A Pseudo-Semantic Loss for Deep Generative Models with Logical Constraints

Kareem Ahmed [1] **Kai-Wei Chang** [1] **Guy Van den Broeck** [1]

Abstract

Neuro-symbolic AI bridges the gap between purely symbolic and neural approaches to learning. This often requires maximizing the likelihood of a symbolic constraint w.r.t. the neural network's output distribution. Such output distributions are typically assumed to be fully-factorized. This limits the applicability of neuro-symbolic learning to the more expressive auto-regressive distributions, e.g., transformers. Under such distributions, computing the likelihood of even simple constraints is #P-hard. Instead of attempting to enforce the constraint on the entire output distribution, we propose to do so on a random, local approximation thereof. More precisely, we optimize the likelihood of the constraint under a pseudolikelihood-based approximation centered around a model sample. Our approximation is factorized, allowing the reuse of solutions to subproblems—a main tenet for efficiently computing neuro-symbolic losses. Moreover, it is a local, high-fidelity approximation of the likelihood, exhibiting low entropy and KL-divergence around the model sample. We evaluate our approach on Sudoku and shortest-path prediction cast as auto-regressive generation, and observe that we greatly improve upon the base model's ability to predict logically-consistent outputs. We also evaluate on the task of detoxifying large language models. Using a simple constraint disallowing a list of toxic words, we are able to steer the model's outputs away from toxic generations, achieving SoTA detoxification compared to previous approaches.

1. Introduction

Neuro-symbolic AI aims to consolidate purely statistical approaches, chiefly using neural networks, with purely symbolic approaches for learning and reasoning. It has thus far shown great promise in addressing many of the shortcomings of both paradigms, developing scalable approaches that learn from unstructured data while leveraging domain knowledge to ensure the explainability, trusted behavior as well as reduce the amount of labeled data required by typically data-hungry deep neural networks.

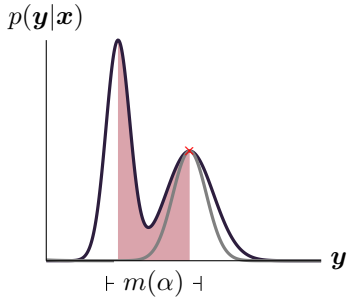

Figure 1. **Our approach in a nutshell**. Given a data point x, we approximate the likelihood of the constraint α (area under the graph shown shaded in pink) with the pseudolikelihood (shown in gray) of the constraint in the neighborhood of a sample (denoted $\times$), where $m(\alpha)$ denotes the region of the constraint support.

More specifically, a common approach to neuro-symbolic learning consists in injecting knowledge regarding the underlying problem domain into the training process as an auxiliary form of supervision. Such knowledge typically takes the form of a sentence in logic, and relates the outputs of the neural network, delineating assignments to the output variables that constitute a valid object from those that do not. For instance, only an assignment to the cells of a Sudoku puzzle such that each row, column, and 3×3 square contain all of the digits from 1 to 9 constitutes a valid Sudoku solution. Injecting such knowledge into training is typically achieved by maximizing the probability of the constraint—the sum of product of probabilities of all the solutions to the constraint—w.r.t. to the network's output distribution. There the outputs of the neural network are assumed to be conditionally independent given the learned features, and therefore the distribution over the solutions of the constraint assumed to be fully-factorized.

In this paper we move beyond fully-factorized output distributions and towards auto-regressive ones, including those induced by large language models such as GPT (Radford

[1]Department of Computer Science, University of California, Los Angeles, USA. Correspondence to: Kareem Ahmed <ahmedk@cs.ucla.edu>.

Accepted at the Knowledge and Logical Reasoning in the Era of Data-driven Learning Workshop at the 40^{th} International Conference on Machine Learning Honolulu, Hawaii, USA. 2023.

et al., 2019), where the output at any given time step depends on the outputs at all previous time steps. Computing the probability of an arbitrary constraint under fully-factorized output distributions is #P-hard. Intuitively, the hardness of the problem can be attributed to the possibly exponentially-many solutions of the constraint. Under an auto-regressive distribution, however, computing the probability of even a single literal as a constraint is #P-hard (Roth, 1993). That is, under auto-regressive distributions, the hardness of computing the probability of an arbitrary constraint is now due to two distinct factors: the hardness of the logical constraint as well as the hardness of the distribution. Throughout this paper, we will assume the inherent hardness of the constraint can be sidestepped: for many applications, we can come up with compact representations of the constraint's solutions that are amenable to computing its probability under the fully-factorized distribution efficiently (cf. Section 3.1). When such compact representations are unavailable, we fall back to approximate representations (Ahmed et al., 2023a).

Our contribution lies in proposing what is, to the best of our knowledge, the first approach to learning with constraints under auto-regressive generative models. Concretely, we approximate the likelihood of the constraint w.r.t. the auto-regressive distribution with its probability in a local pseudolikelihood distribution—a product of conditionals—centered around a model sample. This leads to a factorizable objective which allows us to efficiently compute the probability of constraints by reusing solutions to common sub-problems. Experiments show our approximation is low-entropy, allocating most of its mass around the sample, and has low KL- divergence from the true distribution. Intuitively, we want to stay close to the sample to ensure high fidelity, while retaining a distribution to ensure differentiability and maximum generality within tractability bounds. Our approach is depicted in Figure 1.

Empirically, we start by evaluating our approach on the tasks of solving a Sudoku puzzle and generating a shortest path in a given Warcraft map where, conditioned on the input puzzle (map, resp.), the neural network auto-regressively generates a Sudoku solution (shortest path, resp.), taking into account generations at previous time steps. We observe that our auto-regressive models improve upon the non-auto-regressive baselines, and that our approach leads to models whose predictions are even more accurate, and even more likely to satisfy the constraint. Lastly, we evaluated our approach on the challenging task of detoxifying pretrained large language models where the aim is to move the model's distribution away from toxic generations and towards non-toxic ones without sacrificing the model's overall language modeling abilities. We show that, perhaps surprisingly, using only a simple constraint disallowing a list of toxic words, the model exhibits a great reduction in the toxicity of the generated sentences, as measured using the perspective API [1], at almost no cost in terms of the model's language modeling capabilities, measured in perplexity; this is, to the best of our knowledge, the first use of logical constraints in such a task. Our code will be made publicly available.

Contribution In summary, we propose approximating the likelihood of the constraint w.r.t. the model parameters with the pseudolikelihood of the constraint centered around a model sample. Our approach can be thought of as penalizing the neural network for all the probability mass it allocates to the local perturbations of a model sample that volate the logical constraint. We empirically demonstrate that our approach leads to models whose predictions are more consistent with the constraint on the tasks of Sudoku and Warcraft shortest path generation, and to less toxic generations on the task of large language models detoxification prompted with the RealToxicityPrompts dataset.

2. Background

We first introduce needed background on propositional logic and how neural networks induce distributions over output structures. Afterwards, we motivate and define our loss.

