On the benefits of label preserving augmentations for self-supervised SAT solvers

Nikita Kostin
TUM
nikita.kostin@tum.de

Nikolaos Karalias
MIT
stalence@mit.edu

Stefanie Jegelka
TUM and MIT
stefje@mit.edu

Abstract

Data augmentations have been previously leveraged for neural SAT solvers to reduce the number of labeled instances that are required to successfully train a model. In this work, we show how data augmentations can be used to enhance neural SAT solver without access to any labeled instances. We conduct a theoretical analysis of their impact on the loss function in the self-supervised setting. Through extensive benchmarking, we establish the empirical benefits of those augmentations for both training and inference and compare them against several other augmentation techniques commonly found in the literature.

1 Introduction

Methodological advancements at the intersection of machine learning and combinatorial optimization have significantly improved the state of the art in neural solvers for many well-known combinatorial optimization (CO) problems [27, 15, 22, 25, 19, 34, 1, 5]. Due to the inherent difficulty in obtaining large amounts of labeled instances for hard combinatorial problems, a promising approach that has been proposed to reduce the amount of labeled instances that is necessary for training is data augmentation with label preserving augmentations [8]. Inspired by techniques used in SAT solvers, these are augmentations that modify the instance without affecting its satisfiability. Motivated by those results, we show how label-preserving data augmentations can be leveraged to build powerful neural SAT solvers without access to any labeled instances. Our contributions can be summarized as follows:

- On the theoretical front, we study the effect of those augmentations on the smoothness of the loss function and demonstrate that certain augmentations lead to more well behaved losses.
- We provide a comprehensive summary of data augmentation techniques for neural CO.
- We conduct several experimental comparisons of different data augmentations for self-supervised SAT solving. We show how label preserving augmentations can help with training by constructing larger augmented datasets. We also show how they can be used at inference by providing an effective way to inject randomness in the predictions of a neural network, leading to drastic improvements in performance.
- To understand their impact on optimization, we also investigate their effect when directly optimizing assignments using first order methods (e.g., Adam [7]).

2 Self supervised solvers for SAT

Background. Let $V = \{v_1, \dots, v_n\}$ be a set of Boolean variables with domain $D = \{0, 1\}$. A Boolean satisfiability instance is a tuple $T = (V, \Phi)$, where $\Phi = C_1 \wedge \cdots \wedge C_m$ is a conjunctive

39th Conference on Neural Information Processing Systems (NeurIPS 2025) Workshop: Differentiable Learning of Combinatorial Algorithms.

normal form (CNF) formula over V. In CNF formulas, each clause C_j is a disjunction of literals. The evaluation of a clause is given by $C(\mathbf{x}) \in \{0, 1\}$. An assignment is a vector $\mathbf{x} \in \{0, 1\}^n$, where $x_i = 1$ denotes $v_i = \text{True}$ and $x_i = 0$ denotes $v_i = \text{False}$. The evaluation of a given assignment $\mathbf{x}$ on a formula Φ is given by $\Phi(\mathbf{x}) \in \{0, 1\}$. We say $\mathbf{x}$ satisfies Φ if $\Phi(\mathbf{x}) = 1$. The feasible set of satisfying assignments is $\mathcal{C} = \{\mathbf{x} \in \{0, 1\}^n : \ \Phi(\mathbf{x}) = 1\}$. The SAT decision problem asks whether $\mathcal{C} \neq \emptyset$, i.e., if the formula admits any satisfying assignments. Solving the problem typically entails finding such an assignment. A common optimization variant of the problem is MAX-SAT. Let

$$f(\mathbf{x}; T) = \sum_{j=1}^{m} \mathbf{1}\big[C_j(\mathbf{x}) = 1\big]$$

be the number of satisfied clauses under $\mathbf{x}$ for formula T. The optimization problem is

$$\max_{\mathbf{x} \in \{0,1\}^n} f(\mathbf{x}; T). \tag{1}$$

Solving CO problems with neural nets. First, we provide a brief description of the standard self-supervised approach to neural CO, and then we will discuss the specifics of how we handle the case of boolean satisfiability. Given a problem instance T and input features $\mathbf{Z}_T \in \mathbb{R}^{n \times d}$, we use a neural network g_θ to map the instance data to an output prediction $\mathbf{x} \in \mathbb{R}^n$ by computing $\mathbf{x} = g_\theta(\mathbf{Z}_T; T)$. For example, the input instance could be a graph, the input features could be positional encodings for the graph, and the neural net a Multi-Layer Perceptron (MLP). The output $\mathbf{x}$ may not correspond to a discrete feasible solution. In those cases, certain rounding algorithms or heuristics can be used to map $\mathbf{x}$ to a feasible solution. The goal is for the neural net to learn to predict the optimal solution $\mathbf{x}^*$.

Self-Supervised Learning for SAT. In self-supervised CO, this is done by training NN_θ on a collection of instances $T_1, T_2, \ldots, T_m$. The neural net minimizes the problem-specific loss function $\mathcal{L}_\mathcal{T}$ computed for each instance $\mathcal{L}_\mathcal{T}(g_\theta\left(\mathbf{Z}_{T_i}; T_i\right))$ and averaged over a batch (or the entire training set). At inference time, a test instance is processed through the neural network to obtain a prediction. We follow a standard probabilistic approach [12, 6]. Given instance features $\mathbf{Z}_T \in \mathbb{R}^{n \times d}$, a neural network produces logits $\mathbf{z} = g_\theta(\mathbf{Z}_T; T) \in \mathbb{R}^n$, which are mapped to assignment probabilities $\mathbf{p} = \sigma(\mathbf{z}) \in [0, 1]^n$, where σ is a sigmoid. Here, the probability that sampling independently $v_i = \text{True}$ is given by p_i. For each clause C_j, the probability it is violated under $\mathbf{p}$ is

$$Pr[C_j(\mathbf{p}) = 0] = \prod_{i \in N^+(j)} (1 - p_i) \prod_{i \in N^-(j)} p_i. \tag{2}$$

Here $N^-(j)$ denotes the variables that are negated in clause C_j and $N^+(j)$ those that are not. Let $V_T(\mathbf{x}) = \sum_{j=1}^{m} \mathbf{1}\big[C_j(\mathbf{x}) = 1\big]$ be the number of clauses violated by the assignment $\mathbf{x}$. It can be shown that under the product measure, the expected number of violated clauses is calculated by:

$$\mathbb{E}_{\mathbf{x} \sim \mathbf{p}}[V_T(\mathbf{x})] = \sum_{j=1}^{m} Pr[C_j(\mathbf{p}) = 0] = \sum_{j=1}^{m} \left(\prod_{i \in N^+(j)} (1 - p_i) \prod_{i \in N^-(j)} p_i \right). \tag{3}$$

This is a multilinear extension of the function V_T and can serve as a continuous proxy for unsatisfiability because its minima correspond to the optimal values of the optimization problem (1). If we treat this proxy as a loss function, we can use a neural network to find such minima. The training loss for a given instance is therefore defined as the expected number of violated clauses under $\mathbf{p}$:

$$\mathcal{L}_T(\mathbf{p}) \triangleq \mathbb{E}_{\mathbf{x} \sim \mathbf{p}}[V_T(\mathbf{x})]. \tag{4}$$

To obtain an assignment of truth values to the variables from the marginals $\mathbf{p}$, we will use the method of conditional expectation [3, 20] .

Properties of the loss. To better understand the optimization properties of our loss function, we will focus on its smoothness. This will also help us analyze the effect of specific data augmentations in subsequent sections.

