural prediction** This uses a neural network to do prediction. For A-NESI, this is $\hat{\mathbf{y}} = \arg\max_{\mathbf{y}} q_{\boldsymbol{\phi}_p}(\mathbf{y}|\mathbf{P} = f_{\boldsymbol{\theta}}(\mathbf{x}))$, where we use a beam search to find the most likely prediction. For Embed2Sym, this corresponds to Embed2Sym-FN, which also uses a prediction network but is only trained on the training data given and does not use the prior to sample additional data.

For MNISTAdd, we cannot perform better than symbolic prediction since all digits are independent. Therefore, we consider the prediction network adequately trained if it matches the accuracy of symbolic prediction. For other problems such as constrained output prediction, we will want to use neural prediction to ensure predictions are also constrained at test time.

H Hyperparameters

We performed hyperparameter tuning on a held-out validation set by splitting the training data into 50.000 and 10.000 digits, and forming the training and validation sets from these digits. We progressively increased N from $N = 1$, $N = 3$, $N = 4$ to $N = 8$ during tuning to get improved insights into what hyperparameters are important. The most important parameter, next to learning rate, is the weight of the L2 regularization on the Dirichlet prior's parameters which should be very high. We used ADAM Kingma & Ba (2017) throughout. We used Nvidia RTX A4000s GPUs and 24-core AMD EPYC-2 (Rome) 7402P CPUs.

We give the final hyperparameters in Table 2. We use this same set of hyperparameters for all N. # of samples refers to the number of samples we used to train the inference model in Algorithm 1. For simplicity, it is also the beam size for the beam search at test time. The hidden layers and width refer to MLP that computes each factor of the inference model. There is no parameter sharing. The perception model is fixed in this task to ensure performance gains are due to neurosymbolic reasoning (see Manhaeve et al. (2018)).

I Additional experiments

Different variants of A-NESI We perform experiments on four different variants of A-NESI in Table 3.

- *explain*: Refers to the explainable variant that includes the explanation model.
- *pruning*: Refers to the explainable variant with symbolic pruning discussed in Section C.3, which ensures safe predictions. We discuss how to perform linear-time symbolic pruning for the MNISTAdd task in Appendix J.
- *predict*: Refers to the prediction-only variant.
- *no prior*: Refers to the prediction-only variant that is trained *without* the prior $p(\mathbf{P})$. Instead, we only train the prediction model using beliefs $\mathbf{P} = f_{\boldsymbol{\theta}}(\mathbf{x})$.

[1]We take the results of LTN from Pryor et al. (2022) because Badreddine et al. (2022) averages over the 10 best outcomes of 15 runs and overestimates its average accuracy.

Parameter name	Value
Learning rate	0.001
Epochs	100
Batch size	16
# of samples	600
Hidden layers	3
Hidden width	800
Prior learning rate	0.01
Amount beliefs prior	2500
Prior initialization	0.1
Prior iterations	50
L2 on prior	900.000

Table 2: Final hyperparameters

	N=1	N=3	N=6	N=10	N=15
A-NeSI			**Symbolic prediction**		
explain	97.37 ± 0.32	**94.53 ± 0.50**	89.27 ± 1.83	84.06 ± 1.48	**76.84 ± 2.82**
pruning	97.57 ± 0.27	94.33 ± 0.42	90.06 ± 1.05	83.20 ± 1.80	76.46 ± 1.39
predict	**97.66 ± 0.21**	94.18 ± 0.69	**90.24 ± 1.57**	**85.04 ± 1.22**	75.90 ± 2.21
no prior	76.54 ± 27.38	51.80 ± 41.94	13.77 ± 25.31	7.21 ± 19.81	0.03 ± 0.09
A-NeSI			**Neural prediction**		
explain	97.37 ± 0.32	**94.55 ± 0.52**	89.01 ± 1.95	82.68 ± 2.22	**61.77 ± 2.37**
pruning	97.57 ± 0.27	94.35 ± 0.42	89.94 ± 1.21	82.19 ± 1.98	59.88 ± 2.95
predict	**97.66 ± 0.21**	94.20 ± 0.67	**90.00 ± 1.38**	**83.32 ± 1.24**	54.66 ± 1.87
no prior	76.54 ± 27.01	50.33 ± 40.74	9.83 ± 17.86	1.52 ± 4.00	0.00 ± 0.00

Table 3: Accuracy of predicting the correct sum for different variants of A-NESI.

We see there are no significant differences between the first three methods. All perform well, with significant differences only appearing at $N = 15$ where the explainable and pruning variants outperform the predict-only model, especially on neural prediction. We conclude that modellers can freely add the explanation model and the symbolic pruner to enjoy explanations of outputs and safety guarantees on samples without suffering performance penalties.