2.1. Notation

We write uppercase letters (X, Y) for Boolean variables and lowercase letters (x, y) for their instantiation ($Y = 0$ or $Y = 1$). Sets of variables are written in bold uppercase ($\mathbf{X}$, $\mathbf{Y}$), and their joint instantiation in bold lowercase ($\boldsymbol{x}$, $\boldsymbol{y}$). A literal is a variable (Y) or its negation ($\neg Y$). A logical sentence (α or β) is constructed from variables and logical connectives ($\wedge$, $\vee$, etc.), and is also called a (logical) formula or constraint. A state or world $\boldsymbol{y}$ is an instantiation to all variables $\mathbf{Y}$. A state $\boldsymbol{y}$ satisfies a sentence α, denoted $\boldsymbol{y} \models \alpha$, if the sentence evaluates to true in that world. A state $\boldsymbol{y}$ that satisfies a sentence α is also said to be a model of α. We denote by $m(\alpha)$ the set of all models of α. The notation for states $\boldsymbol{y}$ is used to refer to an assignment, the logical sentence enforcing the assignment, or the binary output vector capturing the assignment, as these are all equivalent notions. A sentence α entails another sentence β, denoted $\alpha \models \beta$, if all worlds that satisfy α also satisfy β.

2.2. A Probability Distribution over Possible Structures

Let α be a logical sentence defined over Boolean variables $\mathbf{Y} = \{Y_{11}, \ldots, Y_{nk}\}$, where n denotes the number of time steps in the sequence, and k denotes the number of possible classes at each step.

The neural network's outputs induce a probability distribution $p(\cdot)$ over possible states $\boldsymbol{y}$. However, the neural network will ensure that, for each time step i, there is ex-

[1] https://www.perspectiveapi.com/

actly one class being predicted in each possible state. That is, exactly one Boolean variable $\{Y_{i1}, \ldots, Y_{ik}\}$ can be set to true for each time step i. We will use $\boldsymbol{y}_i$ to denote that variable Y_{ij} set to true in state $\boldsymbol{y}$. More precisely, we let $\boldsymbol{y}_i \in \{0,1\}^k$ be the one-hot encoding of Y_{ij} being set to 1 among $\{Y_{i1}, \ldots, Y_{ik}\}$. The probability assigned by the auto-regressive neural network to a state $\boldsymbol{y}$ is then

$$p(\boldsymbol{y}) = \prod_{i=1}^{n} p(\boldsymbol{y}_i \mid \boldsymbol{y}_{<i}), \tag{1}$$

where $\boldsymbol{y}_{<i}$ denotes the prefix $\boldsymbol{y}_1, \ldots, \boldsymbol{y}_{i-1}$. The most common approaches (Mullenbach et al., 2018; Xu et al., 2018; Giunchiglia and Lukasiewicz, 2020) to neuro-symbolic learning assume the conditional independence of the network outputs given the learned embeddings. More precisely, let f be a neural network that maps inputs $\boldsymbol{x}$ to M-dimensional embeddings $\boldsymbol{z} = f(\boldsymbol{x})$. Under such assumption, we obtain the *fully-factorized* distribution

$$p(\boldsymbol{y} \mid \boldsymbol{z}) = \prod_{i=1}^{n} p(\boldsymbol{y}_i \mid \boldsymbol{z}). \tag{2}$$

We no longer have a notion of ordering under the fully-factorized distribution—and each possible $p(\boldsymbol{y}_i = 1 \mid \boldsymbol{z})$ is computed as $\sigma(\mathbf{w}_{ij}^\top \boldsymbol{z})$ where $\mathbf{w}_i \in \mathbb{R}^M$ is a vector of parameters and $\sigma(x)$ is the softmax function. The appeal of such distribution is that it enables the tractability of many reasoning tasks, but the downside is that it dismisses any correlation between the output labels. As we will show in our experimental section (cf. Section 5), using auto-regressive distributions, even simple ones such as LSTMs, already outperforms a neural network where the labels are assumed

2.3. Neuro-Symbolic Losses

In neuro-symbolic learning, we often assume access to symbolic knowledge connecting the different outputs of a neural network, typically in the form of a constraint (or sentence) α in Boolean logic. We are concerned with maximizing the likelihood of the constraint α w.r.t. network's parameters $\boldsymbol{\theta}$:

$$\operatorname*{argmax}_{\boldsymbol{\theta}} p_{\boldsymbol{\theta}}(\alpha) = \operatorname*{argmax}_{\boldsymbol{\theta}} \mathbb{E}_{\boldsymbol{y} \sim p_{\boldsymbol{\theta}}} \left[\mathbb{1}\{\boldsymbol{y} \models \alpha\}\right] \tag{3}$$

$$= \operatorname*{argmax}_{\boldsymbol{\theta}} \sum_{\boldsymbol{y} \models \alpha} p_{\boldsymbol{\theta}}(\boldsymbol{y}), \tag{4}$$

where, with a slight abuse of notation, we omit the inputs $\boldsymbol{x}$. The expectation in Equation (3) quantifies how close the neural network comes to satisfying the constraint. It does so by reducing the problem of probability computation to weighted model counting (WMC): summing up the models of α, each weighted by its likelihood under p. The negative logarithm of this expectation yields a loss function called semantic loss (Xu et al., 2018). Depending on how one chooses to compute said expectation, we recover different approaches. T-norms (Medina Grespan et al., 2021) make various assumptions, for instance, that the clauses of the constraint are independent (Rocktäschel et al., 2015). Ahmed et al. (2022a) estimate the expectation by sampling, using various gradient estimators for learning. Xu et al. (2018) compute the objective exactly, leveraging knowledge compilation techniques that exploit the structure embedded in the solution space. They obtain a target representation (a circuit) in which computing the expectation in Equation (3) using dynamic programming is linear in the size of the target representation (the number of circuit edges). We note that computing this expectation is, for arbitrary constraints, #P-hard (Valiant, 1979a;b). Indeed the size of the compiled circuit can grow exponentially in the constraint. In practice, we can obtain compact circuits for many constraints of interest, or effectively decompose the constraints as an approximation (Ahmed et al., 2023a).

3. Pseudo-Semantic Loss

Unfortunately, as previously mentioned, moving beyond the fully-factorized distribution, we are faced with another source of intractability: the hardness of the distribution w.r.t. which the expectation in Equation (3) is being computed. Assuming a deep generative model whose distribution p can capture a Bayesian network distribution, the problem of computing even a single marginal—i.e., the marginal probability of a single variable—is known to be #P-hard (Roth, 1993). This class of models includes the auto-regressive distribution. Intuitively, a constraint might have exponentially-many solutions, yet lend itself nicely to reusing of solutions to sub-problems, and therefore a tractable calculation of the expectation in Equation (3). An example being the n choose k constraint (Ahmed et al., 2023b), where the expectation in Equation (3) can be computed in quadratic time under the fully-factorized distribution, despite having a normally-prohibitive number of solutions. Moving away from the fully-factorized distribution, however, entails that in the worst case, we would need to compute a sub-problem combinatorial number of times—for all possible sequences—for the exponentially many solutions of the constraint.