Theorem 2.1 (Global L–smoothness of the multilinear loss $\mathcal{L}_T$)**.** *Fix a CNF instance $T = (V, \Phi)$ with $V = \{v_1, \ldots, v_n\}$ and $\Phi = C_1 \wedge \cdots \wedge C_m$. For each clause C_j, let $N^+(j) \subseteq [n]$ be the indices of variables that appear in C_j with positive polarity (i.e., as literals v_i), and let $N^-(j) \subseteq [n]$ be* *those that appear with negative polarity (i.e., as negated literals $\neg v_i$).*[1] *Let the arity (i.e., the number of literals) of a clause be $a(C_j)$ and, for each variable index i, let its degree be $d_i := \big|\{\, j : i \in N^+(j) \cup N^-(j)\,\}\big|$. Define, for each $i \in [n]$,*

$$L_i(T) \;:=\; \sum_{j:\, i \in N^+(j) \cup N^-(j)} \big(a(C_j) - 1\big), \tag{5}$$

and let L be their maximum, i.e., $L(T) \;:=\; \max_{i\in[n]} L_i(T)$. Then $\mathcal{L}_T$ is globally L–smooth on $[0,1]^n$ with respect to the Euclidean norm, i.e.,

$$\|\nabla^2 \mathcal{L}_T(\mathbf{p})\|_2 \;\le\; L \qquad \textit{for all } \mathbf{p} \in [0,1]^n. \tag{6}$$

The proof can be found in the appendix. L-smoothness is often a desirable property since it enables faster convergence to stationary points in both convex and non-convex settings. Intuitively, the Lipschitz constant of the gradient is obtained by a bound on the operator norm of the Hessian, which in turn depends on the sums of partial derivatives (row sums of the Hessian). The row sums depend on clause arities and variable occurences. Smaller values of L are desirable which implies that smaller clause arity and smaller number of variable occurrences can be beneficial.

Since no ground truth information about the solutions is available during optimization, some methods in the literature optimize a neural model directly on the test data [23, 2].This setting is essentially a reparametrized optimization problem that is solved with first-order methods (e.g., Adam [13]). Additionally, one may also remove the model and directly optimize an assignment in the hypercube by minimizing the loss, as is often done in non-convex optimization settings [14].

3 Neural CO with data-transformations

Our focus will be on the self-supervised learning paradigm for SAT and on label-preserving augmentations. However, it should be noted that many of the techniques that we will discuss could be extended to the supervised and reinforcement learning settings and to different problems.

3.1 Augmenting input instances

An important category of augmentations involves transforming the instance itself. Edge and node dropout are a standard technique that has been used in the literature on graph neural networks [21, 18]. However, there are some obstacles when applying dropout in the context of combinatorial optimization. The most important consideration is whether the transformation preserves the original solution. For example, for a combinatorial problem on graphs, node and edge dropouts clearly will affect the graph structure and so they will not preserve the optimal solution in general. Therefore, it may be more appropriate to consider label preserving augmentations.

Following [8], we consider a collection of label-preserving augmentations that are motivated by techniques used in SAT solvers: Add Unit Literal (AU), Subsumed Clause Elimination (SC), Clause Resolution (CR), and Variable Elimination (VE). Please see the Appendix for a detailed description and examples. These augmentations can be leveraged during training, inference, or even with direct optimization. These augmentations can be applied in stochastic fashion to benefit both training and inference. A total of τ rounds of LPAs are performed. In each round, each LPA is applied according to its corresponding probability. Next, we outline the different settings in which they can be used.

Augmentations for training. While these augmentations have been previously used for contrastive pretraining and combined with a supervised classifier [8], we instead use them directly for dataset augmentation. That is, given a training dataset of n unlabeled instances $\mathcal{D}$, we may increase the number of unlabeled examples in the dataset by generating LPAs of the original n instances to obtain the augmented dataset $\tilde{\mathcal{D}}$. Then a model is trained according to the setting described in section 2, i.e. given a model g_θ, its parameters θ are optimized with stochastic gradient descent (e.g., Adam [13]) to minimize

$$\mathbb{E}_{T\sim\tilde{\mathcal{D}}}[\mathcal{L}_T(g_\theta(\mathbf{Z}_T;T))] = \frac{1}{|\tilde{\mathcal{D}}|}\sum_{T\in\tilde{\mathcal{D}}}\sum_{j=1}^{|\mathcal{F}_T|} Pr[C_j(g_\theta(\mathbf{Z}_T;T)) = 0]. \tag{7}$$

[1] Throughout, we call the polarity of the variable negative if the variable occurs negated ($-$) in the clause and positive otherwise.

Inference time augmentations. Augmentations can also be an invaluable tool at inference time. We may provide τ augmented copies of an instance T as inputs to the model. This produces τ predictions, one for each input. Typically, each soft prediction from the network will be discretized (i.e., predictions in $[0,1]^n$ will be mapped to $\{0,1\}^n$), yielding a total of τ assignments.

Direct optimization. Another setting in which we will examine the effect of augmentations is that of direct optimization. In this case, for a given instance T, starting from a random initial point $\mathbf{x}_0 \in [0,1]^n$, we will directly minimize the loss $\mathcal{L}_T$ with Adam until the iterates $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n$ converge to a (local) minimum. Augmentations here are used as a way to restart the algorithm. After convergence, starting from a new random point, Adam is executed on an augmented version of the instance until convergence and the process is repeated for τ times.

3.2 Effect of augmentations on the smoothness of the loss

To develop a better understanding of the consequences of different data augmentations for optimization we study how LPAs affect the smoothness of the loss function. Lower values of the smoothness constant L are generally preferable as they enable faster convergence to stationary points and lead to more robust losses. According to theorem 2.1, we expect that augmentations that increase the number of clauses and/or the arity of clauses will increase the smoothness constant and will therefore have a detrimental effect on optimization. The following is a straightforward consequence of theorem 2.1.

Corollary 3.1. *Let $T = (V, \Phi)$ be a SAT instance. Let T_{AU}, T_{SC}, T_{CR} be the instance T after applying each of the following augmentations: Add Unit Literal, Subsumed Clause Elimination, and Clause Resolution. The following hold:*

- *$\mathcal{L}_{T_{AU}}$ is $L(T_{AU})$-smooth with $L(T_{AU}) \geq L(T)$.*
- *$\mathcal{L}_{T_{SC}}$ is $L(T_{SC})$-smooth with $L(T_{SC}) \leq L(T)$.*
- *$\mathcal{L}_{T_{CR}}$ is $L(T_{CR})$-smooth with $L(T_{CR}) \geq L(T)$.*

Proof can be found in the appendix. Intuitively, the AU augmentation increases the arity of several clauses by introducing to them a negated literal and also increases the number of variable occurrences by adding new clauses. Therefore, this leads to a larger L in (5). For SC, recall that if a clause contains another, then the larger clause is removed. Note that eliminating a clause according to this augmentation will reduce the number of terms in the sum in 5. These terms will also be of larger arity. This will lead to a smaller value of L for the loss. Finally, in resolution we add a new clause which increase the number of variable occurrences, and hence negatively affect smoothness. A similar inequality cannot be established for variable elimination by resolution.

3.3 Other augmentations

Here we will provide a brief overview of the different kinds of noising procedures that can be incorporated in neural CO pipelines. Each technique can be instantiated in several ways so we present a high level overview. Specific instantiations of those techniques have been leveraged in the literature and are also extensively benchmarked in our experiments.

Augmenting input features. A natural way to enhance the self-supervised pipeline is to introduce stochasticity in the input. Edge dropout was used in conjunction with noisy input features to improve downstream performance [6]. Noisy inputs have been used in the self-supervised setting in order to obtain multiple predictions from a model, effectively turning it into a randomized algorithm [26, 12, 28, 17].There are typically two ways that this is done in the literature. The first perturbs or augments the node features of the input graph with a random vector while the second uses random assignments as initial features. Noisy inputs are also useful in directly optimizing the model on a test set in self-supervised CO. Researchers will supply the model with some noisy initial features, and restart the model if the model gets stuck on a local minimum [23, 11]. Noisy inputs have also been leveraged in reinforcement learning for SAT to generate multiple action trajectories [33].

Chained/recurrent predictions. One benefit of using assignments as inputs is that one can leverage this for prediction chaining. This has been done in the literature with transformers [31] and message passing models [17]. It involves starting from some assignment as input to the model, producing a new assignment as output, and feeding the assignment back to the model as input recurrently.

Layerwise Noise. Another approach is to introduce noise at intermediate stages of the model prediction. The way this is implemented is model-specific. For example, a ℓ-layer neural network where each layer takes as input a collection of d (soft) assignments to variables (i.e., $[0,1]^{n\times d}$), lifts them to a higher dimensional representation in $\mathbb{R}^{n\times f}$ and then projects back down to $[0,1]^{n\times d}$. This could be done by common graph neural network layers combined with a simple linear layer. We may insert Gaussian noise on the layer output and clip to ensure that all values remain in $[0,1]$. This is provided as input to the next layer where the same process can be repeated.