However, when removing the prior $p(\mathbf{P})$, the performance quickly degrades as N increases. The apparent reason is that the prediction model sees much fewer beliefs $\mathbf{P}$ than when sampling from a (high-entropy) prior $p(\mathbf{P})$. A second and more subtle reason is that at the beginning of training, beliefs $\mathbf{P} = f_{\boldsymbol{\theta}}(\mathbf{x})$ will be mostly uniform because the perception model is not yet learned. Therefore, the observations $\mathbf{P} = f_{\boldsymbol{\theta}}(\mathbf{x})$ for different $\mathbf{x}$ will all be mostly the same, and the prediction model learns to ignore the input belief $\mathbf{P}$.

Inference speed We showed that the approximation of A-NESI maintains strong predictive performance at MNISTAdd. In Figure 5 we compute how long it takes to run inference for a single input $\mathbf{x}$.

We compare A-NESI with DeepProbLog, as we approximate its semantics, and DPLA*, which also approximates DeepProbLog. We see that inference in DeepProbLog increases with a factor 100 as N increases, and DPLA* without pre-training is not far behind. Even with pre-training, which uses labeled MNIST digits, DPLA* inference time grows exponentially. Inference in DeepStochLog, which uses different semantics, is efficient due to highly efficient caching. However, this requires a grounding step that is exponential both in time and memory, and we could not ground beyond $N = 4$ because of memory issues. A-NESI overcomes this exponential growth by not having to perform grounding. In fact, as shown on the right plot, A-NESI scales slightly slower than linear. We note that A-NESI is much faster in practice as it is trivial to parallelize the computation of multiple queries on GPUs.

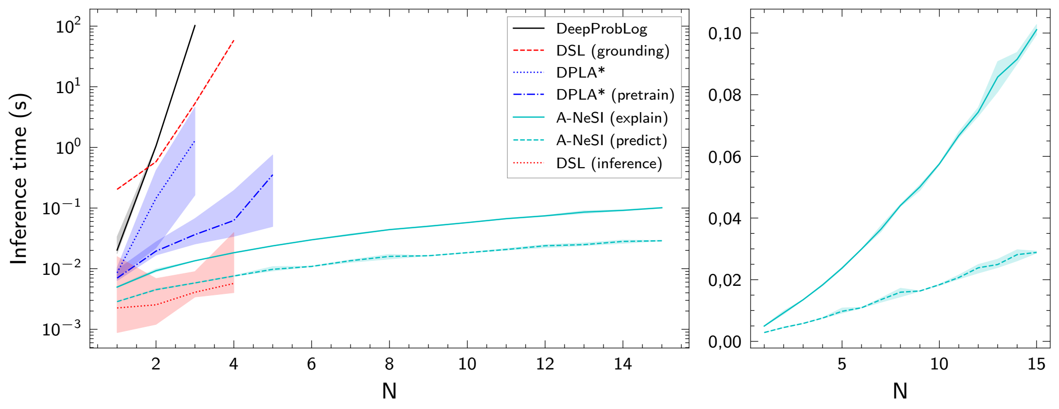


Figure 5: Inference time for a single input $\mathbf{x}$ in seconds for different amounts of digits. The left plot compares with other methods in log scale, and the right plot uses a linear scale. DSL refers to DeepStochLog. These numbers are not to be compared absolutely; instead, the shape of the curve is of primary interest. For DPLA*, we take the GM variant. For DPLA* (pre-train), we report results when pre-training on 128 digits.

J MNISTAdd Symbolic Pruner

In this section, we describe a symbolic pruner for the Multi-digit MNISTAdd problem, which we compute in time linear to N. Note that $\mathbf{w}_{1:N}$ represents the first number and $\mathbf{w}_{N+1:2N}$ the second. We define $n_1 = \sum_{i=1}^N w_i \cdot 10^{N-i-1}$ and $n_2 = \sum_{i=1}^N w_{N+i} \cdot 10^{N-i-1}$ for the integer representations of these numbers, and $y = \sum_{i=1}^{N+1} y_i \cdot 10^{N-i}$ for the sum label encoded by $\mathbf{y}$. We say that partial generation $\mathbf{w}_{1:k}$ has a *completion* if there is a $\mathbf{w}_{k+1:2N} \in \{0, \dots, 9\}^{2N-k}$ such that $n_1 + n_2 = y$.

Proposition J.1. *For all $N \in \mathbb{N}$, $\mathbf{y} \in \{0,1\} \times \{0, \dots, 9\}^N$ and partial generation $\mathbf{w}_{1:k-1} \in \{0, \dots, 9\}^k$ with $k \in \{1, \dots, 2N\}$, the following algorithm rejects all w_k for which $\mathbf{w}_{1:k}$ has no completions, and accepts all w_k for which there are:*

- *$k \leq N$: Let $l_k = \sum_{i=1}^{k+1} y_k \cdot 10^{k+1-i}$ and $p_k = \sum_{i=1}^{k} w_k \cdot 10^{k-i}$. Let $S = 1$ if $k = N$ or if the $(k+1)$th to $(N+1)$th digit of y are all 9, and $S = 0$ otherwise. We compute two boolean conditions for all $w_k \in \{0, \dots, 9\}$:*

$$0 \leq l_k - p_k \leq 10^k - S \tag{19}$$

  We reject all w_k for which either condition does not hold.