To sidestep the intractability of the expectation in Equation (3), as a first step, we consider the *pseudolikelihood* $\tilde{p}(\cdot)$ of a set of parameters given an assignment (Besag, 1975), as a surrogate for its likelihood i.e.,

$$p(\boldsymbol{y}) \approx \tilde{p}(\boldsymbol{y}) \coloneqq \prod_{i} p(\boldsymbol{y}_i \mid \boldsymbol{y}_{-i}), \tag{5}$$

where $\boldsymbol{y}_{-i}$ denotes $\boldsymbol{y}_1, \ldots, \boldsymbol{y}_{i-1}, \boldsymbol{y}_{i+1}, \ldots, \boldsymbol{y}_n$. Consequently, we can consider *the pseudolikelihood* of a set of parameters given a logical constraint α as a surrogate for its

true likelihood i.e.,

$$p(\alpha) \approx \tilde{p}(\alpha) = \mathbb{E}_{\boldsymbol{y}\sim\tilde{p}}\left[\mathbb{1}\{\boldsymbol{y} \models \alpha\}\right] = \sum_{\boldsymbol{y}\models\alpha} \tilde{p}(\boldsymbol{y}). \tag{6}$$

Intuitively, the pseudolikelihood objective aims to measure our ability to predict the value of each variable given a full observation of all other variables. The pseudolikelihood objective is attempting to match all of the model's conditional distributions to the conditional distributions computed from the data. If it succeeds in matching them exactly, then a Gibbs sampler run on the model's conditional distributions attains the same invariant distribution as a Gibbs sampler run on the true data distribution.

On its own, the above would still not be sufficient to ensure the tractability of the expectation in Equation (3). Intuitively, different solutions depend on different sets of conditionals, meaning we would have to compute the probabilities of many of the solutions of the constraint from scratch.

Instead, *we compute the pseudolikelihood of the constraint in the neighborhood of a model sample*[2]

$$\begin{aligned}\tilde{p}(\alpha) &= \mathbb{E}_{\boldsymbol{y}\sim\tilde{p}}\left[\mathbb{1}\{\boldsymbol{y} \models \alpha\}\right] &(7)\\ &\approx \mathbb{E}_{\boldsymbol{y}\sim p}\,\mathbb{E}_{\tilde{\boldsymbol{y}}\sim\tilde{p}_{\boldsymbol{y}}}\left[\mathbb{1}\{\tilde{\boldsymbol{y}} \models \alpha\}\right] &(8)\\ &= \mathbb{E}_{\boldsymbol{y}\sim p}\,\tilde{p}_{\boldsymbol{y}}(\alpha) = \mathbb{E}_{\boldsymbol{y}\sim p}\sum_{\tilde{\boldsymbol{y}}\models\alpha}\tilde{p}_{\boldsymbol{y}}(\tilde{\boldsymbol{y}}), &(9)\end{aligned}$$

where

$$\tilde{p}_{\boldsymbol{y}}(\tilde{\boldsymbol{y}}) \coloneqq \prod_i p(\tilde{\boldsymbol{y}}_i \mid \boldsymbol{y}_{-i}) \tag{10}$$

which can be seen as the pseudolikelihood $\tilde{p}(\cdot)$ of an assignment in the neighborhood of a sample $\boldsymbol{y}$. Crucially this distribution is fully factorized, making it amenable to the efficient computation of neuro-symbolic loss functions.

Definition 3.1 (Pseudo-Semantic Loss)**.** Let α be a sentence in Boolean logic, and let $\tilde{p}_{\boldsymbol{y}}(\cdot)$ be the pseudolikelihood function parameterized by $\boldsymbol{\theta}$ and centered around state $\boldsymbol{y}$, as defined in Equation (10). Then, we define the pseudo-semantic loss between α and $\boldsymbol{\theta}$ to be

$$\begin{aligned}\mathcal{L}^{\mathsf{SL}}_{\mathsf{pseudo}}(\alpha, p_{\boldsymbol{\theta}}) &\coloneqq -\log \mathbb{E}_{\boldsymbol{y}\sim p}\,\tilde{p}_{\boldsymbol{y}}(\alpha) &(11)\\ &= -\log \mathbb{E}_{\boldsymbol{y}\sim p}\sum_{\tilde{\boldsymbol{y}}\models\alpha}\tilde{p}_{\boldsymbol{y}}(\tilde{\boldsymbol{y}}). &(12)\end{aligned}$$

Intuitively, our pseudo-semantic loss between α and $p_{\boldsymbol{\theta}}$ can be thought of as penalizing the neural network for all probability mass it allocates to the local perturbations $\tilde{\boldsymbol{y}}$ of the model sample $\boldsymbol{y}$ that violate the logical constraint α.

[2]We sample y_1 conditioned on the beginning-of-sentence token, then y_2 conditioned on the sampled y_1, followed by y_3 conditioned on both y_1 and y_2 and so on until the end-of-sentence token.

3.1. Tractable Expectation Computations

We appeal to knowledge compilation techniques—a class of methods that transform, or *compile*, a logical theory into a *tractable circuit* target form, which represent functions as parameterized computational graphs. By imposing certain structural properties on them, we enable the tractable computation of certain classes of probabilistic queries over the encoded functions. Circuits then provide a language for building and reasoning about tractable representations.

Logical Circuits More formally, a *logical circuit* is a directed, acyclic computational graph representing a logical formula. Each node n in the DAG encodes a logical subformula, denoted $[n]$. Each inner node in the graph is either an AND or an OR gate, and each leaf node encodes a Boolean literal (Y or $\neg Y$). We denote by $\mathsf{in}(n)$ the set of n's children, that is, the operands of its logical gate.

Structural Properties Circuits enable the tractable computation of certain classes of queries over encoded functions granted that a set of structural properties are enforced.

A circuit is *decomposable* if the inputs of every AND gate depend on disjoint sets of variables i.e. for $\alpha = \beta \wedge \gamma$, $\mathsf{vars}(\beta) \cap \mathsf{vars}(\gamma) = \varnothing$. Intuitively, decomposable AND nodes encode local factorizations over variables of the function. For simplicity, we assume that decomposable AND gates always have two inputs, a condition enforceable on any circuit in exchange for a polynomial increase in size.

A second useful property is *smoothness*. A circuit is *smooth* if the children of every OR gate depend on the same set of variables i.e. for $\alpha = \bigvee_i \beta_i$, we have that $\mathsf{vars}(\beta_i) = \mathsf{vars}(\beta_j)\ \forall i, j$. Decomposability and smoothness are a sufficient and necessary condition for tractable integration over arbitrary sets of variables in a single pass, as they allow larger integrals to decompose into smaller ones (Choi et al., 2020).

Furthermore, a circuit is said to be *deterministic* if, for any input, at most one child of every OR node has a non-zero output i.e. for $\alpha = \bigvee_i \beta_i$, we have that $\beta_i \wedge \beta_j = \bot$ for all $i \neq j$. Similar to decomposability, determinism induces a recursive partitioning of the function, but over the support, i.e. satisfying assignments, of the function, rather than the variables. Determinism, taken together with smoothness and decomposability, allows us to tractably compute a constraint probability (Darwiche and Marquis, 2002). Given a smooth, deterministic and decomposable logical circuit c_α encoding a constraint α[3] we can compute the probability $p(\alpha)$ w.r.t. a distribution p that factorizes by feeding the probability of each literal at the corresponding leaf node and evaluating the circuit upwards, taking sums at OR nodes and products

[3]Such a circuit can always be constructed, (Appendix A) although it can grow exponentially in the worst case.

at AND nodes. Figure 2 shows an example of computing the probability of such a circuit.

