Training Neural Networks Using SMT and OMT Solvers

Author: Barry Jinks

Course: CPSC 513

Date: December 19, 2024

Contents

1 Introduction

1.1 Overview

This report explores the innovative approach of training neural networks using Satisfiability Modulo Theories (SMT) and Optimization Modulo Theories (OMT) solvers, focusing on the potential advantages and limitations of these methods. Unlike traditional gradient-based training techniques, solvers provide a logic-driven, constraint-based framework for determining neural network weights and biases. This report summarizes research findings, methodology, experiments, and potential avenues for future work.

1.2 Motivation

The motivation for this study stems from the increasing complexity of machine learning models and the challenges associated with traditional training methods. Neural networks trained with gradient-based optimization methods often face issues like local minima, overfitting, and lack of interpretability. Furthermore, these methods require extensive hyperparameter tuning and can struggle to provide guarantees about the resulting models.

SMT and OMT solvers offer a logic-driven alternative by allowing neural network training to be framed as a constraint satisfaction or optimization problem. This approach is particularly appealing for applications requiring high interpretability, strict adherence to constraints, or insights into feature importance. For example, deception detection in high-stakes scenarios, such as 911 call analysis, requires not only accurate predictions but also explainable models that align with domain-specific constraints.

Another motivation arises from the potential of solvers to identify patterns in datasets with noise or loosely correlated features. Traditional methods may fail to generalize well in such cases, while solvers can leverage their inherent rigour to provide meaningful insights. By exploring the applicability of SMT and OMT solvers in these contexts, this research seeks to bridge the gap between machine learning and formal methods.

1.3 Research Questions

The research aimed to answer the following questions:

1. Can an SMT or OMT solver train weights in a neural network effectively?
2. How can activation functions compatible with such solvers be constructed?
3. Can multi-layer networks be implemented while ensuring satisfiability?
4. What are the inherent limitations of using solvers for neural network training?
5. What future work might be done to further refine the approaches investigated?

2 Methodology

2.1 Workflow

The methodology followed a systematic approach integrating machine learning techniques with several solvers. The first step involved data preprocessing. Each dataset was cleaned to handle missing or invalid entries and normalized to ensure all features were on a comparable scale, reducing numerical instability during constraint solving. After preprocessing, the dataset was divided into training and testing subsets, ensuring representative samples in both sets to assess generalization performance effectively.

As illustrated in Figure 1, a Python script was developed to translate the neural network architecture and its constraints into SMT-LIBv2 format. This encoding defined variables for weights, biases, and inputs, along with constraints for activation functions and output ranges. The constraints ensured the mathematical consistency of the network and enforced the bounds for weights and biases, crucial for maintaining interpretability and avoiding overfitting.

The training phase utilized the cvc5, z3 and OptiMath solvers to find satisfiable solutions to the encoded constraints. The solvers iteratively adjusted weights and biases to ensure the output matched the expected labels for the training data. In networks with multiple layers, constraints for intermediate outputs (activations) were also encoded, ensuring that each layer contributed to a consistent and accurate final output.

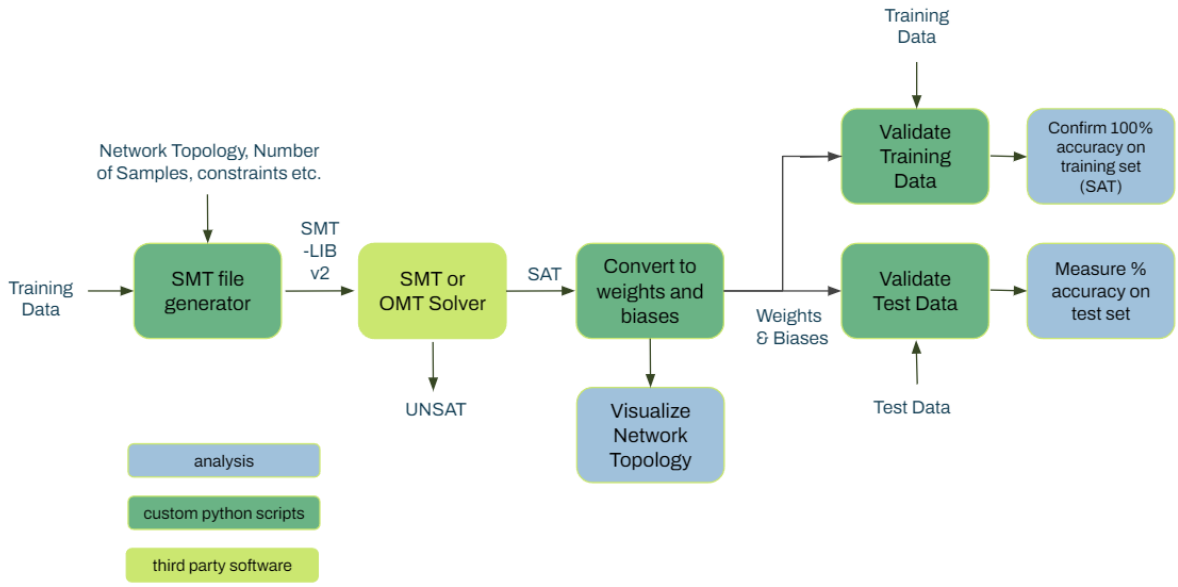


Figure 1: Tool Flow

Validation involved running the trained network on test data to measure its accuracy. The predictions were compared with the ground truth, and performance metrics were computed. These metrics were analyzed to identify patterns of overfitting or underfitting and to understand the solver's limitations in generalizing to unseen data.