Output prediction augmentation. Another approach that has been considered is that of output augmentations. A common way to improve results is to also perturb the output of the neural network at inference time before discretizing it. This has been done in the literature using Gumbel-Sinkhorn [16], for cardinality constrained problems [29] and the maximum common subgraph problem [32]. Another way to augment the self-supervised learning pipeline is to introduce stochasticity to the outputs i.e., after the neural network has produced a soft prediction but before the vector is rounded to an assignment. This procedure allows us to sample multiple solutions from a single neural net output.

4 Experiments

In this section, we perform an experimental evaluation of the proposed methods in context of inference time, training time, and direct optimization. For the evaluation, we use SAT instances sampled from three different distributions: random 3-SAT, dominating set, and k-clique identification. To be more precise, random 3-SAT formulas follow the $\texttt{rand}_3(n,m)$ distribution, where n denotes the number of variables, and m denotes the number of clauses. For k-domset and k-color the problem instances are generated using a random Erdős–Rényi graph [9] with N nodes and edge probability p. We use WalkSAT [24] with τ additional executions as the stochastic baseline and the Lingeling solver [4] for computing the upper bound of satisfiable formulas in the generated dataset (see Appendix D for dataset statistics). Our neural approaches follow the "perform τ additional executions and choose the best output" pipeline discussed in section 3. The influence of τ on the performance in different settings is presented in Table 1, Table 2, as well as in Appendix D. For the neural model, we use a custom architecture that utilizes a mix of GINE [30] and linear layers (see Appendix C for details).

Table 1: Average number of unsatisfied clauses (↓ is better) depending on the number of additional executions τ. The standard deviation of each test group is reported using superscript.

Dataset	Method	$\tau=0$	$\tau=1$	$\tau=2$	$\tau=4$	$\tau=8$	$\tau=16$	$\tau=32$	$\tau=100$
$\texttt{rand}_3$ $(100,430)$	`WalkSAT`	$1.74^{\pm1.28}$	$1.49^{\pm1.12}$	$1.14^{\pm0.86}$	$1.01^{\pm0.80}$	$0.89^{\pm0.76}$	$0.79^{\pm0.71}$	$0.63^{\pm0.61}$	$0.56^{\pm0.59}$
	`Augmentations`	$1.43^{\pm0.96}$	$\mathbf{1.27}^{\pm\mathbf{0.80}}$	$1.18^{\pm0.81}$	$1.14^{\pm0.79}$	$\mathbf{1.02}^{\pm\mathbf{0.70}}$	$\mathbf{0.97}^{\pm\mathbf{0.70}}$	$\mathbf{0.90}^{\pm\mathbf{0.67}}$	$\mathbf{0.79}^{\pm\mathbf{0.69}}$
	`LN`	$1.43^{\pm0.93}$	$1.38^{\pm0.90}$	$1.38^{\pm0.89}$	$1.37^{\pm0.90}$	$1.36^{\pm0.88}$	$1.34^{\pm0.87}$	$1.34^{\pm0.87}$	$1.33^{\pm0.85}$
	`Input Noise`	$\mathbf{1.40}^{\pm\mathbf{0.93}}$	$1.31^{\pm0.84}$	$1.29^{\pm0.91}$	$1.27^{\pm0.80}$	$1.23^{\pm0.79}$	$1.19^{\pm0.77}$	$1.16^{\pm0.76}$	$1.08^{\pm0.79}$
	`Output Noise`	$1.46^{\pm0.95}$	$1.45^{\pm0.95}$	$1.43^{\pm0.90}$	$1.38^{\pm0.91}$	$1.40^{\pm0.90}$	$1.40^{\pm0.91}$	$1.34^{\pm0.86}$	$1.35^{\pm0.86}$
	`Chaining`	$1.44^{\pm0.95}$	$1.30^{\pm0.89}$	$\mathbf{1.16}^{\pm\mathbf{0.80}}$	$\mathbf{1.09}^{\pm\mathbf{0.78}}$	$1.03^{\pm0.76}$	$1.02^{\pm0.74}$	$1.01^{\pm0.70}$	$0.96^{\pm0.70}$
$\texttt{domset}_3$ $(15,0.3)$	`WalkSAT`	$0.65^{\pm0.58}$	$0.60^{\pm0.65}$	$0.57^{\pm0.59}$	$0.56^{\pm0.59}$	$0.51^{\pm0.58}$	$0.50^{\pm0.58}$	$0.50^{\pm0.58}$	$0.50^{\pm0.58}$
	`Augmentations`	$0.67^{\pm0.64}$	$0.66^{\pm0.65}$	$0.62^{\pm0.63}$	$0.62^{\pm0.63}$	$0.62^{\pm0.63}$	$\mathbf{0.57}^{\pm\mathbf{0.59}}$	$0.58^{\pm0.61}$	$\mathbf{0.55}^{\pm\mathbf{0.59}}$
	`LN`	$0.66^{\pm0.64}$	$0.65^{\pm0.63}$	$0.64^{\pm0.61}$	$0.65^{\pm0.63}$	$0.63^{\pm0.60}$	$0.64^{\pm0.61}$	$0.63^{\pm0.61}$	$0.63^{\pm0.61}$
	`Input Noise`	$0.66^{\pm0.62}$	$\mathbf{0.64}^{\pm\mathbf{0.61}}$	$0.62^{\pm0.63}$	$\mathbf{0.59}^{\pm\mathbf{0.60}}$	$0.61^{\pm0.62}$	$0.59^{\pm0.60}$	$\mathbf{0.56}^{\pm\mathbf{0.61}}$	$0.57^{\pm0.62}$
	`Output Noise`	$\mathbf{0.64}^{\pm\mathbf{0.61}}$	$0.66^{\pm0.62}$	$0.66^{\pm0.64}$	$0.67^{\pm0.64}$	$0.66^{\pm0.61}$	$0.64^{\pm0.61}$	$0.68^{\pm0.65}$	$0.64^{\pm0.61}$
	`Chaining`	$0.67^{\pm0.65}$	$0.67^{\pm0.64}$	$\mathbf{0.61}^{\pm\mathbf{0.62}}$	$0.63^{\pm0.61}$	$\mathbf{0.59}^{\pm\mathbf{0.60}}$	$0.59^{\pm0.60}$	$0.59^{\pm0.60}$	$0.60^{\pm0.62}$
$\texttt{kclique}_3$ $(15,0.2)$	`WalkSAT`	$0.09^{\pm0.29}$	$0.07^{\pm0.26}$	$0.06^{\pm0.24}$	$0.06^{\pm0.24}$	$0.06^{\pm0.24}$	$0.06^{\pm0.24}$	$0.06^{\pm0.24}$	$0.06^{\pm0.24}$
	`Augmentations`	$\mathbf{0.25}^{\pm\mathbf{0.44}}$	$0.24^{\pm0.43}$	$0.19^{\pm0.39}$	$\mathbf{0.14}^{\pm\mathbf{0.35}}$	$\mathbf{0.12}^{\pm\mathbf{0.33}}$	$\mathbf{0.11}^{\pm\mathbf{0.31}}$	$\mathbf{0.11}^{\pm\mathbf{0.31}}$	$\mathbf{0.10}^{\pm\mathbf{0.30}}$
	`LN`	$0.27^{\pm0.47}$	$0.25^{\pm0.44}$	$0.23^{\pm0.42}$	$0.25^{\pm0.44}$	$0.24^{\pm0.43}$	$0.24^{\pm0.43}$	$0.23^{\pm0.42}$	$0.24^{\pm0.43}$
	`Input Noise`	$0.27^{\pm0.45}$	$\mathbf{0.21}^{\pm\mathbf{0.41}}$	$\mathbf{0.18}^{\pm\mathbf{0.39}}$	$\mathbf{0.14}^{\pm\mathbf{0.35}}$	$0.13^{\pm0.34}$	$\mathbf{0.11}^{\pm\mathbf{0.31}}$	$\mathbf{0.11}^{\pm\mathbf{0.31}}$	$\mathbf{0.10}^{\pm\mathbf{0.30}}$
	`Output Noise`	$0.28^{\pm0.47}$	$0.25^{\pm0.44}$	$0.27^{\pm0.47}$	$0.25^{\pm0.46}$	$0.28^{\pm0.47}$	$0.26^{\pm0.44}$	$0.26^{\pm0.46}$	$0.26^{\pm0.46}$
	`Chaining`	$0.26^{\pm0.44}$	$0.23^{\pm0.42}$	$0.23^{\pm0.42}$	$0.22^{\pm0.42}$	$0.20^{\pm0.40}$	$0.19^{\pm0.39}$	$0.19^{\pm0.39}$	$0.19^{\pm0.39}$