Algorithm 1 $\mathcal{L}^{\mathsf{SL}}_{\mathsf{pseudo}}(\alpha; p_{\boldsymbol{\theta}})$

1: **Input**: Logical constraint α and model $p_{\boldsymbol{\theta}}$.
2: **Output**: Pseudo-semantic loss of α w.r.t. $\boldsymbol{\theta}$
3: // Obtain sample y from $p_{\boldsymbol{\theta}}$
4: $\boldsymbol{y} \sim p_{\boldsymbol{\theta}}$
5: // Get sequence length and num. of categories
6: seq, cats = $\boldsymbol{y}$.shape()
7: // Expand the batch to contain all perturbations
8: // of $\boldsymbol{y}$ that are a Hamming distance of 1 away
9: $\boldsymbol{y} = \boldsymbol{y}$.expand(seq_len, num_cat)
10: $\boldsymbol{y}$[:, range(seq), :, range(seq)] = range(cats)
11: // Evaluate expanded samples through model
12: $\log p_{\boldsymbol{\theta}} = p_{\boldsymbol{\theta}}(\boldsymbol{y})$.log_softmax(dim= -1)
13: // Compute the conditional probabilities:
14: // $\log \tilde{p}_{\boldsymbol{\theta}}[i][j] = p_{\boldsymbol{\theta}}(\boldsymbol{y}_j | \boldsymbol{y}_{-j})$
15: $\log \tilde{p}_{\boldsymbol{\theta}} = \log p_{\boldsymbol{\theta}} - \log p_{\boldsymbol{\theta}}$.logsumexp(dim=$-1$)
16: // Compute the probability of α under $\tilde{p}_{\boldsymbol{y}}$
17: // by propagating the conditionals through c_α
18: **return** $-\log \tilde{p}_{\boldsymbol{y}}(\alpha)$

3.2. The Algorithm

We will now give a walk through of computing our pseudo-semantic loss. We note that our algorithm is implemented in log-space to preserve numerical stability and uses PyTorch (Paszke et al., 2019). Our full algorithm is shown in Algorithm 1. We sample an assignment $\boldsymbol{y} \sim p_{\boldsymbol{\theta}}$ from the model (line 4). We compute the sample pseudolikelihood

$$\log \tilde{p}_\theta(\boldsymbol{y}) = \sum_i \log p(\boldsymbol{y}_i \mid \boldsymbol{y}_{-i})$$
$$= \sum_i \log p(\boldsymbol{y}_i, \boldsymbol{y}_{-i}) - \operatorname*{LSE}_{\boldsymbol{y}'_i} \log p(\boldsymbol{y}'_i, \boldsymbol{y}_{-i}),$$

where LSE is the logsumexp function. That is, for every element in the sequence, we need to marginalize over all categories $\boldsymbol{y}'_i$. This entails, for every element in the sampled sequence, we need to substitute each of the categories (lines 9-10) and compute the probability of the sample under the model (line 12), obtaining sequence length $\times$ number of categories sequences. Now we can compute the log-conditional probabilities $\log p(\boldsymbol{y}_i \mid \boldsymbol{y}_{-i})$. We marginalize over the categories $\boldsymbol{y}'_i$ to obtain the log-marginal $\log p(\boldsymbol{y}_{-i})$ = LSE $\boldsymbol{y}'_i(\log p(\boldsymbol{y}'_i, \boldsymbol{y}_{-i}))$. We then condition the probability of every sequence by subtracting the log-marginals i.e., $\log p(\boldsymbol{y}_i, \boldsymbol{y}_{-i}) - \log p(\boldsymbol{y}_{-i})$ (line 15). We use these conditionals to compute the pseudolikelihood assigned by the neural network to local perturbations of the model sample y that satisfy the constraint (line 18). As per Section 3.1, we can compute the pseudolikelihood of a constraint α locally around the sample y by pushing the computed conditionals at the respective input nodes of c_α, propagating up through the circuit, reading the value at the circuit root. Figure 2 shows a toy example run of our algorithm in non-log space.

4. Related Work

In an acknowledgment to the need for both symbolic as well as sub-symbolic reasoning, there has been a plethora of recent works studying how to best combine neural networks and logical reasoning, dubbed *neuro-symbolic reasoning*. The focus of such approaches is typically making probabilistic reasoning tractable through first-order approximations, and differentiable, through reducing logical formulas into arithmetic objectives, replacing logical operators with their fuzzy t-norms, and implications with inequalities (Kimmig et al., 2012; Rocktäschel et al., 2015; Fischer et al., 2019).

Another class of neuro-symbolic approaches have their roots in logic programming. DeepProbLog (Manhaeve et al., 2018) extends ProbLog, a probabilistic logic programming language, with the capacity to process neural predicates, whereby the network's outputs are construed as the probabilities of the corresponding predicates. This simple idea retains all essential components of ProbLog: the semantics, inference mechanism, and the implementation. In a similar vein, Dai et al. (2018) combine domain knowledge specified as purely logical Prolog rules with the output of neural networks, dealing with the network's uncertainty through revising the hypothesis by iteratively replacing the output of the neural network with anonymous variables until a consistent hypothesis can be formed. Bošnjak et al. (2017) present a framework combining prior procedural knowledge, as a Forth program, with neural functions learned through data. The resulting neural programs are consistent with specified prior knowledge and optimized with respect to data.

Diligenti et al. (2017) and Donadello et al. (2017) use first-order logic to specify constraints on outputs of a neural network. They employ fuzzy logic to reduce logical formulas into differential, arithmetic objectives denoting the extent to which neural network outputs violate the constraints, thereby supporting end-to-end learning under constraints. Xu et al. (2018) introduced semantic loss, which circumvents the shortcomings of fuzzy approaches, while still supporting end-to-end learning under constraints. More precisely, *fuzzy reasoning* is replaced with *exact probabilistic reasoning*, made possible by compiling logical formulae into structures supporting efficient probabilistic queries.

