SAT Solver implementations analysis

Carlo Kohmann

Dipartimento di Informatica Giovanni Degli Antoni, Corso di Laurea Magistrale in Informatica, Via Celoria 18, Milano, Italy.

Contributing authors: carlo.kohmann@studenti.unimi.it;

1 Introduction

This project is developed for educational purposes and explores the application of machine learning and symbolic reasoning techniques to the task of propositional logic formula classification. Specifically, the goal is to determine whether a given well-formed propositional logic formula is satisfiable (SAT) or unsatisfiable (UNSAT).

The project provides a series of scripts that allow for the comparison of three different approaches:

- Pretrained BERT (`bert-base-uncased`) fine-tuned for the SAT/UNSAT classification task.
- Mini-BERT, a lightweight BERT model trained from scratch with a custom tokenizer designed specifically for propositional logic formulae.
- A procedural solver, based on resolution and CNF transformation, that deterministically evaluates the satisfiability of the formula.

2 Formula Generation

The first step of the project consists in the automatic generation of well-formed propositional logic formulae, which are then used to train and evaluate the learning models. The generation process satisfies the following constraints:

- **Well-formedness:** All generated formulae are syntactically valid, i.e., they are constructed recursively using a predefined set of propositional variables and logical connectives in accordance with the grammar of propositional logic.
- **Balanced dataset:** The dataset is constructed in a balanced fashion, containing an equal number of satisfiable (SAT) and unsatisfiable (UNSAT) formulae. This ensures that the classification models are not biased toward the majority class.

- **Expressive completeness:** The formulae are constructed using only three logical connectives: conjunction ($\wedge$), disjunction ($\vee$), and negation ($\neg$), which are sufficient to express any Boolean function and thus guarantee the expressive completeness of the generated logic.

Formally, each formula is generated through a recursive random procedure that takes as input a maximum depth parameter. The process ensures structural diversity while maintaining syntactic correctness. At each level of recursion:

- A base case selects a random propositional variable from a fixed set (e.g., `a--g`).
- At higher depths, binary connectives (`AND`, `OR`) are applied to subformulae, with optional negation applied to the result.

Once a candidate formula is generated, its satisfiability is determined using the `sympy` logic module. Depending on the result, it is assigned a binary label (1 for SAT, 0 for UNSAT). This process is repeated until a predefined number of SAT and UNSAT formulae is reached, ensuring class balance.

The complete dataset is then randomly shuffled and split into two disjoint subsets:

- **Training set:** Consisting of 1000 formulae (500 SAT, 500 UNSAT), used for model training.
- **Test set:** Consisting of 200 formulae (100 SAT, 100 UNSAT), used for performance evaluation.

Both subsets are stored in CSV format with two columns: `formula` and `satisfiable`, where the latter contains binary labels.

The formula generation process is encapsulated in a dedicated Python script and can be executed independently. Upon completion, it outputs the two CSV files `propositional_dataset_train.csv` and `propositional_dataset_test.csv`, which are later used by the model training and evaluation scripts.

3 PLLM Bert

This chapter describes the use of a *pretrained large language model* to classify propositional logic formulas according to their satisfiability. In particular, we employed the `bert-base-uncased` model, which was already pretrained on large-scale text corpora, and subsequently fine-tuned to distinguish between satisfiable (`SAT`) and unsatisfiable (`UNSAT`) formulas.

3.1 Architecture and Classification Objective

The BERT model used is the base version (12 layers, 768 hidden size, 12 attention heads), uncased, pretrained on English Wikipedia and the BooksCorpus. The classification head consists of a single linear layer, taking the `[CLS]` token representation as input and outputting two logits corresponding to the two classes: `SAT` (label 1) and `UNSAT` (label 0).

The goal of fine-tuning is to train the model to learn a decision function between these two classes using the textual representation of propositional logic formulas as input.

3.2 Fine-Tuning Script

The training process was implemented in Python using the HuggingFace `Transformers` library. The script includes the following stages:

- Loading the `propositional_dataset_train.csv` file, which contains formulas and their satisfiability labels.
- Splitting the dataset into training and validation sets with stratification.
- Tokenizing the formulas using `AutoTokenizer.from_pretrained("bert-base-uncased")`.
- Constructing a custom dataset class, `FormulaDataset`, to package tokens and labels.
- Configuring training parameters, including GPU-based training with `fp16` precision.
- Training the model using HuggingFace's `Trainer`, with evaluation at each epoch.

The final model is saved to the directory `./trained_sat_unsat_model` and reused during testing.

3.3 Model Evaluation

For final evaluation, a separate testing script was used. It loads the model and tokenizer from the saved directory and computes classification metrics on a test set of 200 formulas, synthetically generated using the script described in Chapter **??**.