Inference-time augmentations. For evaluation of the inference methods, we first train a single model for each of the three datasets (`rand`/`domset`/`kclique`) using the SSL approach from section 2. We use 1000 instances for the training datasets and 100 instances for the test sets. The gist of the neural methods is as follows: (i) `Augmentations` generate a label-preserving perturbation of the input, (ii) `Layerwise Noise` (`LN`) uses pure restarts using the same input features (randomness comes from noise within the model, see Appendix C) (iii) `Input Noise` uses a random initial assignment $\{0,1\}^n$, (iv) `Output Noise` performs a direct perturbation of the probabilities $\tilde{\mathbf{p}} = \Pi_{[0,1]^n}(\mathbf{p}+\varepsilon)$ with $\varepsilon \sim \mathcal{N}(0, 0.01)$ before passing them to the discretization step, and (v) `Chaining` feeds the soft outputs $[0,1]^n$ of the model back as inputs. Note: the best assignment is chosen after the discretization procedure. For augmentations, the resulting assignment is projected back to the original formula. In particular, this means that if a variable is deleted, its assignment is assumed to be 0.5.

In Table 1, we report the empirical performance of each method. Augmentations outperform the rest of the inference methods on `rand`, and demonstrate results similar with `Input Noise` on `domset` and `kclique`. On `rand` augmentations are able to provide significant improvements after $\tau = 8$, whereas the other approaches mostly showcased marginal performance boost. In Appendix D, we report results on additional datasets as well as the percentage of satisfied formulas per test run. It is worth highlighting that WalkSAT with 100 restarts achieves near optimal results on all of the datasets.

Augmenting training data. When training the network on a problem with scarce data, augmentations can provide a way to generate artificial samples. In this experiment, we take a look at the effectiveness of these instances on $\texttt{rand}_3(100, 430)$. To do so, we generate training sets of equal total size, consisting either of 100% original formulas or half original formulas enriched with another augmented half. We then measure the generalization capabilities of models trained on these datasets using a test set of 100 $\texttt{rand}_3(100, 430)$ samples (D) as well as a test set containing both 100 original and 100 augmented formulas (D_{aug}). Note that in this setting we perform just a single forward pass. As can be seen in Figure 1, augmented data samples can serve as a substitute for original formulas if the size of the original set is high enough. This is especially noticeable in the 400-sample setting, where combining original and augmented data achieves nearly the same performance as a regular 400-sample dataset, clearly outperforming the 200-sample version.

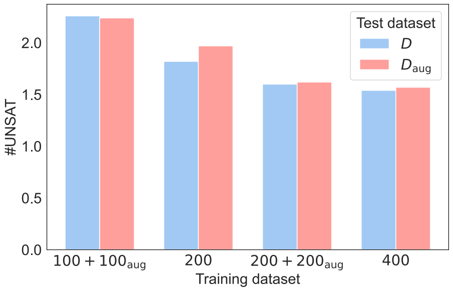

Figure 1: Average number of UNSAT clauses (↓ is better) on different train and test setups.

Optimization. We further measure performance of the proposed methods in pure optimization setting on 100 $\texttt{rand}_3(100, 430)$ formulas. To optimize a single instance, we run Adam with a learning rate of 0.01 until the training loss (see (4)) improves less than 0.001 over the last 100 epochs. We execute τ additional restarts on the input and pick the best loss without discretization. The starting point is chosen as follows: (i) `Noop` uses a constant 0.5 assignment for each run, (ii) `Noise` adds the noise from $\mathcal{N}(0,1)$ before the sigmoid, and (iii) `Augmentations` starts with a constant 0.5 assignment on an augmented instance.

Table 2: Average loss (↓ is better) reported for the gradient descent experiments. The standard deviation of the loss for each test group is reported in superscript.

Method	$\tau = 0$	$\tau = 1$	$\tau = 2$	$\tau = 4$	$\tau = 8$	$\tau = 16$	$\tau = 32$
`Noop`	$4.36^{\pm 1.75}$	$4.36^{\pm 1.75}$	$4.36^{\pm 1.75}$	$4.36^{\pm 1.75}$	$4.36^{\pm 1.75}$	$4.36^{\pm 1.75}$	$4.36^{\pm 1.75}$
`Noise`	$5.48^{\pm 2.06}$	$4.39^{\pm 1.74}$	$4.07^{\pm 1.77}$	$3.66^{\pm 1.51}$	$2.95^{\pm 1.30}$	$\mathbf{2.51}^{\pm \mathbf{1.27}}$	$\mathbf{2.09}^{\pm \mathbf{1.07}}$
`Augmentations`	$\mathbf{4.28}^{\pm \mathbf{1.67}}$	$\mathbf{3.78}^{\pm \mathbf{1.53}}$	$\mathbf{3.48}^{\pm \mathbf{1.45}}$	$\mathbf{3.09}^{\pm \mathbf{1.39}}$	$\mathbf{2.82}^{\pm \mathbf{1.29}}$	$2.65^{\pm 1.25}$	$2.34^{\pm 1.08}$

Table 2 reports the average loss per setup and the number of restarts. Interestingly, while starting off with better performance, `Noise` surpasses `Augmentations` with growing τ, which differs the direct optimization setting from neural nets.

5 Conclusion

We have proposed the use of label preserving augmentations in self-supervised neural SAT solving and examined their theoretical and empirical benefits. There are several open problems in this direction. Our theoretical results on smoothness are just the first step and more investigation is required to understand the effects of augmentations on escaping local minima, generalization, and the effect of augmentations on the function being learned.

References

[1] Sungsoo Ahn, Younggyo Seo, and Jinwoo Shin. Learning what to defer for maximum independent sets. In *International Conference on Machine Learning*, pages 134–144. PMLR, 2020.

[2] Ismail R Alkhouri, George K Atia, and Alvaro Velasquez. A differentiable approach to the maximum independent set problem using dataless neural networks. *Neural Networks*, 155:168–176, 2022.

[3] Noga Alon and Joel H Spencer. *The probabilistic method*. John Wiley & Sons, 2016.

[4] Armin Biere et al. Lingeling, plingeling and treengeling entering the sat competition 2013. *Proceedings of SAT competition*, 2013:1, 2013.

[5] Maximilian Böther, Otto Kißig, Martin Taraz, Sarel Cohen, Karen Seidel, and Tobias Friedrich. What's wrong with deep learning in tree search for combinatorial optimization. *arXiv preprint arXiv:2201.10494*, 2022.

[6] Fanchen Bu, Hyeonsoo Jo, Soo Yong Lee, Sungsoo Ahn, and Kijung Shin. Tackling prevalent conditions in unsupervised combinatorial optimization: Cardinality, minimum, covering, and more. *arXiv preprint arXiv:2405.08424*, 2024.

[7] Simon S Du, Xiyu Zhai, Barnabas Poczos, and Aarti Singh. Gradient descent provably optimizes over-parameterized neural networks. *arXiv preprint arXiv:1810.02054*, 2018.

[8] Haonan Duan, Pashootan Vaezipoor, Max B Paulus, Yangjun Ruan, and Chris Maddison. Augment with care: Contrastive learning for combinatorial problems. In *International Conference on Machine Learning*, pages 5627–5642. PMLR, 2022.

[9] P ERDdS and A R&wi. On random graphs i. *Publ. math. debrecen*, 6(290-297):18, 1959.

[10] Matthias Fey and Jan Eric Lenssen. Fast graph representation learning with pytorch geometric. In *ICLR (Workshop on Representation Learning on Graphs and Manifolds)*, volume 7, 2019.

[11] Yuma Ichikawa. Controlling continuous relaxation for combinatorial optimization. *Advances in Neural Information Processing Systems*, 37:47189–47216, 2024.

[12] Nikolaos Karalias and Andreas Loukas. Erdos goes neural: an unsupervised learning framework for combinatorial optimization on graphs. *arXiv preprint arXiv:2006.10643*, 2020.