Finally, the results were compared against baseline models trained with traditional gradient-based methods, such as those implemented in Scikit-learn. The comparison highlighted the strengths and weaknesses of the Solver-based approach.

All of these experiments were conducted using a Linux VM running on a Windows 64-bit machine. All the software and resulting output files can be viewed on github at https://github.com/vancuvrboy/513Project.

3 Linear Regression Classifier with SMT and OMT Solvers

The Loan Approval Dataset (1), a common benchmarking dataset for classifier experiments, was used for training and testing for this phase of the project. The data set had 11 features, 4269 samples, and a binary label "loan_status" of Approved or Rejected. After cleaning, normalization, and randomization there were 3,415 samples in the training set and 854 samples in the test set.

3.1 Software Classifier

The first step was to implement the software classifier in python as a baseline, then measure it's performance. The workflow for all software classifiers is shown in Figure 2. The software model was implemented using the LinearRegression classifier from sklearn.linear_model. After training, it achieved a 93% accuracy on the test set. One interesting, but not unexpected, observation is that loan approvals are highly correlated with credit (of CIBIL) score as can be seen in Figure 3. This leads to a large coefficient for that feature after training..

3.2 SMT Classifier

An SMT solver determines whether a given formula is satisfiable under a specific theory (e.g., linear arithmetic, non-linear arithmetic, bit-vectors, arrays, etc.). Given a logical formula, the solver checks if there exists an assignment to variables that makes the formula true. In order to train the weights of a neural network classifier using an SMT solver, the problem must be expressed as a collection of assertions that specify the relationships between features (inputs) and labels (outputs) for each sample, leaving the values of weights as the variables that must be derived to satisfy all the given constraints simultaneously. In the case of a binary classifier the output is either 1 or 0. To return SAT, the outputs

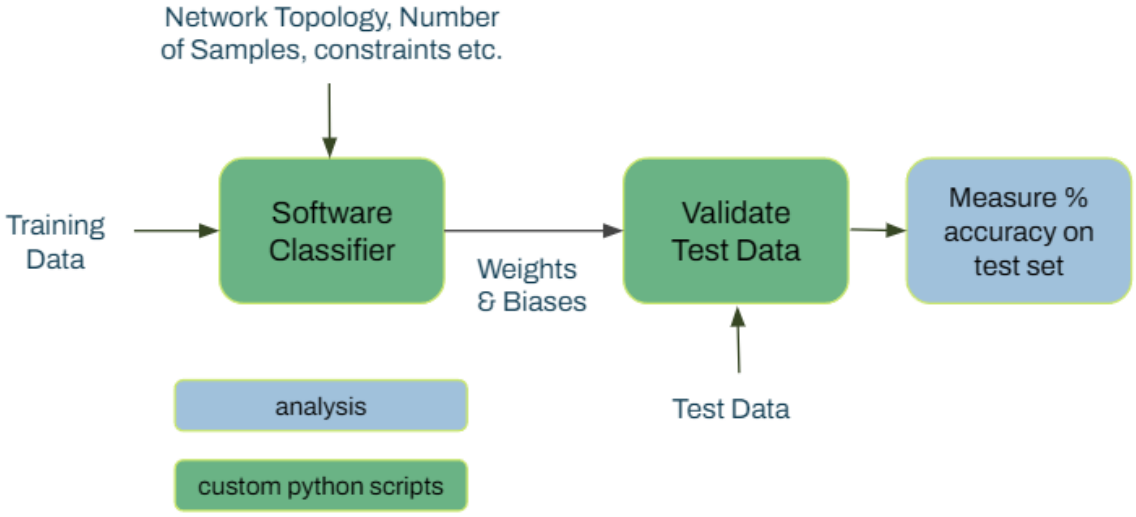


Figure 2: Software Classifier Workflow

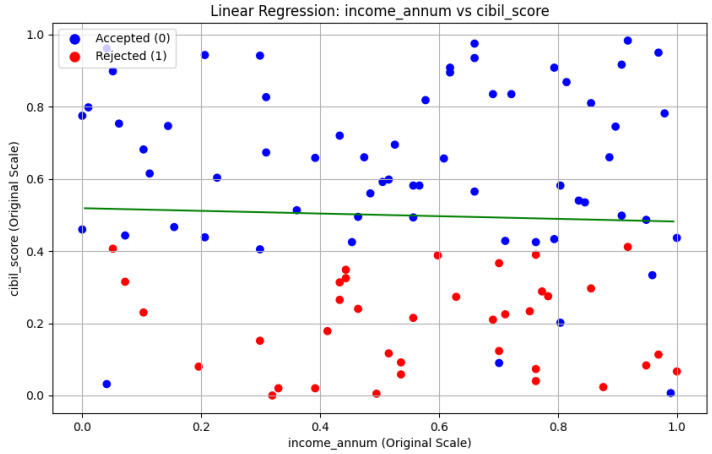
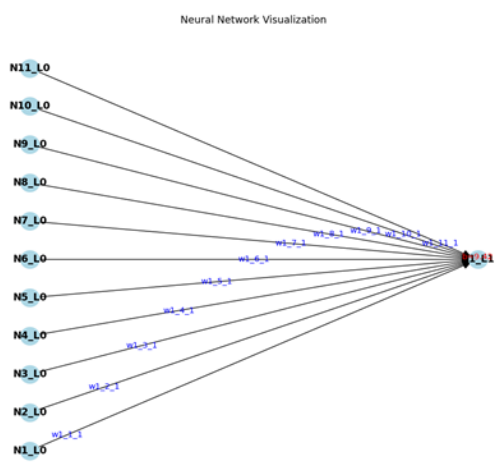