The resulting metrics are as follows:

- **Accuracy**: 0.895
- **Precision (class 0)**: 0.8559
- **Recall (class 0)**: 0.9500
- **F1-score (class 0)**: 0.9005
- **Precision (class 1)**: 0.9438
- **Recall (class 1)**: 0.8400
- **F1-score (class 1)**: 0.8889

The overall model accuracy on the test set reached 89.5%, with a satisfactory balance between precision and recall across both classes. The model shows a slight tendency to favor classification as `SAT`, though with higher precision on the `UNSAT` class.

3.4 Conclusions

The results confirm that a pretrained model such as BERT can effectively learn to distinguish between satisfiable and unsatisfiable formulas. This suggests that the structural information contained in the textual representation of the formulas is sufficient to allow the model to generalize the concept of satisfiability, at least within a controlled propositional setting. But what would happen if a from scratch machine-learning model would be trained for this specific task?

4 Mini-BERT From Scratch on a Minimal Vocabulary

In this chapter, we explore the training of a BERT-based model from scratch to classify propositional logic formulas as satisfiable or unsatisfiable. Unlike the previous approach, which leveraged a pretrained language model, here we constructed and trained a custom model using a minimal vocabulary specifically tailored to the task.

4.1 Minimal Vocabulary

To focus exclusively on the syntactic structure of propositional logic, we designed a dedicated vocabulary composed of only the following tokens:

- Special tokens: `[PAD]`, `[UNK]`, `[CLS]`, `[SEP]`, `[MASK]`
- Propositional letters: `a`, `b`, `c`, `d`, `e`, `f`, `g`
- Logical connectives: | (OR), `~` (NOT), `&` (AND)
- Parentheses: `(`, `)`

This constrained vocabulary ensures that the model cannot rely on any pretrained natural language knowledge, thereby isolating its ability to learn logical patterns from scratch.

4.2 Model Architecture

The model is based on a scaled-down version of the BERT architecture (hereafter referred to as Mini-BERT), defined by the following configuration:

- **Vocabulary size**: 16
- **Hidden size**: 256
- **Number of layers**: 4
- **Attention heads**: 4
- **Intermediate size**: 512
- **Maximum sequence length**: 128
- **Number of output labels**: 2 (SAT/UNSAT)

The model was initialized with random weights and trained from scratch on a large synthetic dataset.

4.3 Training Dataset and Procedure

The training data consists of 10,000 synthetically generated propositional formulas labeled for satisfiability. This is a significantly larger dataset than the one used in Chapter 3, enabling the model to learn from a broader range of structures and logical configurations.

The dataset was split into 90% training and 10% validation, preserving the class distribution through stratification. Tokenization was performed using the custom vocabulary via a modified `BertTokenizerFast`, and the model was trained for 30 epochs using a batch size of 16 and the AdamW optimizer with a learning rate of 2×10^{-5} and weight decay of 0.01.

Training and evaluation were handled via HuggingFace's `Trainer` API, with metrics computed at the end of each epoch.

4.4 Final Evaluation

The model was evaluated on a separate test set of 200 formulas generated via the same synthesis method described in Chapter **??**. The classification performance is summarized as follows:

- **Accuracy**: 0.850
- **Precision (class 0 - UNSAT)**: 0.7869
- **Recall (class 0)**: 0.9600
- **F1-score (class 0)**: 0.8649
- **Precision (class 1 - SAT)**: 0.9487
- **Recall (class 1)**: 0.7400
- **F1-score (class 1)**: 0.8315

These results indicate a strong ability to generalize over both satisfiable and unsatisfiable formulas, with a slightly higher performance on the UNSAT class.

4.5 Conclusions

Despite being trained from scratch and under significant hardware and time constraints, the Mini-BERT model achieved an accuracy of 85%, closely matching the results of the pretrained model described in the previous chapter. This suggests that with a well-designed architecture and sufficient data, it is possible to effectively learn logical structures even in the absence of natural language pretraining.

Moreover, the success of this approach demonstrates that a symbolic vocabulary tailored to the logical domain can serve as a viable alternative to general-purpose tokenization, particularly when the task does not rely on semantic information beyond propositional structure.

5 Symbolic Reasoning with the DPLL Algorithm

5.1 Motivation

While neural language models exhibit remarkable generalization capabilities, especially in natural language understanding tasks, their performance on formal logic reasoning remains limited. This motivates the introduction of a classical symbolic reasoning approach as a performance upper bound. The Davis–Putnam–Logemann–Loveland (DPLL) algorithm, a complete backtracking-based SAT solver, is well-suited for this task due to its deterministic nature and guaranteed correctness.

5.2 Preprocessing: Conversion to CNF

To leverage the DPLL algorithm, all propositional formulas must be converted into *Conjunctive Normal Form* (CNF). This transformation is non-trivial for arbitrary logical expressions and is handled programmatically using the `sympy` library.

The CNF conversion pipeline consists of the following steps:

1. Extraction of propositional variables from each formula.
2. Symbolic parsing of formulas using `sympy`'s transformation utilities.
3. CNF translation via `to_cnf()` with structural simplification.
4. Clause normalization: each CNF expression is converted to a list-of-lists of integers representing disjunctions of literals (positive or negative).