Lastly, there has recently been a plethora of approaches ensuring consistency by embedding the constraints as predictive layers, including semantic probabilistic layers (SPLs) (Ahmed et al., 2022b), MultiplexNet (Hoernle et al., 2022)

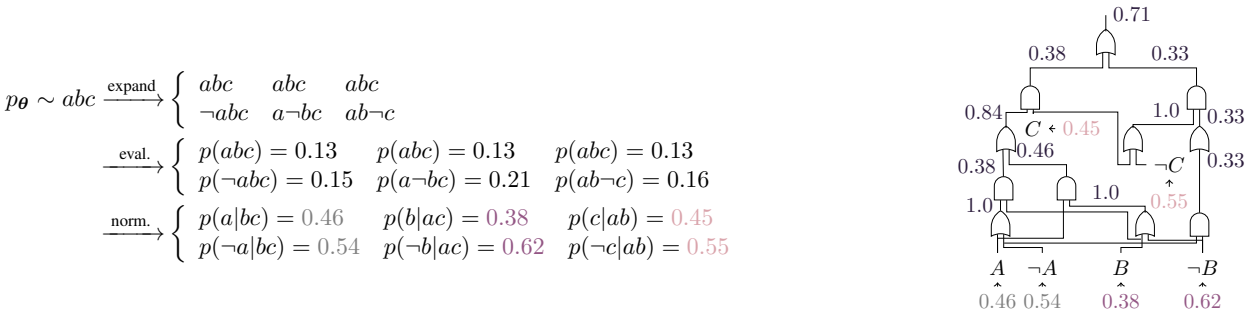

Figure 2. **An example of our pipeline.** (Left) We start by sampling an assignment from the model $p_{\boldsymbol{\theta}}$. Our goal is to compute the pseudolikelihood of the model sample—the product of the sample's conditionals. We start by expanding the model sample to include all samples that are a Hamming distance of 1 away from the sample. We proceed by (batch) evaluating the samples through the model, obtaining the joint probability of each sample. We then normalize along each column, obtaining the conditionals. (Right) A logical circuit encoding constraint (Cat $\implies$ Animal) $\wedge$ (Dog $\implies$ Animal). To compute the pseudolikelihood of the constraint in the neighborhood of the sample abc, we feed the computed conditional at the corresponding literals. We push the probabilities forward, taking products at AND nodes and sums at OR nodes. The number accumulated at the root of the circuit is the pseudolikelihood of the constraint in the

and HMCCN (Giunchiglia and Lukasiewicz, 2020). Much like semantic loss (Xu et al., 2018), SPLs maintain sound probabilistic semantics, and while displaying impressive scalability to real world problems, but might struggle with encoding harder constraints. MultiplexNet is able to encode only constraints in disjunctive normal form, which is problematic for generality and efficiency as neuro-symbolic tasks often involve an intractably large number of clauses. HMCCN encodes label dependencies as fuzzy relaxation and is the current state-of-the-art model for hierarchical multi-label classification (Giunchiglia and Lukasiewicz, 2020), but, similar to its recent extension (Giunchiglia and Lukasiewicz, 2021), is restricted to a certain family of constraints.

Throughout this paper, we assumed that the constructing a logical circuit from a logical formula was easy. This is, in general, not the case. Ahmed et al. (2023a) offer an approach, by assuming the sub-problems are independent, and iteratively relaxing the independence assumption according to the sub-problems that most violate that assumption as measured using the conditional mutual information.

5. Experimental Evaluation

We evaluate our pseudo-semantic loss on several tasks, spanning a number of domains. We start by evaluating on Warcraft shortest-path finding, where we are given an image of a Warcraft tilemap, and are tasked with *auto-regressively* generating one of the potentially many minimum-cost paths between two end points conditioned on the map, where the cost is determined by the *underlying* cost of the tiles spanned by the path. We move on to evaluating on the classic, yet challenging, task of solving a 9×9 Sudoku puzzle where, once again, the generation proceeds auto-regressively, conditioned on the input Sudoku puzzle. It is worth noting that such tasks have been considered as a test bed for other neuro-symbolic approaches before, but never from an auto-regressive generation perspective.

We also evaluate on the task of large language models (LLMs) detoxification. In this task, we are interested in the generations produced by an LLM when presented by a prompt input by the user. More specifically, we are interested not only in how good these models are at the modeling aspect, but also how *toxic* their outputs might be, a measure which includes sexual explicitness, identity attacks, and profanity, among others. Our goal in this task is then to shift the model's distribution away from toxic generations, and toward non-toxic ones, all while maintaining its original ability to model text. We believe this to be a timely and important problem due to their recent prevalence and widespread usage coupled with the fact that previous work (Gehman et al., 2020) has found non-negligible amounts of toxic, harmful, and abusive text in the corpora used to train LLMs.

Lastly, we evaluated our approximation's fidelity by comparing the entropy of our local approximation against that of the GPT-2 distribution, as well as how close our approximation is to the true likelihood in the proximity of the sampled data point as measured by the KL-divergence between the two. All experimental details, hardware specifications, as well as training details are provided in the appendix.

Warcraft Shortest Path For this task, we follow the experimental setting set forth by (Pogančić et al., 2020), where our training set consists of $10,000$ terrain maps curated using Warcraft II tileset. Each map encodes a 12×12 grid superimposed on a Warcraft terrain map, where each vertex is weighted according to the cost of the tile, which in turn depends on type of terrain it represents e.g., earth has lower cost than water. These costs are *not* presented to the

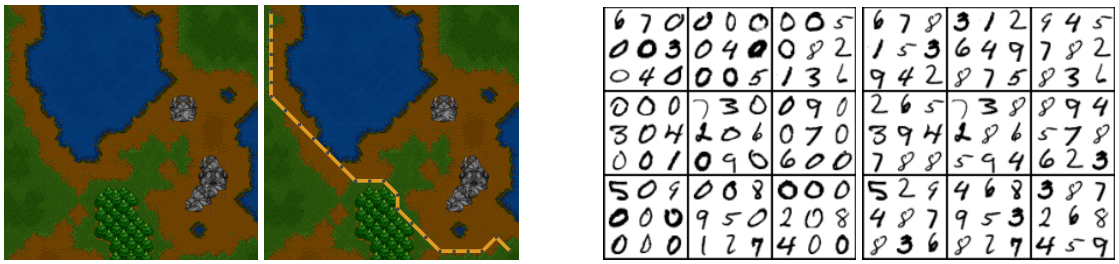

Figure 3. **Example inputs and groundtruth labels for two of the three tasks considered in our experimental evaluation.** (Left) Example Warcraft terrain map and a possible minimum-cost shortest path. (Right) Example Sudoku puzzle and its corresponding solution.

network. The task is then to generate a minimum-cost path from the upper left to the lower right vertices, where the cost of a path is defined as the sum of costs of the vertices visted by the edges along the path, and the minimum-cost path is not unique, i.e., there exists many paths with the minimum cost, and are all considered correct. The minimum cost path between the top left and bottom right vertices is encoded as an indicator matrix, and serves as a label. Figure 3 shows an example input to the network, and a possible path.

We use a CNN-LSTM model, where, presented with an image of a terrain map, we use a ResNet18 (He et al., 2016) to obtain a 128 image embedding, which is then passed on to an LSTM with a single layer, a hidden dim of size 512, and at every time step predicts the next edge in the path conditioned on the image embedding and previous edges. The constraint being maximized by pseudo-semantic loss in this task is that the predicted edges form a valid path.

As previously established (Xu et al., 2018; Ahmed et al., 2022c;b), the accuracy of predicting individual labels is often a poor indicator of the performance in neuro-symbolic settings, where we are rather more interested in the accuracy of our predicted structure object *exactly* matching the groundtruth label , e.g., *is the prediction a shortest path?*, a metric which we denote "Exact" in our experiments, as well as the accuracy of predicting objects that are *consistent* with the constraint, e.g., *is the prediction a valid path?*, a metric denoted "Consistent". Our results are shown in Table 2.