Figure 3: CIBIL score is highly correlated with Loan Approvals (blue dots)




```
; SMT-LIBv2 configurations
(set-logic QF_NRA)
(set-option :produce-models true)
(set-option :produce-unsat-cores true)

; SMT-LIBv2 file generated for cvc5 with multi-layer configuration
; Number of features (input neurons): 11
; Number of layers: 0
; Number of samples: 10

(declare-fun w1 () Real) ; Weight for feature 'no_of_dependents'
(assert (and (>= w1 -100.0) (<= w1 100.0)))
(declare-fun w2 () Real) ; Weight for feature 'education'
(assert (and (>= w2 -100.0) (<= w2 100.0)))
....
(assert (< (+ (* w1 1.0) (* w2 1.0) (* w3 0.0) ... (* w11
0.591836734693878) b) 0.0)) ; Constraint for sample 1 (Label: 0)
(assert (< (+ (* w1 0.0) (* w2 1.0) (* w3 1.0) ... (* w11
0.530612244897959) b) 0.0)) ; Constraint for sample 2 (Label: 0)
...
(check-sat)
(get-model)
```

weight constraints

sample constraints

Figure 4: Representation of a single neuron with 11 features for input to an SMT solver to model a Linear Regression Classifier. Left neural network diagram, Right snippets from SMT-LIBv2 file showing constraints

of all training samples must agree with the predicted result. Much like a typical neural network, a binary activation function is placed on the output: if the output is $>= 0.0$, then it is deemed a 1 and < 0.0, a 0. In practice, however, this can lead to outputs that are arbitrarily small, so they cannot be distinguished from one another. Thus a constraint on the output ensures that values span from -100.0 to -0.001 or +0.001 to +100.0.

3.2.1 Training an SMT Classifier

To model a linear regression classifier, a neural network with a single neuron was constructed in an SMT-LIBv2 file. The weights of each input to the neuron modeled the coefficients of the linear regression and the bias modeled the intercept. For this experiment the cvc5 SMT solver ((7)) was utilized. When the initial models were run it became apparent that the solver might find solutions with unrealistically large weights in order to overwhelm the calculation, thus making the value of certain weights immaterial. The constraints placed on the range of weight values mentioned above solved the problem. Figure 4 shows the layout of the neural network and the resulting SMT-LIBv2 file.

3.2.2 Results with SMT Solver

Table 1 illustrates the performance of a linear regression classifier modeled using the cvc5 SMT solver with varying sizes of the training set. The "Number of Samples" column reflects the number of samples in the training set used for each experiment, while the "Accuracy on Test Set" column indicates the corresponding performance on the test set. Note that for every run, the weights were verified against the training set, to ensure that there was 100% agreement between the predicted results and the ground truth, which must be the case if the solution is SAT.

Number of Samples	Accuracy on Test Set
10	40%
50	72%
72 (SAT limit)	88%
92, with 73 plucked out (2nd SAT limit)	85%

Table 1: Performance on Test Set with Different Sample Sizes

The solver was able to process up to 72 samples in the training set before producing UNSAT, achieving an accuracy of 88%. When extending the training set to 92 samples, it was necessary to exclude sample 73 (likely due to the features of the sample differing too much from the norm). Surprisingly, despite increasing the number of training samples, the accuracy on the test set dropped to 85%. This counterintuitive result suggests that the additional samples, or the removal of sample 73, may have introduced noise or reduced the model's generalizability, emphasizing the importance of carefully selecting

training samples for SMT solver-based machine learning models. One interesting observation was that, like the software linear regression classifier, the weight associated with the CIBIL score was by far the largest, confirming that the SMT solver may help to identify which features are the most significant in predicting the output.

These findings demonstrate that the SMT solver can produce results comparable to conventional methods when hyperparameters and constraints are tuned appropriately. However, due to inconsistencies (noise) in the training data it becomes increasingly difficult for the solver to return a SAT result.

3.3 OMT Classifier with Sample Slack Adjustment

In the previous experiment it became apparent that the more samples the solver was exposed to, the more difficulty it had finding a set of weights that satisfied the constraints, until eventually it became UNSAT. Some samples were clearly out of place ("noisy") in the training set. So the question was posed: could a solver help to identify these errant samples? Thus an OMT solver was employed to investigate this.

An OMT solver extends the capabilities of an SMT solver by introducing optimization. Instead of just checking satisfiability, it searches for a solution that optimizes an objective function while satisfying the constraints. It can find an optimal solution to a satisfiable problem, where "optimal" means minimizing or maximizing a specified objective function.