Each formula is then transformed into a structured CNF representation suitable for symbolic reasoning. An excerpt of the preprocessing logic is shown below:

Listing 1 Formula preprocessing using sympy

```
expr = parse_expr(formula_str, local_dict=symbol_table, transformations=transfo
cnf_expr = to_cnf(expr, simplify=True)
clauses = cnf_expr_to_list(cnf_expr)
```

This preprocessing step ensures compatibility between symbolic inputs and the SAT solver, forming the foundation of the symbolic evaluation phase.

5.3 Implementation of the DPLL Solver

A recursive DPLL-based SAT solver capable of evaluating the satisfiability of CNF formulas has been implemented. The key algorithmic steps include:

- **Unit Propagation**: If a clause contains a single literal, it must be satisfied. All clauses containing it are removed, and its negation is eliminated from the rest.
- **Variable Selection**: The most frequent literal is heuristically selected as a branching variable.
- **Recursive Branching**: The solver explores both assignments (true/false) for the selected literal.
- **Backtracking**: If a contradiction (empty clause) arises, backtracking is triggered.

Listing 2 Core of the DPLL recursion

```
def dpll(self, cnf, l):
    for item in cnf:
        if len(item) == 1:
            cnf, l = self.unit_p(cnf, l)
    if [] in cnf:
        return False
    if not cnf:
        return True
    t = self.most_common(cnf)
    r1 = self.reduced(cnf, t)
    r2 = self.reduced(cnf, -t)
    return self.dpll(r1, l) or self.dpll(r2, l)
```

5.4 Evaluation and Results

The solver is evaluated against the same test set used for neural models. After converting each formula into CNF, the solver is applied and its output is compared with the labels:

Listing 3 Accuracy evaluation script

```
solver = Solver(parsed_cnf)
result = solver.solve()
if int(result) == expected:
    correct += 1
```

The symbolic solver achieves a flawless accuracy of **100%**, demonstrating the theoretical soundness and completeness of the DPLL algorithm.

```
Accuracy finale: 100.00% su 500 esempi.
```

5.5 Comparison with Neural Approaches

Table 1 summarizes the performance of different methods evaluated on the SAT/UNSAT classification task.

Approach	**Accuracy (%)**
MiniBERT	85.0
BERT fine-tuned	89.5
Symbolic Solver (DPLL)	**100.0**

Table 1 Comparison of model accuracy on the propositional reasoning dataset

6 Concluding Remarks

6.1 Symbolic vs Neural Reasoning

The experiments and analyses conducted in this project underline a crucial distinction in AI reasoning paradigms: the gap between **symbolic logic processing** and **neural language understanding**. While large language models like BERT have shown promise in capturing shallow patterns of logical form, they lack the systematic rigor and deductive capabilities of symbolic solvers when applied to strictly structured tasks.

In scenarios involving purely symbolic input, such as propositional logic formulae, the advantage of **procedural models** becomes evident. The DPLL algorithm, with its deterministic and exhaustive search mechanism, achieves perfect accuracy and provides provable correctness, something currently out of reach for transformer-based models.

6.2 Toward Hybrid Neuro-Symbolic Architectures

Despite the superior performance of procedural solvers in symbolic domains, neural models maintain an edge in processing ambiguous or naturalistic inputs. This motivates a hybrid approach: **transformers for language-to-logic translation**, followed by **symbolic solvers for inference**.

Such a pipeline would exploit the strengths of both paradigms:

- Neural models (e.g., BERT, GPT) could act as semantic translators, converting natural language statements or questions into formal logic representations.
- Symbolic engines (e.g., SAT solvers, theorem provers) would then execute precise reasoning over the resulting logical forms.

This modular strategy is especially relevant for tasks that begin in natural language but require rigorous logical resolution, such as legal reasoning, scientific hypothesis verification, or mathematical problem solving.

6.3 Related Work

A precursor to this idea was explored in the study *"Logical Reasoning Comprehension with BERT"* by myself, which investigated the ability of the `bert-base-uncased` model to solve multiple-choice questions designed to require logical inference. The experimental results showed that:

> *The overall accuracy remained relatively low (maximum 31%), indicating that even large pre-trained models struggle with structured logical inference.*

For more details about the study, readers are encouraged to contact the author via the email address provided in this document.

A broader perspective is also offered by the dataset and benchmark introduced in **LogiQA: A Challenge Dataset for Machine Reading Comprehension with Logical Reasoning**, which systematically evaluates models on tasks requiring both reading comprehension and logical deduction. The paper is available at:

https://www.ijcai.org/proceedings/2020/0501.pdf

6.4 Final Thoughts

The difficulty neural models face with logic-based tasks is not a limitation, but a challenge — one that invites collaboration between paradigms rather than competition. The future of machine reasoning likely lies in the synthesis of neural perception and symbolic precision.