As alluded to repeatedly throughout the course of the paper, the first observation is that using an auto-regressive model to predict the shortest path in the grid, even a simple single layer LSTM outperforms both a ResNet-18, as well as a ResNet-18 trained with semantic loss, improving the exact match from 55.00% and 59.40% to 62.00%, and greatly improving the consistency of the predicted paths to 76.00%, an improvement by almost 15%. We also see using our pseudo-semantic loss, denoted PSEUDOSL, improves the exact and consistent accuracies to 66.00% and 79.00%, respectively.

Sudoku Next, we consider the task of predicting a solution to a given Sudoku puzzle. Here the task is, given a 9×9 partially-filled grid of numbers to fill in the remaining cells such that the entries each row, column, and 3×3 square are unique i.e., each number from 1 to 9 appears exactly once.

Table 1. Our experimental results on Sudoku.

Test accuracy %	Exact	Consistent
ConvNet	16.80	16.80
ConvNet + SL	22.10	22.10
RNN	22.40	22.40
RNN + PSEUDOSL	**28.20**	**28.20**

Table 2. Our experimental results on Warcraft.

Test accuracy %	Exact	Consistent
ResNet-18	55.00	56.90
ResNet-18 + SL	59.40	61.20
CNN-LSTM	62.00	76.60
CNN-LSTM + PSEUDOSL	**66.00**	**79.00**

We use the dataset provided by Wang et al. (2019), consisting of 10K Sudoku puzzles, split into 9K training examples, and 1K test samples, all puzzles having 10 missing entries.

As our baseline, we use a 5-layer RNN with a hidden dimension of 128, $\tanh$ non-linearity and a dropout of 0.2. At each time step, the RNN predicts the next cell given as input a one-hot encoding of the previous cell, and conditioned on the partially filled Sudoku. The constraint being maximized here is that entries in each row, column, and 3×3 squares are unique, each containing each number from 1 to 9 appears exactly once. Our results are shown in Table 1. In line with our previous experiment, we observe that, one again, a simple RNN outperforms the non-auto-regressive model, as well as the same model augmented with semantic loss, although the difference is not that big with regards to semantic loss. Augmenting that same auto-regressive model with pseudo-semantic loss, however, increases the gap to a convolutional network, and the same convolutional network augmented with semantic loss to 11.40 and 7.10, resp.

LLM detoxification Lastly, we consider the task of LLM

Table 3. Evaluation of LM toxicity and quality across different detoxification methods GPT-2 with 124 million parameters. Model toxicity is evaluated on REALTOXICITYPROMPTS benchmark through Perspective API. **Full** refers to the full set of prompts, **Toxic** and **Nontoxic** refer to the toxic and nontoxic subsets of prompts. **PPL** refers to the model perplexity (exponentiated average log-likelihood; a measure of language modeling quality) on the WebText validation set.

Models		Avg. Toxicity (↓) Full	Toxic	Nontoxic	Valid. PPL
Domain-Adaptive Training	GPT-2	0.12 ± 0.15	0.67 ± 0.12	0.10 ± 0.11	24.52
	SGEAT	$\mathbf{0.07 \pm 0.09}$	0.64 ± 0.11	$\mathbf{0.06 \pm 0.08}$	25.93
	PseudoSL	$\mathbf{0.07 \pm 0.09}$	$\mathbf{0.61 \pm 0.09}$	0.07 ± 0.09	26.60

detoxification. That is, we investigate the effectiveness of logical constraints, enforced using pseudo-semantic loss, at steering the model away from toxic prompted-generations. We choose a *very* simple constraint to be *minimized* by pseudo-semantic loss throughout this task, namely we minimize the probability that any of a list of profanity, slurs, and swear words[4] appear as part of the model generations. Following previous work (Gehman et al., 2020; Wang et al., 2022), we evaluate on the REALTOXICITYPROMPTS, a dataset of almost 100k prompts ranging from non-toxic, assigned a toxicity score of 0, to very toxic, assigned a toxicity score of 1. We focus on GPT-2 (Radford et al., 2019) as a base model for detoxification. As is customary, (Gehman et al., 2020; Wang et al., 2022), we use Perspective API, an online automated model for toxic language and hate speech detection, to score the toxicity of our predictions. It returns scores in the range 0 to 1.0, corresponding to non-toxic on the one end, and extremely toxic on the other. Though not without limitations, studies (Wang et al., 2022; Welbl et al., 2021) have shown that the toxicity scores from Perspective API are strongly correlated with human evaluations.

We compare off-the-shelf GPT-2 against SGEAT (Wang et al., 2022)—which finetunes GPT-2 on the non-toxic portion of its self generation through performing unconditional text generation and retaining only generations with toxicity < 0.5—and against SGEAT augmented with pseudo-semantic loss. We report the average toxicity on all generations, the toxic portion, and the non-toxic portion, averaged over 5 different seeds. To understand the impact of detoxification, we also evaluate the quality of the finetuned LLM using perplexity on the validation split of WebText.

Our results are shown in Table 3. It was previously shown that SGEAT lowers the toxicity of the generations produced by GPT-2, albeit at a slight cost in terms of perplexity. This is confirmed by our numbers, where we see that SGEAT reduces the toxicity across all data splits. We also see that using pseudo-semantic loss along side SGEAT greatly improves the average toxicity of the generations considered to be toxic (i.e., scored as > 0.5 by Perspective API) from 0.64 to 0.61, as much as the improvement achieved by SGEAT compared to GPT-2, at a slight increase in perplexity. We note that this is to be expected: the validation set on which we evaluate our perplexity contains toxic-sentences. By decreasing the probability of these sentences under our model, we are less well aligned to GPT-2, at least on the toxic generations, and therefore achieve a higher perplexity.

Fidelity evaluation Lastly, we evaluated the fidelity of our approximation. We compare the entropy of our approximate distribution to the true distribution. We want this quantity to be low, as it would mean our approximation only considers assignments centered around the model sample. We also evaluate the KL-divergence of our approximate distribution from the true distribution in the neighborhood of a model sample. We want this quantity to also be low, as it corresponds to how faithful our approximation is to the true distribution in the neighborhood of the model sample. Intuitively, the KL-divergence measures the extra bits needed to encode samples from our approximation using a code optimized for GPT-2, and is zero when the two distributions coincide. We find the entropy of GPT-2 is 80.89 bits while the entropy of our approximation is, on average, 35.08 bits. We also find the KL-divergence $D_{\mathrm{KL}}(\tilde{p}_{\boldsymbol{y}} \;||\; p_{\boldsymbol{\theta}})$ is on average 4.8 bits. That is we only need 4 extra bits on average to encode the true distribution w.r.t. our approximation. Intuitively, we want to stay close to the sample to ensure high fidelity, while retaining a distribution to ensure differentiability and maximum generality within tractability bounds.