In the case at hand, slack variables were introduced to gain insights into which samples caused the problems. The use of slack variables in optimization traces its origins back to Linear Programming (LP) in the mid-20th century, where they were introduced to transform inequality constraints into equalities, allowing for efficient solving using algorithms like the simplex method (6). Over time, the concept extended beyond LP to various domains, including Constraint Logic Programming (CLP) (5) in the 1990s. By the early 2000s, slack variables found renewed importance when OMT solvers were introduced, where they allowed for optimization in combinatorial and logic-based problems, particularly when soft constraints or noisy data were involved. Slack variables with OMT solvers provide a means to adjust constraints and identify outlier samples while maintaining a focus on optimization.

In this study, slack variables enabled the solver to move a sample from one label to the other, thus adjusting the training set to conform to a pattern that leads to SAT. To achieve that, variables named slack# were added to (in the case of a ground truth $>= 0$) or subtracted from (in the case of ground truth < 0) the output value of every sample. The objective function was then defined as the minimum value of the sum of all slack variables necessary to get to SAT. If a sample was within boundaries, the solver would set its slack value to 0.0, but the out-of-bounds samples would receive positive slack values, thus identifying them. Figure 5 is an example of a small SMT-LIBv2 file that illustrates how these constraints are specified.

Given that the experimental setup was already based on SMT-LIBv2 files, a search was undertaken to find a compatible OMT solver that could handle non-linear problems since multi-layer networks (to be investigated later) are inherently non-linear. In this case, OptiMathSAT was chosen (4).

3.3.1 Results with OMT Solver

As with the previous experiments, for small numbers of samples, the OminMathSAT solver had no problem returning SAT. As illustrated in Table 2, the OMT solver with slack constraints does better than the SMT solver, delivering a solid 91% accuracy on the test set at 72 samples. This compares very favorably with the software linear regression classifier at 93%. Note that at 80 and 100 samples, the solver is still able to converge to SAT because of the slack variables, whereas the SMT solver stalled out at 72. At 80 samples, the process identified 6 noisy samples and at 100, 1 more. As expected, the accuracy with the test set starts to deteriorate at that point because the output values of the samples modified by the slack variables cause the values of weights to be modified. I tried to run 200 samples, but at 15 hours run time it had not converged.

To determine if the information gleaned could be used to improve the result, I re-ran the 100 sample model, with the 7 noisy samples removed. Unfortunately, it did not yield a positive result with only 80% accuracy. This is likely because the samples flagged by the optimization process were necessary to make the training set run converge to SAT, but were not necessarily indicative of the ideal pattern of weights to predict the sample outputs of unseen samples in the test set. There are likely other experiments to be run here to try to get a better result but that is out of scope for this study.

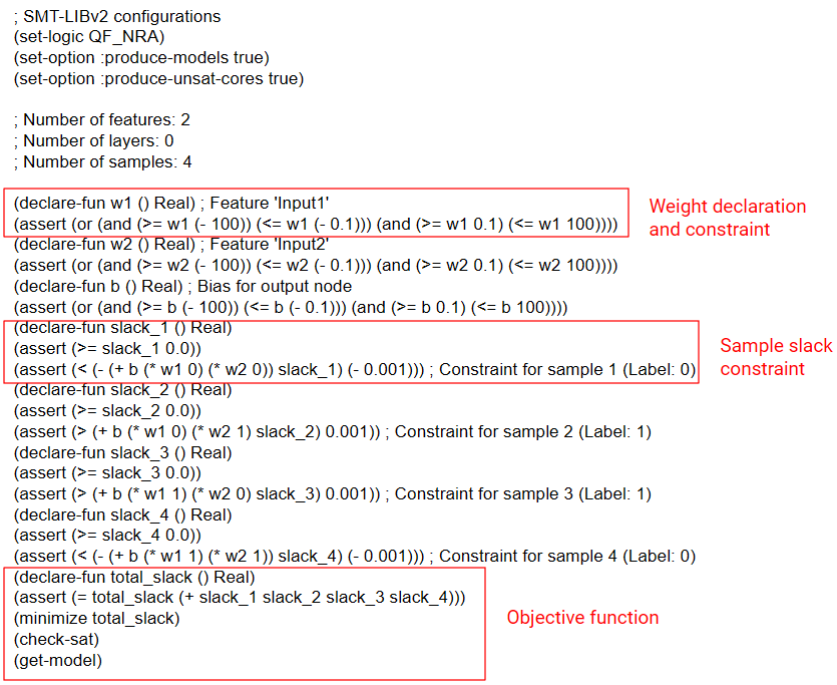


Figure 5: Small example SMT-LIBv2 file showing how slack variables are specified in order to identify "noisy" samples

Number of Samples	Accuracy on Test Set	"Noisy" Samples	Total Slack
10	52%	0	0
50	89%	0	0
72	91%	0	0
80	85%	6	0.34
100	83%	7	0.44
93 (100 - 7 samples removed)	80%	0	0

Table 2: Performance on Test Set with Different Sample Sizes

4 Multi-layer Neural Network Experiments with SMT and OMT Solvers

As illustrated in Figure 1, the SMT-LIBv2 file generator is capable of producing different network topologies, including different depths and number of neurons in hidden layers.