[13] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[14] Anastasios Kyrillidis, Anshumali Shrivastava, Moshe Vardi, and Zhiwei Zhang. Fouriersat: A fourier expansion-based algebraic framework for solving hybrid boolean constraints. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 1552–1560, 2020.

[15] Yang Li, Jinpei Guo, Runzhong Wang, and Junchi Yan. T2t: From distribution learning in training to gradient search in testing for combinatorial optimization. *Advances in Neural Information Processing Systems*, 36:50020–50040, 2023.

[16] Gonzalo Mena, David Belanger, Scott Linderman, and Jasper Snoek. Learning latent permutations with gumbel-sinkhorn networks. In *International Conference on Learning Representations*, 2018.

[17] Emils Ozolins, Karlis Freivalds, Andis Draguns, Eliza Gaile, Ronalds Zakovskis, and Sergejs Kozlovics. Goal-aware neural sat solver. In *2022 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2022.

[18] Pál András Papp, Karolis Martinkus, Lukas Faber, and Roger Wattenhofer. Dropgnn: Random dropouts increase the expressiveness of graph neural networks. *Advances in Neural Information Processing Systems*, 34:21997–22009, 2021.

[19] Ruizhong Qiu, Zhiqing Sun, and Yiming Yang. DIMES: A differentiable meta solver for combinatorial optimization problems. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho, editors, *Advances in Neural Information Processing Systems*, 2022.

[20] Prabhakar Raghavan. Probabilistic construction of deterministic algorithms: approximating packing integer programs. *Journal of Computer and System Sciences*, 37(2):130–143, 1988.

[21] Yu Rong, Wenbing Huang, Tingyang Xu, and Junzhou Huang. Dropedge: Towards deep graph convolutional networks on node classification. *arXiv preprint arXiv:1907.10903*, 2019.

[22] Sebastian Sanokowski, Sepp Hochreiter, and Sebastian Lehner. A diffusion model framework for unsupervised neural combinatorial optimization. In *Proceedings of the 41st International Conference on Machine Learning*, pages 43346–43367, 2024.

[23] Martin JA Schuetz, J Kyle Brubaker, and Helmut G Katzgraber. Combinatorial optimization with physics-inspired graph neural networks. *Nature Machine Intelligence*, 4(4):367–377, 2022.

[24] Bart Selman, Henry A Kautz, Bram Cohen, et al. Local search strategies for satisfiability testing. *Cliques, coloring, and satisfiability*, 26:521–532, 1993.

[25] Zhiqing Sun and Yiming Yang. Difusco: Graph-based diffusion solvers for combinatorial optimization. *arXiv preprint arXiv:2302.08224*, 2023.

[26] Jan Toenshoff, Martin Ritzert, Hinrikus Wolf, and Martin Grohe. Run-csp: Unsupervised learning of message passing networks for binary constraint satisfaction problems. *arXiv preprint arXiv:1909.08387*, 2019.

[27] Jan Tönshoff, Berke Kisin, Jakob Lindner, and Martin Grohe. One model, any csp: Graph neural networks as fast global search heuristics for constraint satisfaction. *arXiv preprint arXiv:2208.10227*, 2022.

[28] Haoyu Wang and Pan Li. Unsupervised learning for combinatorial optimization needs meta-learning. *International Conference on Learning Representations*, 2023.

[29] Runzhong Wang, Li Shen, Yiting Chen, Xiaokang Yang, Dacheng Tao, and Junchi Yan. Towards one-shot neural combinatorial solvers: Theoretical and empirical notes on the cardinality-constrained case. In *The Eleventh International Conference on Learning Representations*, 2022.

[30] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *International Conference on Learning Representations*, 2018.

[31] Yudong W Xu, Wenhao Li, Scott Sanner, and Elias B Khalil. Self-supervised transformers as iterative solution improvers for constraint satisfaction. *arXiv preprint arXiv:2502.15794*, 2025.

[32] Chaolong Ying, Yingqi Ruan, Xuemin Chen, Yaomin Wang, and Tianshu Yu. Neural graduated assignment for maximum common edge subgraphs. *arXiv preprint arXiv:2505.12325*, 2025.

[33] Emre Yolcu and Barnabás Póczos. Learning local search heuristics for boolean satisfiability. *Advances in Neural Information Processing Systems*, 32, 2019.

[34] Dinghuai Zhang, Hanjun Dai, Nikolay Malkin, Aaron Courville, Yoshua Bengio, and Ling Pan. Let the flows tell: Solving graph combinatorial optimization problems with gflownets. *arXiv preprint arXiv:2305.17010*, 2023.

A Augmentation probabilities

Dataset	AU	SC	CR	VE
rand	0.0	1.0	1.0	0.2
domset	0.0	1.0	0.0	0.4
kclique	0.0	1.0	0.2	0.0

Table 3: Augmentation probabilities for AU, SC, CR, and VE reported per dataset.

In our work, we follow the augmentation pipeline described in [8]. In particular, we focus on the Subsumed Clause Elimination (SC), Clause Resolution (CR), and Variable Elimination (VE) operations. Add Unit Literal (AU) is omitted. The order of their execution is VE $\rightarrow$ CR $\rightarrow$ SC. In Table 3 we report the probabilities of each operation being applied depending on the dataset. To choose a proper configuration for each dataset, we started with SC $=$ CR $= 1.0$ and AU $=$ VE $= 0.0$ and performed a local grid search on each parameter with a step size of 0.2. To determine the objective function for a given configuration, we used an inference step with $\tau = 4$.

B Label-preserving augmentations

We describe a set of label-preserving augmentations (LPAs) for Boolean satisfiability problems. Each transformation preserves the satisfiability status of a formula φ, i.e., for the augmented instance $\widehat{\varphi}$ we have

$$f(\widehat{\varphi}) = f(\varphi),$$

where f is the satisfiability function. Thus, if φ is satisfiable (resp. unsatisfiable), the augmented instance $\widehat{\varphi}$ is also satisfiable (resp. unsatisfiable).

Unit Propagation (UP). If a formula contains a unit clause ℓ, one may set $\ell = 1$ and simplify accordingly by removing all clauses satisfied by ℓ and deleting $\neg\ell$ from all remaining clauses. *Example:* $\varphi = (x_1) \wedge (\neg x_1 \vee x_2) \quad \longrightarrow \quad \varphi' = (x_2)$.

Add Unit Literal (AU). One can introduce a new literal as a unit clause. Furthermore its negation is added to some predetermined number of existing clauses. Finally, new random clauses that include the literal are also added. *Example:* $\varphi = (x_1 \vee x_2) \quad \longrightarrow \quad \varphi' = (x_3) \wedge (x_1 \vee x_2 \vee \neg x_3) \wedge (x_3 \vee \neg x_1)$.

Pure Literal Elimination (PL). A variable is pure if it occurs only with one polarity in the formula. In this case, all clauses containing the variable can be safely removed. *Example:* $\varphi = (x_1 \vee x_2) \wedge (x_1 \vee \neg x_3) \quad \longrightarrow \quad \varphi' = \top$.

Subsumed Clause Elimination (SC). If one clause is a subset of another, the larger clause is redundant and may be deleted. *Example:* $\varphi = (x_1 \vee x_2) \wedge (x_1 \vee x_2 \vee x_3) \quad \longrightarrow \quad \varphi' = (x_1 \vee x_2)$.

Clause Resolution (CR). Given two clauses containing complementary literals, one may combine them into a new resolvent clause implied by both. *Example:* $(x_1 \vee x_2),\ (\neg x_1 \vee x_3) \quad \longrightarrow$ resolvent $(x_2 \vee x_3)$.

Variable Elimination (VE). For a variable x, one considers the set of clauses containing x and those containing $\neg x$, and replaces them with all possible resolvents. This effectively removes x from the formula while maintaining satisfiability. *Example:* $\varphi = (x_1 \vee x_2) \wedge (\neg x_1 \vee x_3) \quad \longrightarrow \quad \varphi' = (x_2 \vee x_3)$.

These transformations provide effective label-preserving augmentations for SAT. In the context of contrastive learning, resolution-based augmentations such as CR and VE often yield harder positive pairs, which improve representation quality (see more in [8]).