6. Conclusion

In conclusion, we proposed pseudo-semantic loss, a neuro-symbolic loss for learning with logical constraints in deep generative models. Instead of attempting to enforce the constraint on the entire distribution, our approach does so on a local distribution centered around a model sample. Our approach factorizes, allowing us to efficiently compute such an approximation. Our approach is able to greatly improve the *accuracy* and *consistency* of the baselines on structured-output prediction tasks, and is more effective at reducing the toxicity of GPT-2 compared to adaptive SoTA.

[4]List downloaded from here.

REFERENCES

Kareem Ahmed, Tao Li, Thy Ton, Quan Guo, Kai-Wei Chang, Parisa Kordjamshidi, Vivek Srikumar, Guy Van den Broeck, and Sameer Singh. Pylon: A pytorch framework for learning with constraints. In *Proceedings of the 36th AAAI Conference on Artificial Intelligence (Demo Track)*, feb 2022a.

Kareem Ahmed, Stefano Teso, Kai-Wei Chang, Guy Van den Broeck, and Antonio Vergari. Semantic probabilistic layers for neuro-symbolic learning. In *NeurIPS*, 2022b.

Kareem Ahmed, Eric Wang, Kai-Wei Chang, and Guy Van den Broeck. Neuro-symbolic entropy regularization. In *The 38th Conference on Uncertainty in Artificial Intelligence*, 2022c.

Kareem Ahmed, Kai-Wei Chang, and Guy Van den Broeck. Semantic strengthening of neuro-symbolic learning. In *Proceedings of the 26th International Conference on Artificial Intelligence and Statistics (AISTATS)*, apr 2023a.

Kareem Ahmed, Zhe Zeng, Mathias Niepert, and Guy Van den Broeck. Simple: A gradient estimator for k-subset sampling. In *Proceedings of the International Conference on Learning Representations (ICLR)*, may 2023b.

Julian Besag. Statistical analysis of non-lattice data. *Journal of the Royal Statistical Society. Series D (The Statistician)*, pages pp. 179–195, 1975.

Matko Bošnjak, Tim Rocktäschel, Jason Naradowsky, and Sebastian Riedel. Programming with a differentiable forth interpreter. In *Proceedings of the 34th ICML*, 2017.

YooJung Choi, Antonio Vergari, and Guy Van den Broeck. Probabilistic circuits: A unifying framework for tractable probabilistic modeling. 2020.

Wang-Zhou Dai, Qiu-Ling Xu, Yang Yu, and Zhi-Hua Zhou. Tunneling neural perception and logic reasoning through abductive learning, 2018.

Adnan Darwiche and Pierre Marquis. A knowledge compilation map. *JAIR*, 2002.

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

Ivan Donadello, Luciano Serafini, and Artur d'Avila Garcez. Logic tensor networks for semantic image interpretation. In *IJCAI*, 2017.

Marc Fischer, Mislav Balunovic, Dana Drachsler-Cohen, Timon Gehr, Ce Zhang, and Martin Vechev. DL2: Training and querying neural networks with logic. In *ICML*, 2019.

Samuel Gehman, Suchin Gururangan, Maarten Sap, Yejin Choi, and Noah A. Smith. Realtoxicityprompts: Evaluating neural toxic degeneration in language models. *ArXiv*, abs/2009.11462, 2020.

Eleonora Giunchiglia and Thomas Lukasiewicz. Coherent hierarchical multi-label classification networks. *Advances in Neural Information Processing Systems*, 33: 9662–9673, 2020.

Eleonora Giunchiglia and Thomas Lukasiewicz. Multi-label classification neural networks with hard logical constraints. *Journal of Artificial Intelligence Research*, 72: 759–818, 2021.

Sylvain Gugger, Lysandre Debut, Thomas Wolf, Philipp Schmid, Zachary Mueller, and Sourab Mangrulkar. Accelerate: Training and inference at scale made simple, efficient and adaptable. https://github.com/huggingface/accelerate, 2022.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, June 2016.

Nicholas Hoernle, Rafael-Michael Karampatsis, Vaishak Belle, and Ya'akov Gal. Multiplexnet: Towards fully satisfied logical constraints in neural networks. In *AAAI*, 2022.

Angelika Kimmig, Stephen Bach, Matthias Broecheler, Bert Huang, and Lise Getoor. A short introduction to probabilistic soft logic. In *Proceedings of the NIPS Workshop on Probabilistic Programming: Foundations and Applications*, 2012.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2017.

Robin Manhaeve, Sebastijan Dumancic, Angelika Kimmig, Thomas Demeester, and Luc De Raedt. Deepproblog: Neural probabilistic logic programming. In *NeurIPS*, 2018.

Mattia Medina Grespan, Ashim Gupta, and Vivek Srikumar. Evaluating relaxations of logic for neural networks: A comprehensive study. In *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21*, pages 2812–2818, 8 2021.

Wannes Meert. Pysdd. In *Recent Trends in Knowledge Compilation, Report from Dagstuhl Seminar 17381*, sep 2017.

James Mullenbach, Sarah Wiegreffe, Jon Duke, Jimeng Sun, and Jacob Eisenstein. Explainable prediction of medical codes from clinical text. *arXiv preprint arXiv:1802.05695*, 2018.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper/2019/file/bdbca288fee7f92f2bfa9f7012727740-Paper.pdf.

Marin Vlastelica Pogančić, Anselm Paulus, Vit Musil, Georg Martius, and Michal Rolinek. Differentiation of blackbox combinatorial solvers. In *ICLR*, 2020.

Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. 2019.

Tim Rocktäschel, Sameer Singh, and Sebastian Riedel. Injecting logical background knowledge into embeddings for relation extraction. In *Proceedings of the 2015 Conference of the NAACL*, 2015.

Dan Roth. On the hardness of approximate reasoning. In *IJCAI*, pages 613–619. Morgan Kaufmann, 1993.

Leslie G. Valiant. The complexity of enumeration and reliability problems. *SIAM Journal on Computing*, 1979a.

L.G. Valiant. The complexity of computing the permanent. *Theoretical Computer Science*, 1979b.

Boxin Wang, Wei Ping, Chaowei Xiao, Peng Xu, Mostofa Patwary, Mohammad Shoeybi, Bo Li, Anima Anandkumar, and Bryan Catanzaro. Exploring the limits of domain-adaptive training for detoxifying large-scale language models. In *Neurips*, 2022.

Po-Wei Wang, Priya L. Donti, Bryan Wilder, and J. Zico Kolter. Satnet: Bridging deep learning and logical reasoning using a differentiable satisfiability solver. In *ICML*, volume 97 of *Proceedings of Machine Learning Research*, pages 6545–6554. PMLR, 2019.

Johannes Welbl, Amelia Glaese, Jonathan Uesato, Sumanth Dathathri, John F. J. Mellor, Lisa Anne Hendricks, Kirsty Anderson, Pushmeet Kohli, Ben Coppin, and Po-Sen Huang. Challenges in detoxifying language models. *ArXiv*, abs/2109.07445, 2021.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online, October 2020. Association for Computational Linguistics. URL https://www.aclweb.org/anthology/2020.emnlp-demos.6.