4.1 Structure of the SMT-LIBv2 File for Multi-Layer Networks with Optional Threshold Activation

The SMT-LIBv2 file for a multi-layer network includes the following components:

4.1.1 General Configuration

- **Logic:** Quantifier-Free Nonlinear Real Arithmetic (QF_NRA).

4.1.2 Network Specification

- Each weight in the input layer is represented as a real variable, constrained within specific bounds. These bounds are configurable at runtime, but default to the values shown below. For example:

```
(declare-fun w1_1_1 () Real)
(assert (or (and (>= w1_1_1 (- 100.0)) (<= w1_1_1 (- 0.1)))
    (and (>= w1_1_1 0.1) (<= w1_1_1 100.0))))
```

- Each weight in the hidden layers is represented as a real variable, constrained within specific bounds. These bounds are configurable at runtime, but default to the values shown below. For example:

```
(declare-fun w2_1_1 () Real)
(assert (or (and (>= w2_1_1 (- 100.0)) (<= w2_1_1 (- 0.1)))
    (and (>= w2_1_1 0.1) (<= w2_1_1 100.0))))
```

- Threshold activations between the hidden layers and output layer are optional. They enforced using logical constraints as below.

4.1.3 Threshold Activation Function

Threshold activation functions are modeled using Boolean variables and logical constraints that enforce binary activation states. For each node in the network, the activation output is declared as a Boolean variable:

```
(declare-fun z1_1_s1 () Bool) ; Binary activation for layer 1, node 1, sample 1
```

The activation output is constrained based on the node's output value. For example, if the node's output value is non-negative, the activation is set to `true`; otherwise, it is set to `false`:

```
(assert (or (and (>= node_out_1_1_s1 0) (= z1_1_s1 true))
(and (< node_out_1_1_s1 0) (= z1_1_s1 false))))
```

This constraint ensures that the binary activation aligns with the threshold behavior, where the node is activated only when its input exceeds the threshold value. This approach allows for strict enforcement of binary decision boundaries within the SMT solver framework.

4.2 Validation Using an XOR Gate Training Set

Before proceeding to train a relatively complex model, a simple multi-layer network was tested to validate that all parts of the system were working as expected. To validate the software implementation, an XOR gate training set was employed with a two-layer network configuration. The XOR gate is a well-known problem in machine learning, as it is not linearly separable and requires at least one hidden layer to solve. In this case, a threshold activation function is used between the hidden layer and the output layer.

Configuration:

- **Input Features:** Two binary inputs.
- **Hidden Layer:** Two nodes with threshold activation functions.
- **Output Layer:** Single node.
- **Constraints:** Weights and biases were constrained within the range [-1, 1].

4.2.1 Results

The SMT solver (cvc5) successfully trained the network to achieve 100% accuracy on the XOR truth table. Figure 6 confirms that the software model of a 2 layer network that models the XOR gate is identical to the logical representation modeled in the SMT-LIBv2 file. Each hidden node learned to represent an intermediate logical condition (e.g., AND or OR), and the output node combined these conditions to produce the correct XOR behavior. This validation demonstrated the solver's capability to learn non-linearly separable functions using multi-layer configurations. Interestingly, while each is logically correct, the weights generated by the software model trained using gradient decent differed from those produced by the SMT solver.

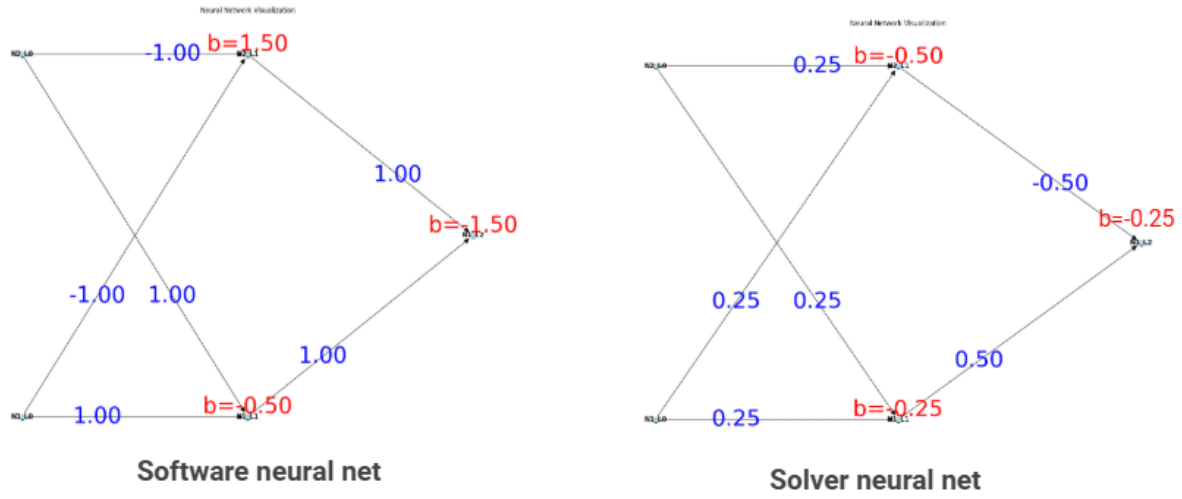


Figure 6: XOR gate implementation using a 2-layer neural network. The weights generated by the software model are on the left and by the SMT solver on the right.