C Architecture details

We use a custom architecture as the neural SAT solver. This model leverages blocks containing a mixture of GINE [30] and linear layers and uses SiLU activation functions. Both input and output of the model are assignment probabilities $\mathbf{p} \in [0,1]^n$. Some methods (such as Chaining) can feed an initial SAT assignment as the input features. If no input assignment is provided, it is inferred from node positional encodings obtained with random walk. Each block takes a collection of d soft assignments (as discussed in subsection 3.3) as its inputs and outputs. Inside these blocks, the model operates on higher dimensional features, not tied to the assignment representation. To make the model more robust, we perturb the features between blocks with a random Gaussian noise $\varepsilon \sim \mathcal{N}(0, 10^{-4})$. Note that we use the same architecture parameters across different datasets. All of the components are implemented using publicly available PyTorch Geometric [10] code. All of the experiments in this paper were executed on a single Nvidia A100 GPU with 40GB of VRAM.

D Additional inference-time results

In this section, we report additional results for inference-time augmentations. Percentage of satisfiable formulas obtained with the Lingeling solver for all of the formula distributions used in this paper are reported in Table 4. First, we take a look at the influence of standalone augmentations on the inference pipeline (Table 5). Here, just the selected augmentation is applied in each execution. For instance for unit literal we set $\texttt{AU} = 1.0$ and $\texttt{CR} = \texttt{VE} = \texttt{SC} = 1.0$. The size of the test set here is set to 25 instances. Results on the additional datasets and the percentages of satisfied formulas are further denoted in Table 6 and Table 7.

Dataset	Parameters	SAT Percentage
$\texttt{rand}_3$	$(100, 430)$	49%
$\texttt{domset}_3$	$(15, 0.3)$	54%
$\texttt{domset}_4$	$(12, 0.2)$	55%
$\texttt{kclique}_3$	$(15, 0.3)$	94%
$\texttt{kclique}_3$	$(20, 0.05)$	11%

Table 4: Percentage of satisfiable formulas per dataset.

Table 5: Standalone influence of augmentations on the inference pipeline. Same as before, wee report the average number of unsatisfied clauses (↓ is better). The standard deviation for each test set is reported in superscript.

Dataset	Aug.	$\tau = 0$	$\tau = 1$	$\tau = 2$	$\tau = 4$	$\tau = 8$	$\tau = 16$	$\tau = 32$	$\tau = 100$
$\texttt{rand}_3$ $(100, 430)$	AU	$1.16^{\pm 0.94}$	$1.08^{\pm 0.91}$	$1.20^{\pm 1.12}$	$1.08^{\pm 0.91}$	$1.12^{\pm 0.88}$	$1.12^{\pm 0.88}$	$1.16^{\pm 0.94}$	$1.20^{\pm 1.00}$
	CR	$1.16^{\pm 0.94}$	$\mathbf{0.96}^{\pm \mathbf{0.68}}$	$\mathbf{0.96}^{\pm \mathbf{0.68}}$	$\mathbf{0.84}^{\pm \mathbf{0.55}}$	$\mathbf{0.76}^{\pm \mathbf{0.60}}$	$\mathbf{0.72}^{\pm \mathbf{0.54}}$	$\mathbf{0.64}^{\pm \mathbf{0.49}}$	$\mathbf{0.56}^{\pm \mathbf{0.51}}$
	VE	$1.32^{\pm 1.22}$	$1.16^{\pm 0.90}$	$1.12^{\pm 0.88}$	$1.04^{\pm 0.89}$	$1.16^{\pm 0.94}$	$1.08^{\pm 0.91}$	$1.12^{\pm 0.97}$	$1.08^{\pm 0.91}$
	SC	$\mathbf{1.08}^{\pm \mathbf{0.91}}$	$1.08^{\pm 0.91}$	$1.04^{\pm 0.89}$	$1.00^{\pm 0.82}$	$1.00^{\pm 0.82}$	$1.00^{\pm 0.82}$	$0.92^{\pm 0.70}$	$0.96^{\pm 0.79}$
$\texttt{domset}_3$ $(15, 0.3)$	AU	$0.64^{\pm 0.70}$	$0.64^{\pm 0.76}$	$0.64^{\pm 0.70}$	$0.60^{\pm 0.71}$	$0.64^{\pm 0.70}$	$0.60^{\pm 0.71}$	$0.60^{\pm 0.71}$	$0.64^{\pm 0.70}$
	CR	$\mathbf{0.60}^{\pm \mathbf{0.71}}$	$0.64^{\pm 0.70}$	$0.64^{\pm 0.81}$	$0.60^{\pm 0.71}$	$0.56^{\pm 0.65}$	$0.52^{\pm 0.65}$	$0.48^{\pm 0.65}$	$0.44^{\pm 0.58}$
	VE	$\mathbf{0.60}^{\pm \mathbf{0.65}}$	$0.60^{\pm 0.71}$	$0.60^{\pm 0.71}$	$0.64^{\pm 0.76}$	$0.60^{\pm 0.71}$	$0.64^{\pm 0.70}$	$0.64^{\pm 0.76}$	$0.60^{\pm 0.65}$
	SC	$\mathbf{0.60}^{\pm \mathbf{0.71}}$	$\mathbf{0.56}^{\pm \mathbf{0.65}}$	$\mathbf{0.52}^{\pm \mathbf{0.71}}$	$\mathbf{0.52}^{\pm \mathbf{0.65}}$	$\mathbf{0.52}^{\pm \mathbf{0.65}}$	$\mathbf{0.44}^{\pm \mathbf{0.58}}$	$\mathbf{0.44}^{\pm \mathbf{0.58}}$	$\mathbf{0.40}^{\pm \mathbf{0.58}}$
$\texttt{kclique}_3$ $(15, 0.2)$	AU	$\mathbf{0.32}^{\pm \mathbf{0.48}}$	$\mathbf{0.32}^{\pm \mathbf{0.48}}$	$\mathbf{0.32}^{\pm \mathbf{0.48}}$	$0.32^{\pm 0.48}$	$0.32^{\pm 0.48}$	$0.32^{\pm 0.48}$	$\mathbf{0.24}^{\pm \mathbf{0.44}}$	$\mathbf{0.16}^{\pm \mathbf{0.37}}$
	CR	$\mathbf{0.32}^{\pm \mathbf{0.48}}$	$\mathbf{0.32}^{\pm \mathbf{0.48}}$	$\mathbf{0.32}^{\pm \mathbf{0.48}}$	$0.32^{\pm 0.48}$	$0.32^{\pm 0.48}$	$0.32^{\pm 0.48}$	$0.32^{\pm 0.48}$	$0.28^{\pm 0.46}$
	VE	$\mathbf{0.32}^{\pm \mathbf{0.48}}$	$\mathbf{0.32}^{\pm \mathbf{0.48}}$	$\mathbf{0.32}^{\pm \mathbf{0.48}}$	$0.32^{\pm 0.48}$	$0.32^{\pm 0.48}$	$0.32^{\pm 0.48}$	$0.32^{\pm 0.48}$	$0.24^{\pm 0.44}$
	SC	$\mathbf{0.32}^{\pm \mathbf{0.48}}$	$\mathbf{0.32}^{\pm \mathbf{0.48}}$	$\mathbf{0.32}^{\pm \mathbf{0.48}}$	$\mathbf{0.24}^{\pm \mathbf{0.44}}$	$\mathbf{0.24}^{\pm \mathbf{0.44}}$	$\mathbf{0.24}^{\pm \mathbf{0.44}}$	$\mathbf{0.24}^{\pm \mathbf{0.44}}$	$0.20^{\pm 0.41}$

Table 6: Average number of unsatisfied clauses (↓ is better) reported on additional datasets. The standard deviation for each test group is reported in superscript.