Jingyi Xu, Zilu Zhang, Tal Friedman, Yitao Liang, and Guy Van den Broeck. A semantic loss function for deep learning with symbolic knowledge. In *International conference on machine learning*, pages 5502–5511. PMLR, 2018.

A. Circuit Construction

Any logical formula can be compiled into a smooth, deterministic and decomposable logical circuit: every disjunction factorizes the solution space into mutually exclusive events whereas every conjunction factorizes the function into two sub-functions over disjoint sets of variables. Here is a simple albeit potentially sub-optimal recipe: order variables lexicographically. Alternate OR and AND nodes. An OR node branches on the current variable being true or false, and has two children: a left (right) AND node whose children are the positive (negative) literal and the subtree corresponding to substituting the positive (negative) literal into the formula. Repeat while variables remain. We use the PySDD compiler (Meert, 2017) which outputs circuits satisfying the above properties, in addition to structured-decomposability, which asserts that functions, or constraints, over the same variables decompose in the same manner. We say the above recipe is potentially sub-optimal as we use a fixed variable order. In general, there can be an exponential gap in the size of the logical circuit obtained using the worst and best variable order. Finding the best such order is, in general, NP-hard. However, in practice, compilers (PySDD included) use search heuristics that yield demonstrably-good orders.

B. Language Detoxification

The experiments were run on a server with an AMD EPYC 7313P 16-Core Processor @ 3.7GHz, 2 NVIDIA RTX A6000, and 252 GB RAM. Our LLM detoxification experiments utilized both GPUs using the Huggingface Accelerate (Gugger et al., 2022) library.

In order to construct our constraint, we start with the list of bad words[5] and their space-prefixed variants[6]. We then tokenize this list of augment bad words, yielding 871 unique possibly-bad tokens (some tokens are only bad when considered in context with other tokens), in addition to an extra catch-all good token to which remaining tokens map to. Our constraint then disallows all sentences containing any of the words on the augmented list, starting at any of the sentence locations 0 through len(sentence) - len(word). The code to process the list of words, the code to create the constraint as well as the constraint itself will be released with our code.

Similar to SGEAT (Wang et al., 2022), the SoTA domain-adaptive training approach to detoxification, we finetune our model on self-generations as opposed to any external dataset. More specifically, we unpromptedly generate 100k using GPT-2 through Hugging Face (Wolf et al., 2020), which are then filtered through Perspective API, keeping only the 50% most nontoxic portion of the generations. We leverage the curated nontoxic corpus to further fine-tune the pre-trained LLM with standard log-likelihood loss and adapt it to the nontoxic data domain. Unlike the two other tasks where we use model samples, we use the toxic portion of the corpus to which we apply our newly proposed pseudo-semantic loss. The intuition here is that the local perturbations of a toxic sentence are also toxic, and these are exactly the assignments whose probability we would like to penalize.

Our training script is adapted from that provided by Hugging Face[7]. We use a batch size of 16, a learning rate of 1e-5 with the AdamW optimizer (Loshchilov and Hutter, 2017) with otherwise default parameters. We did a grid search over the pseudo-semantic loss weight in the values $\{0.001, 0.005, 0.01, 0.05, 0.1, 0.5, 1, 2, 4, 8\}$. All other hyperparameters were left unchanged. Similar to (Wang et al., 2022), we use use nucleus sampling with $p = 0.9$ and a temperature of 1 during generation. A randomized 10k portion of RealToxicityPrompts dataset was used for early stopping.

For only this task, our pseudo-semantic loss implementation makes use of top-k to construct the pseudo-likelihood distribution (lines 7-12 in Algorithm 1) due to the lack of computational resources. We constructed our distribution using only the top-10 good and the top-470 toxic words.

C. Sudoku

The experiments were run on a server with an AMD EPYC 7313P 16-Core Processor @ 3.7GHz, 2 NVIDIA RTX A6000, and 252 GB RAM. Training utilized only one GPU.

We follow the experimental setting and dataset provided by Wang et al. (2019), consisting of 10K Sudoku puzzles, split into 9K training examples, and 1K test samples, all puzzles having 10 missing entries. Our model consists of an RNN with an input size of 9, a hidden dimension of 128, 5 layers, a tanh nonlinearity and a dropout of 0.2. We used Adam with default PyTorch parameters and a learning rate of 3e-4. We did a grid search over the pseudo-semantic loss weight in the values $\{0.01, 0.05\}$. Our constraint disallows any solution in which rows, columns and square are not unique.

D. Warcraft Shortest Path

The experiments were run on a server with an AMD EPYC 7313P 16-Core Processor @ 3.7GHz, 2 NVIDIA RTX A6000, and 252 GB RAM. Training utilized only one of the two GPUs. We follow the experimental setting and dataset provided by (Pogančić et al., 2020). Our training set consists of $10,000$ terrain maps curated using Warcraft II tileset. We use a CNN-LSTM model for this task. Precisely, a ResNet-18 encodes the map to an embedding of dimension 128. An LSTM with 1 layer, and a hidden size of 512 then

[5] List downloaded from here.

[6] A word will be encoded differently whether it is space-prefixed or not.

[7] Downloaded from here.

predicts the next edge in the shortest path conditioned on the input map and all previous edges. We used Adam with the default PyTorch parameters and a learning rate of 5e-4. We did a grid search over the pseudo-semantic loss weight in the values $\{0.001, 0.005, 0.01, 0.05, 0.1, 0.5, 1\}$. Our constraint disallows any prediction not a valid path connecting the upper left and lower right vertices.

E. Broader Impact

The work presented in this paper, pseudo-semantic loss, has a significant potential for positive societal impact. Neuro-symbolic learning moves us closer to models whose behavior is trustworthy, explainable and fair. This extends to critical domains such as autonomous driving, medical diagnosis and financial planning to name a few. Large language models have recently seen an exponential increase in popularity, crossing the threshold of being mere research tools into products that are utilized by the general public. Unfortunately, the same expressivity that renders these models so powerful also puts them outside the reach of current neuro-symbolic approaches. Our proposed approach, pseudo-semantic loss, tackles exactly this problem, and does so efficiently. Namely, it brings neuro-symbolic learning, and the promise of trustworthy, explainable and fair models to LLMs. And we have shown the merits of our approach when applied to LLM detoxification. We must, however, also be cognizant of the potential negative societal impacts. More precisely, in very much the same way that our approach can be used to steer the model away from toxic, or generally inconsistent, outputs it can also be used to steer the model towards toxic and harmful generations.

F. Limitations

Our approach assumes access to hard symbolic knowledge. Such knowledge is not always available, and is not always easy to capture and express symbolically. Our approach also currently only supports hard symbolic knowledge, whereas often times we might be interested in distributional soft constraints that only hold in expectation. Our approach, while tractable, requires a sufficient amount of memory in order to construct the local distribution centered around the model sample. Lastly, our approach approximates the distribution of the model locally, and although we have empirically shown it's effectiveness on three different tasks, it's not clear what guarantees one can derive in general. We view addressing all of the above limitations as very interesting and impactful future endeavors.