4.3 Multi-Layer Experiments and Results

4.3.1 911 Call Deception Detection Dataset

An emerging area of psychological research is the detection of deception in human interaction. A specialized area is the detection of deception in 911 calls, an important area of study due to the high stakes involved in such calls. Accurately detecting deception in 911 calls can provide first responders with a more complete picture of an emergency and can potentially save lives. Some of the early work in deception detection was done by Harpster et al. in 2009 (2).

More recently, the research has been advanced by Markey et al. (3) at Villanova. Markey's research consisted of humans analyzing 911 calls for 86 semantic cues, such as "Caller acts in a reckless manner". In their experiment, callers were rated for the degree of expression of the 86 cues. Each call was then classified as deceptive ("Guilty") or not deceptive ("Not Guilty"). What resulted was a training set consisting of 82 samples and a test set of 64 samples, each with 86 features and binary labels. After analysis, they boiled down the features to a subset of 39 features that exhibited the highest correlation with predicting guilt or innocence. Unless otherwise stated, the subsequent experiments were conducted on the dataset with 39 features to reduce the complexity of the resulting models.

In the original paper, Markey et al. analyzed the data using statistical tools, but this seemed to me to be something that could be done with machine learning. After reaching out to Patrick Markey, he

was kind enough to let me know that he and his team had just completed some work wherein he was able to use machine learning techniques to predict deception with a 71% accuracy.

When I first looked at the data, an example of which is shown in Figure 7, it became apparent that it would not be easy to separate into classes using a single layer classifier.

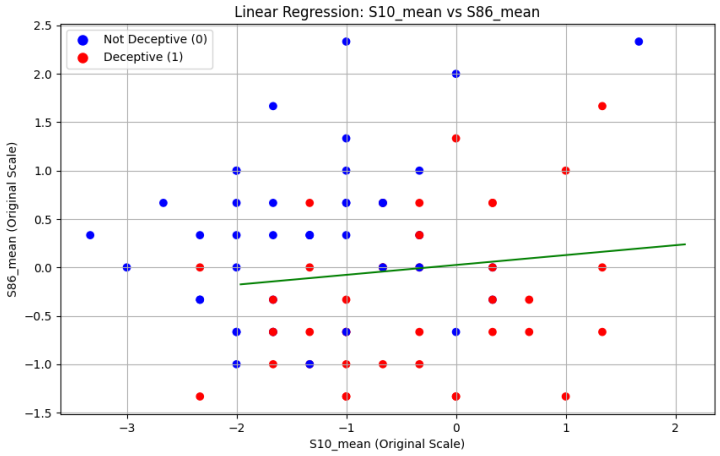


Figure 7: Plot of Cue 86 vs Cue 10 from the 911 Deception Detection Dataset. A Linear Regression Classifier (green line) Struggles to Separate the Samples into Classes.

4.3.2 Single Layer Neural Network Tests on the Deception Detection Dataset

Configuration:

- **Input Features:** 39 real number inputs, ranging from -4.0 to +4.0.
- **Hidden Layer:** N/A
- **Output Layer:** Single node.
- **Constraints:** Weights and biases were constrained within the range [-100.0, 100.0].

The first experiment was designed to run the 39 cue deception dataset on a single layer network using the SMT solver to derive the weights and biases. All 82 samples of the training set could be run and achieved a SAT result. This yielded a prediction accuracy of 66% on the test set. Though not unexpected, this is well below the goal of meeting or exceeding the Markey results.

4.3.3 Two Layer Software Neural Network Tests on the Deception Detection Dataset

Configuration:

- **Hidden Layer:** Experiments ranging from 2 to 39 nodes with threshold activations
- **Constraints:** Weights and biases not constrained

The next step was to benchmark the problem with a software neural net trained using gradient decent. The software model was implemented in python with the MLPClassifier from sklearn.neural_network and included threshold activation between the hidden layer and output layer. In addition to setting a baseline for performance, this enabled me to examine the effect of different network topologies on the prediction performance. As Table 3 shows, the software implementation with just one hidden layer achieved a 77% accuracy with 4 nodes in the hidden layer. This became the benchmark for a typical 2-layer network trained using gradient decent. Since there was no increase in predictive performance with more nodes in the hidden layer, a 2 layer network with 4 nodes was chosen for all tests going forward.

Number of Nodes in Hidden Layer	Accuracy on Test Set
2	72%
3	69%
4	77%
5	73%
7	72%
39	77%

Table 3: Effect of Different Numbers of Nodes in the Hidden Layer on Prediction Accuracy for a 2 Layer Network Modeled with Software

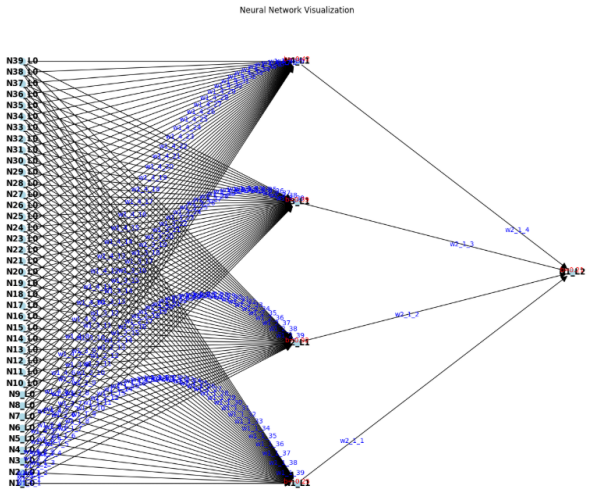


Figure 8: Topology of 2-Layer, 4 Node Network with 39 Features

4.3.4 Two Layer Solver Neural Network Tests on the Deception Detection Dataset

Several experiments were conducted using the 2-layer, 4-node configuration described above. Figure 8 illustrates the network topology.