Dataset	Method	$\tau = 0$	$\tau = 1$	$\tau = 2$	$\tau = 4$	$\tau = 8$	$\tau = 16$	$\tau = 32$	$\tau = 100$
$\texttt{domset}_4$ $(12, 0.2)$	WalkSAT	$0.68^{\pm 0.78}$	$0.65^{\pm 0.80}$	$0.63^{\pm 0.81}$	$0.59^{\pm 0.79}$	$0.59^{\pm 0.79}$	$0.59^{\pm 0.79}$	$0.59^{\pm 0.79}$	$0.59^{\pm 0.79}$
	Augmentations	$\mathbf{0.67}^{\pm \mathbf{0.80}}$	$0.65^{\pm 0.80}$	$0.65^{\pm 0.80}$	$0.63^{\pm 0.80}$	$0.61^{\pm 0.79}$	$0.61^{\pm 0.79}$	$0.61^{\pm 0.79}$	$0.61^{\pm 0.79}$
	LN	$0.69^{\pm 0.81}$	$0.67^{\pm 0.80}$	$0.68^{\pm 0.80}$	$0.67^{\pm 0.80}$	$0.66^{\pm 0.81}$	$0.66^{\pm 0.81}$	$0.66^{\pm 0.81}$	$0.66^{\pm 0.81}$
	Input Noise	$0.69^{\pm 0.80}$	$\mathbf{0.63}^{\pm \mathbf{0.80}}$	$\mathbf{0.62}^{\pm \mathbf{0.79}}$	$\mathbf{0.60}^{\pm \mathbf{0.79}}$	$\mathbf{0.60}^{\pm \mathbf{0.79}}$	$\mathbf{0.60}^{\pm \mathbf{0.79}}$	$\mathbf{0.60}^{\pm \mathbf{0.79}}$	$\mathbf{0.60}^{\pm \mathbf{0.79}}$
	Output Noise	$0.70^{\pm 0.80}$	$0.69^{\pm 0.81}$	$0.68^{\pm 0.80}$	$0.67^{\pm 0.79}$	$0.67^{\pm 0.79}$	$0.67^{\pm 0.80}$	$0.66^{\pm 0.79}$	$0.65^{\pm 0.80}$
	Chaining	$0.69^{\pm 0.80}$	$0.64^{\pm 0.81}$	$0.64^{\pm 0.81}$	$0.63^{\pm 0.80}$	$0.63^{\pm 0.80}$	$0.63^{\pm 0.80}$	$0.63^{\pm 0.80}$	$0.63^{\pm 0.80}$
$\texttt{kclique}_3$ $(20, 0.05)$	WalkSAT	$0.89^{\pm 0.31}$	$0.89^{\pm 0.31}$	$0.89^{\pm 0.31}$	$0.89^{\pm 0.31}$	$0.89^{\pm 0.31}$	$0.89^{\pm 0.31}$	$0.89^{\pm 0.31}$	$0.89^{\pm 0.31}$
	Augmentations	$\mathbf{0.94}^{\pm \mathbf{0.28}}$	$\mathbf{0.92}^{\pm \mathbf{0.27}}$	$0.93^{\pm 0.26}$	$0.92^{\pm 0.27}$	$0.92^{\pm 0.27}$	$0.91^{\pm 0.29}$	$0.91^{\pm 0.29}$	$\mathbf{0.90}^{\pm \mathbf{0.30}}$
	LN	$0.96^{\pm 0.32}$	$0.93^{\pm 0.26}$	$0.93^{\pm 0.26}$	$0.93^{\pm 0.26}$	$0.93^{\pm 0.26}$	$0.93^{\pm 0.26}$	$0.93^{\pm 0.26}$	$0.93^{\pm 0.26}$
	Input Noise	$0.97^{\pm 0.30}$	$\mathbf{0.92}^{\pm \mathbf{0.27}}$	$0.92^{\pm 0.27}$	$0.92^{\pm 0.27}$	$0.92^{\pm 0.27}$	$0.92^{\pm 0.27}$	$0.91^{\pm 0.29}$	$0.91^{\pm 0.29}$
	Output Noise	$\mathbf{0.94}^{\pm \mathbf{0.28}}$	$0.95^{\pm 0.26}$	$0.94^{\pm 0.28}$	$0.94^{\pm 0.28}$	$0.94^{\pm 0.28}$	$0.94^{\pm 0.28}$	$0.94^{\pm 0.28}$	$0.94^{\pm 0.24}$
	Chaining	$\mathbf{0.94}^{\pm \mathbf{0.24}}$	$0.93^{\pm 0.26}$	$\mathbf{0.90}^{\pm \mathbf{0.30}}$	$\mathbf{0.90}^{\pm \mathbf{0.30}}$	$\mathbf{0.90}^{\pm \mathbf{0.30}}$	$\mathbf{0.90}^{\pm \mathbf{0.30}}$	$\mathbf{0.90}^{\pm \mathbf{0.30}}$	$\mathbf{0.90}^{\pm \mathbf{0.30}}$

Table 7: Percentage of satisfied formulas (↑ is better) on each of the datasets depending on the number of additional executions τ.

Dataset	Method	$\tau = 0$	$\tau = 1$	$\tau = 2$	$\tau = 4$	$\tau = 8$	$\tau = 16$	$\tau = 32$	$\tau = 100$
`rand`$_3$ $(100, 430)$	`WalkSAT`	21%	15%	24%	27%	33%	37%	43%	48%
	`Augmentations`	**15%**	14%	**20%**	**20%**	21%	**24%**	**26%**	**35%**
	`LN`	**15%**	15%	14%	15%	15%	15%	15%	15%
	`Input Noise`	**15%**	14%	18%	15%	16%	17%	18%	22%
	`Output Noise`	13%	14%	13%	14%	14%	14%	15%	15%
	`Chaining`	**15%**	**16%**	19%	**20%**	**22%**	**24%**	22%	24%
`domset`$_3$ $(15, 0.3)$	`WalkSAT`	40%	48%	48%	49%	53%	54%	54%	54%
	`Augmentations`	42%	**44%**	**46%**	46%	46%	**48%**	48%	**50%**
	`LN`	**43%**	43%	43%	43%	43%	43%	44%	44%
	`Input Noise`	42%	43%	**46%**	**47%**	46%	47%	**50%**	**50%**
	`Output Noise`	**43%**	42%	43%	42%	41%	43%	42%	43%
	`Chaining`	**43%**	42%	**46%**	44%	**47%**	47%	47%	47%
`kclique`$_3$ $(15, 0.2)$	`WalkSAT`	91%	93%	94%	94%	94%	94%	94%	94%
	`Augmentations`	**75%**	76%	81%	**86%**	**88%**	**89%**	**89%**	**90%**
	`LN`	74%	75%	77%	75%	76%	76%	77%	76%
	`Input Noise`	73%	**79%**	**82%**	**86%**	87%	**89%**	**89%**	**90%**
	`Output Noise`	73%	75%	74%	76%	73%	74%	75%	75%
	`Chaining`	74%	77%	77%	78%	80%	81%	81%	81%
`domset`$_4$ $(12, 0.2)$	`WalkSAT`	46%	50%	53%	55%	55%	55%	55%	55%
	`Augmentations`	**49%**	50%	50%	52%	53%	53%	53%	53%
	`LN`	48%	49%	48%	49%	50%	50%	50%	50%
	`Input Noise`	47%	**52%**	**52%**	**54%**	**54%**	**54%**	**54%**	**54%**
	`Output Noise`	46%	48%	48%	48%	48%	49%	49%	50%
	`Chaining`	47%	**52%**	**52%**	52%	52%	52%	52%	52%
`kclique`$_3$ $(20, 0.05)$	`WalkSAT`	11%	11%	11%	11%	11%	11%	11%	11%
	`Augmentations`	**7%**	**8%**	7%	8%	8%	9%	9%	**10%**
	`LN`	**7%**	7%	7%	7%	7%	7%	7%	7%
	`Input Noise`	6%	**8%**	8%	8%	8%	8%	9%	9%
	`Output Noise`	**7%**	6%	7%	7%	7%	7%	7%	6%
	`Chaining`	6%	7%	**10%**	**10%**	**10%**	**10%**	**10%**	**10%**

E Optimization properties of the multilinear extension

Proposition E.1 (Global L–smoothness of the multilinear loss $\mathcal{L}_T$)**.** *Fix a CNF instance $T = (V, \Phi)$ with $V = \{v_1, \dots, v_n\}$ and $\Phi = C_1 \wedge \cdots \wedge C_m$. For each clause C_j, let $N^+(j) \subseteq [n]$ be the indices of variables that appear in C_j with* positive polarity *(i.e., as literals v_i), and let $N^-(j) \subseteq [n]$ be those that appear with* negative polarity *(i.e., as negated literals $\neg v_i$).*[2] *For a probability vector $\mathbf{p} = (p_1, \dots, p_n) \in [0, 1]^n$, the clause-violation probability is*

$$\Pr\big[C_j(\mathbf{p}) = 0\big] = \prod_{i \in N^+(j)} (1 - p_i) \prod_{i \in N^-(j)} p_i, \tag{8}$$

and the expected number of violated clauses (the multilinear loss) is

$$\mathcal{L}_T(\mathbf{p}) \;\triangleq\; \mathbb{E}_{\mathbf{x} \sim \mathbf{p}}[V_T(\mathbf{x})] \;=\; \sum_{j=1}^{m} \Pr\big[C_j(\mathbf{p}) = 0\big] \;=\; \sum_{j=1}^{m} \Big(\prod_{i \in N^+(j)} (1 - p_i) \prod_{i \in N^-(j)} p_i \Big). \tag{9}$$

Let the arity *of a clause be $a(C_j) := |C_j|$ and, for each variable index i, let its degree be $d_i := \big|\{\, j : i \in N^+(j) \cup N^-(j) \,\}\big|$. Define, for each $i \in [n]$,*

$$L_i \;:=\; \sum_{j\colon i \in N^+(j) \cup N^-(j)} \big(a(C_j) - 1\big), \qquad L \;:=\; \max_{i \in [n]} L_i. \tag{10}$$

[2] Throughout, we call the polarity of the variable negative if the variable occurs negated in the clause and positive otherwise.