- *$k > N$: Let $n_2 = y - n_1$. We reject all $w_k \in \{0, \dots, 9\}$ different from $w_k = \lfloor \frac{n_2}{10^{N-k-1}} \rfloor \bmod 10$, and reject all w_k if $n_2 < 0$ or $n_2 \geq 10^N$.*

Proof. For $k > N$, we note that n_2 is fixed given y and n_1 through linearity of summation, and we only consider $k \leq N$. We define $a_k = \sum_{i=k+2}^{N+1} y_i \cdot 10^{N+1-i}$ as the sum of the remaining digits of y. We note that $y = l_k \cdot 10^{N-k} + a_k$.

Algorithm rejects w_k without completions We first show that our algorithm only rejects w_k for which no completion exists. We start with the constraint $0 \leq l_k - p_k$, and show that whenever this constraint is violated (i.e., $p_k > l_k$), $\mathbf{w}_{1:k}$ has no completion. Consider the smallest possible completion of $\mathbf{w}_{k+1:N}$: setting each to 0. Then $n_1 = p_k \cdot 10^{N-k}$. First, note that

$$10^{N-k} > 10^{N-k} - 1 \geq a_k$$

Next, add $l_k \cdot 10^{N-k}$ to both sides

$$(l_k + 1) \cdot 10^{N-k} > l_k \cdot 10^{N-k} + a_k = y$$

By assumption, p_k is an integer upper bound of l_k and so $p_k \geq l_k + 1$. Therefore,

$$n_1 = p_k \cdot 10^{N-k} > y$$

Since n_1 is to be larger than y, n_2 has to be negative, which is impossible.

Next, we show the necessity of the second constraint. Assume the constraint is unnecessary, that is, $l_k > p_k + 10^k - S$. Consider the largest possible completion $\mathbf{w}_{k+1:N}$ by setting each to 9. Then

$$\begin{aligned} n_1 &= p_k \cdot 10^{N-k} + 10^{N-k} - 1 \\ &= (p_k + 1) \cdot 10^{N-k} - 1 \end{aligned}$$

We take n_2 to be the maximum value, that is, $n_2 = 10^N - 1$. Therefore,

$$n_1 + n_2 = 10^N - (p_k + 1) \cdot 10^{N-k} - 2$$

We show that $n_1 + n_2 < y$. Since we again have an integer upper bound, we know $l_k \geq p_k + 10^k - S + 1$. Therefore,

$$\begin{aligned} y &\geq (p_k + 1 + 10^k - S)10^{N-k} + a_k \\ &\geq n_1 + n_2 + 2 - S \cdot 10^{N-k} + a_k \end{aligned}$$

There are two cases.

- $S = 0$. Then $a_k < 10^{N-k} - 1$, and so
$$y \geq n_1 + n_2 + 2 + a_k > n_1 + n_2.$$
- $S = 1$. Then $a_k = 10^{N-k} - 1$, and so
$$y \geq n_1 + n_2 + 1 > n_1 + n_2.$$

Algorithm accepts w_k with completions Next, we show that our algorithm only accepts w_k with completions. Assume Equation 19 holds, that is, $0 \leq l_k - p_k \leq 10^k - S$. We first consider all possible completions of $\mathbf{w}_{1:k}$. Note that $p_k \cdot 10^{N-k} \leq n_1 \leq p_k \cdot 10^{N-k} + 10^{N-k} - 1$ and $0 \leq n_2 \leq 10^N - 1$, and so

$$p_k \cdot 10^{N-k} \leq n_1 + n_2 \leq (p_k + 1) \cdot 10^{N-k} + 10^N - 2.$$

Similarly,

$$l_k \cdot 10^{N-k} \leq y \leq (l_k + 1) \cdot 10^{N-k} - 1.$$

By assumption, $p_k \leq l_k$, so $p_k \cdot 10^{N-k} \leq l_k \cot 10^{N-k}$. For the upper bound, we again consider two cases. We use the second condition $l_k \leq 10^k + p_k - S$:

- $S = 0$. Then (since there are no trailing 9s),
$$\begin{aligned} y &\leq (l_k + 1) \cdot 10^{N-k} - 2 \\ &\leq (10^k + p_k + 1) \cdot 10^{N-k} - 1 \\ &= (p_k + 1) \cdot 10^{N-k} + 10^N - 2. \end{aligned}$$
- $S = 1$. Then with trailing 9s,
$$\begin{aligned} y &= (l_k + 1) \cdot 10^{N-k} - 1 \\ &\leq (10^k + p_k) \cdot 10^{N-k} - 1 \\ &= p_k \cdot 10^{N-k} + 10^N - 1 \\ &\leq (p_k + 1) \cdot 10^{N-k} + 10^N - 2, \end{aligned}$$
since $10^{N-k} \geq 1$.

Therefore,

$$p_k \cdot 10^{N-k} \leq y \leq (p_k + 1) \cdot 10^{N-k} + 10^N - 2$$

and so there is a valid completion.

$\square$