1. SMT Solver With Threshold Activation

   Configuration:

   - **Hidden Layer:** 4 nodes with threshold activations

   A two-layer network with threshold activation achieved 100% training accuracy on an unlimited number of samples in the training set, and it converged to SAT very quickly. However, when validated against the test set, it exhibited poor performance. The best it was able to achieve was 63% accuracy. Examining the cvc5 output revealed that many of the weights in the second to fourth nodes were unused. This suggests that the solver could easily find a solution among the separable decision planes that the activation functions enabled, indicating that a SAT solver is a poor choice for a 2-layer network with threshold activations because it is so susceptible to significant overfitting.

2. OMT Solver With Threshold Activation

   For this experiment, I used the same configuration as above. The OminMathSAT solver achieved better results than cvc5, except in the case of unlimited samples. It achieved a high accuracy of 70% with 60 samples, however the results deteriorated to 53% with the maximum of 82 samples, likely the case of significant overfitting.

3. SMT Solver With Linear Activation

Configuration:

- **Hidden Layer:** 4 nodes with linear activations

While the results with threshold activations did not yield satisfactory results, I decided to eliminate the activations and just hook up the output of the hidden layer directly to the output layer. While this eliminates the separate decision planes that threshold activations provide, it does create a more nuanced linear decision surface so the solver might find a better solution than the single layer case.

In this experiment, cvc5 was returning unsat once I exceeded 40 samples, suggesting that the constraints become too difficult to satisfy jointly. This situation indicates that some samples are "outliers" as in the previous experiments relative to a potential pattern of weights that would otherwise satisfy a large subset of the training data.

The final results are shown in Table 4. Note that the highest performance is 73% accuracy at both 20 and 40 training samples, which exceeds the benchmark of 71% for the Markey study. The accuracy of results is fairly unpredictable due to the solver reporting SAT once it finds a solution.

Number of Samples	Accuracy on Test Set
10	64%
20	73%
30	64%
40 (SAT limit)	73%

Table 4: Accuracy of 2-Layer, 4-Node Configuration with SMT Solver and Linear Activation

4. OMT Solver With Linear Activation and Slack Variables

Number of Samples	Accuracy with 39 Features	Accuracy with 86 Features
10	73%	66%
20	72%	63%
30	64%	66%
40	61%	70%
50	61%	64%
60	72%	64%
70	did not converge	72%
Unlimited	did not converge	67%

Table 5: Accuracy of 2-Layer, 4-Node Configuration with OMT Solver and Linear Activation. Two Configurations with Different Number of Features in the Training Data Are Presented.

The final experiments were performed using OptiMathSAT with slack variables. Given that the SMT solver was unable to push past 40 samples due to outlier data, it was felt that the optimizer might be able to identify which samples were causing problems and perhaps remove them.

The initial experiment was conducted using the dataset with 39 features as in the previous SMT solver experiment. As indicated in Table 5, it achieved a prediction accuracy of 73% on 10 samples. Subsequent runs did not yield improvements. Interestingly OptiMath SAT was able to converge up to 60 samples without having to use the slack variables. Above 60 samples I could not get it to converge after 12 hours.

Configuration:

- **Input Features:** 86 real number inputs, ranging from -4.0 to +4.0.

I thought I'd try one last experiment with the full 86 emotional cue features which are also presented in Table 5. The hypothesis here was that more features provided more avenues to reach SAT. While I was able to converge to SAT with all samples, the final result was disappointing. The fact that the best result was on 10 samples and 39 cues was a bit of a head scratcher.

5 Unsuccessful Experiments

Despite promising results in some experiments, several configurations failed to produce satisfactory outcomes. Below are the notable unsuccessful experiments:

5.1 Optimizing Weights and Biases by Minimizing the Loss Function

It occurred to me that the capability of an OMT solver to find an optimum solution may provide an avenue to train an optimum set of weights and biases, not just any set of weights and biases that can satisfy the constraints. In machine learning, gradient descent is used to find optimal weights and biases by computing gradients (partial derivatives of the loss function with respect to the weights and biases) during backpropagation, which provides the direction and magnitude of weight updates.

So I built a version of the smt-lib file generator that derived a "hinge loss" on each sample. It works like this:

$$\text{error} = \begin{cases} -\min(\text{output}, 0) & \text{if label} = 1 \\ \max(\text{output}, 0) & \text{if label} = 0 \end{cases} \tag{1}$$

Then I set the solver the task of minimizing the sum of all the losses, thus deriving the optimum weights and biases (or so I thought). I verified it on the xor gate, so I knew it worked. However, even on the simplest networks using the deception detection dataset, it was not able to converge. OptiMathSAT started crashing, and so I moved over to z3 and was able to confirm it worked on xor, but didn't converge on the larger datasets. I believe this is a promising direction deserving of further investigation, but I've run out of time to be able to investigate further.