Then $\mathcal{L}_T$ is globally L–smooth on $[0,1]^n$ with respect to the Euclidean norm, i.e.,

$$\|\nabla^2\mathcal{L}_T(\mathbf{p})\|_2 \;\le\; L \qquad \textit{for all } \mathbf{p}\in[0,1]^n. \tag{11}$$

Proof. From (9), $\mathcal{L}_T$ is multilinear (affine in each coordinate). Writing C_j also for the index set $N^+(j)\cup N^-(j)$, its first derivatives are, for each $i\in[n]$,

$$\frac{\partial \mathcal{L}_T}{\partial p_i}(\mathbf{p}) = \sum_{j:\, i\in C_j} s_{j,i} \prod_{p\in N^+(j)\setminus\{i\}} (1-p_p) \prod_{n\in N^-(j)\setminus\{i\}} p_n, \tag{12}$$

where the *sign* $s_{j,i}$ encodes the polarity of i in C_j:

$$s_{j,i} = \begin{cases} -1, & i\in N^+(j) \quad (v_i \text{ appears in } C_j),\\ \;\;1, & i\in N^-(j) \quad (\neg v_i \text{ appears in } C_j).\end{cases} \tag{13}$$

By multilinearity, the diagonal second derivatives vanish. For distinct $i\neq k$,

$$\frac{\partial^2 \mathcal{L}_T}{\partial p_i\,\partial p_k}(\mathbf{p}) = \sum_{j:\,\{i,k\}\subseteq C_j} s_{j,i,k} \prod_{p\in N^+(j)\setminus\{i,k\}} (1-p_p) \prod_{n\in N^-(j)\setminus\{i,k\}} p_n, \tag{14}$$

where

$$s_{j,i,k} = \begin{cases} \;\;1, & i \text{ and } k \text{ have the same polarity in } C_j \text{ (both in } N^+ \text{ or both in } N^-),\\ -1, & i \text{ and } k \text{ have opposite polarity in } C_j \text{ (one in } N^+\text{, the other in } N^-).\end{cases} \tag{15}$$

Every factor in the products lies in $[0,1]$, hence for all $\mathbf{p}$,

$$\left|\tfrac{\partial^2\mathcal{L}_T}{\partial p_i\,\partial p_k}(\mathbf{p})\right| \;\le\; c_{ik}, \qquad c_{ik} := \big|\{\, j \,:\, \{i,k\}\subseteq C_j \,\}\big| \quad \text{(pair codegree)}. \tag{16}$$

Summing absolute values across the ith row of the Hessian (whose diagonal is zero) gives

$$\begin{align}
\sum_{k\neq i}\left|\tfrac{\partial^2\mathcal{L}_T}{\partial p_i\,\partial p_k}(\mathbf{p})\right| \;&\le\; \sum_{k\neq i}\;\sum_{j:\,\{i,k\}\subseteq C_j} 1 \tag{17}\\
&=\; \sum_{j:\, i\in C_j}\;\sum_{k\in C_j\setminus\{i\}} 1 \tag{18}\\
&=\; \sum_{j:\, i\in C_j} \big(a(C_j)-1\big) \tag{19}\\
&=\; L_i. \tag{20}
\end{align}$$

Therefore $\|\nabla^2\mathcal{L}_T(\mathbf{p})\|_\infty \le \max_i L_i = L$ for all $\mathbf{p}$. This completes the proof. □

Corollary E.2 (Bound using variable degrees.). *Let $a_{\max} := \max_j a(C_j)$ and $\Delta := \max_i d_i$. Then*

$$L \;\le\; (a_{\max}-1)\,\Delta. \tag{21}$$

Proof. For each i,

$$\begin{align}
L_i &= \sum_{j:\, i\in C_j} (a(C_j)-1) \tag{22}\\
&\le \sum_{j:\, i\in C_j} (a_{\max}-1) \tag{23}\\
&= (a_{\max}-1)\, d_i \tag{24}\\
&\le (a_{\max}-1)\,\Delta. \tag{25}
\end{align}$$

The inequality in (25) is obtained by taking the max over all variable degrees. □

Theorem E.3. *Let $T=(V,\Phi)$ be a SAT instance. Let T_{AU}, T_{SC}, T_{CR}, T_{VE} be the instance T after applying each of the following augmentations: Add Unit Literal, Subsumed Clause Elimination, Clause Resolution, Variable Elimination. The following hold:*

- $\mathcal{L}_{T_{AU}}$ *is* $L(T_{AU})$*-smooth with* $L(T_{AU}) \geq L(T)$*.*
- $\mathcal{L}_{T_{SC}}$ *is* $L(T_{SC})$*-smooth with* $L(T_{SC}) \leq L(T)$*.*
- $\mathcal{L}_{T_{CR}}$ *is* $L(T_{CR})$*-smooth with* $L(T_{CR}) \geq L(T)$*.*

Proof. The proof follows from theorem 2.1 To establish the inequalities of the theorem we simply have to compute the new value of L_i for the augmented instance and compare it to its original value.

Add Unit Literal. Let $S^- \subseteq [m]$ be indices of clauses to which $\neg y$ is appended, and let $y \vee R_1, \dots, y \vee R_t$ be new clauses with $r_\ell := |R_\ell|$. For $i \in [n]$ define

$$\alpha_i^- := \big|\{j \in S^- : i \in N^+(j) \cup N^-(j)\}\big|, \qquad \beta_i := \big|\{\ell \in [t] : i \in R_\ell\}\big|. \tag{26}$$

Then

$$L_i(T_{\mathrm{AU}}) = L_i(T) + \alpha_i^- + \sum_{\ell:\, i \in R_\ell} r_\ell, \qquad i \in [n], \tag{27}$$

$$\tag{28}$$

and for the new clauses

$$L_y(T_{\mathrm{AU}}) = \sum_{j \in S^-} a(C_j) + \sum_{\ell=1}^{t} r_\ell. \tag{29}$$

$$\tag{30}$$

The smoothness constant is given by

$$L(T_{\mathrm{AU}}) = \max\Big\{\max_{i \in [n]} L_i(T_{\mathrm{AU}}), L_y(T_{\mathrm{AU}})\Big\}, \tag{31}$$

which yields $L(T_{\mathrm{AU}}) \geq L(T)$.

Subsumed clause elimination. If S collects all subsumed clauses that are dropped, then

$$L_i(T_{\mathrm{SC}}) = L_i(T) - \sum_{j \in S:\, i \in C_j} \big(a(C_j) - 1\big). \tag{32}$$

Therefore, we have that

$$L(T_{\mathrm{SC}}) \leq L(T).$$

Clause Resolution. Let $\mathcal{R}$ be the set of added resolvents, with variable sets $U(R)$ and arities $a(R)$. Then

$$L_i(T_{\mathrm{CR}}) = L_i(T) + \sum_{\substack{R \in \mathcal{R} \\ i \in U(R)}} \big(a(R) - 1\big), \qquad . \tag{33}$$

This gives us the smoothness constant

$$\begin{aligned} L(T_{\mathrm{CR}}) &= \max_i \Big(L_i(T) + \sum_{R \ni i} \big(a(R) - 1\big)\Big) && (34) \\ &\geq L(T). && (35) \end{aligned}$$

□