5.2 Integer-Based Encoding

An initial approach used Quantifier-Free Nonlinear Integer Arithmetic (QF_NIA) to encode weights and biases. The hypothesis was that integer encoding would simplify the problem and reduce computational complexity. After converting the training and test sets to integers, this approach significantly limited the solver's flexibility and granularity, leading to poor performance. The test accuracy for these configurations failed to exceed 64% on the deception detection dataset, making them unsuitable for practical applications.

5.3 Over-Parameterized Networks

Networks with excessively large architectures, such as 100 hidden nodes per layer, were tested to assess the solver's scalability. These configurations resulted in unsatisfiable constraints or prohibitively long solver runtimes. The increased complexity overwhelmed the SMT solver, demonstrating its current limitations in handling over-parameterized models.

5.4 Alternative Activation Functions

Experiments with sigmoid activation functions were conducted to explore smoother decision boundaries. While these activations provided a more continuous transition between states, they introduced additional non-linear constraints, significantly increasing solver runtimes without improving generalization. This suggests that the added complexity of these activations outweighed their potential benefits.

5.5 Other Solvers

z3, Gurobi and Vampire were also installed and tested against some of the experimental setups. z3 failed to converge so often that it was discarded. After looking more closely into Vampire, it was deemed inappropriate. Gurobi complained that the problems were non-linear because in multi-layer networks two variables are being multiplied together. Apparently it does not handle non-linear models.

6 Analysis and Discussion

6.1 Strengths

The SMT solver's ability to rigorously satisfy constraints ensures that trained models adhere strictly to the encoded rules on the training set. This property can expose which features are most important

for predicting labels in the training set. This property may also be valuable for applications requiring interpretability and strict compliance with specifications. Unlike gradient-based methods, which may converge to suboptimal solutions due to local minima, the SMT solver provides deterministic solutions, guaranteeing a globally satisfiable model for the given constraints.

The investigation proved that an OMT solver can be used to identify "noisy" samples in the training set. Through the use of slack variables, OMT solvers might be useful to help prune small, but complex training data. Also, it appears single layer neural networks trained using OptiMathSAT can compete with conventional classifiers employing linear regression.

6.2 Limitations

Despite its strengths, the SMT- and OMT-based approaches face significant challenges. Overfitting is a primary concern, as the solver's exact solutions often fail to generalize to unseen data. Because the solvers are looking for any solution that satisfies constraints, results are unpredictable. Perhaps an optimization technique that employs minimization of total loss might prove more successful.

Scalability is another limitation, with solver performance degrading as dataset size or network complexity increases. Furthermore, the ability of threshold activation functions to expand the decision surface actually increases to solvers propensity to converge on non-optimal solutions, while linear activations, though more flexible, do not model complex decision surfaces.

6.3 Comparison with Gradient-Based Methods

Traditional neural network classifiers trained with gradient descent consistently outperformed SMT- and OMT-trained models, with 77% accuracy on the deception detection test set. The best performance for the solvers was 73% for weights and biases trained using OptiMathSAT. However, OMT solvers were able to identify noise in the training data which may provide some insights.

7 Conclusions and Future Work

The research demonstrates that SMT- and OMT-solvers can train simple neural networks, providing a proof of concept for logic-driven training methods. However, the approach is limited by overfitting, scalability challenges, and the expressiveness of activation functions.

Future work should explore advanced activation functions, such as ReLU or softmax, within SMT frameworks. Also, there is likely some potential to use solvers to find optimal solutions through techniques that minimize loss. Perhaps better formed constraints, more computing power or more powerful models might yield a better result. Comparing alternative solvers and integrating solver-guided synthesis for feature selection and network design could further enhance performance.

8 Acknowledgments

I would like to thank Dr. Alan Hu, Professor in the CS Department at U.B.C., for inspiration, feedback and guidance, and to acknowledge my fellow CPSC513 students for putting up with me.

References

[1] Loan Approval Prediction Dataset. Kaggle (2023). Retrieved from: https://www.kaggle.com/datasets/architsharma01/loan-approval-prediction-dataset.

[2] Harpster, T., Adams, S. H., and Jarvis, J. P. (2009). Analyzing 911 homicide calls for indicators of guilt or innocence: An exploratory analysis. *Homicide Studies*, 13(1):69–93.

[3] Markey, P. M., Feeney, E., Berry, B., Hopkins, L., & Creedon, I. (2022). Deception Cues During High-Risk Situations: 911 Homicide Calls. *Psychological Science, 33(7), 1040-1047.*

[4] Roberto Sebastiani and Patrick Trentin, *OptiMathSAT: A Tool for Optimization Modulo Theories*, Journal of Automated Reasoning, vol. 64, no. 3, pp. 423–460, 2020. doi: 10.1007/s10817-018-09508-6.

[5] Holzbaur, C., Menezes, F., Barahona, P. (1996). Barbosa, H. et al. (2022) Defeasibility in CLP through generalized slack variables. In: Freuder, E.C. (eds) Principles and Practice of Constraint Programming — CP96. CP 1996. Lecture Notes in Computer Science, vol 1118. Springer, Berlin, Heidelberg. https://doi.org/10.1007/3-540-61551-2_76.

[6] George B. Dantzig, *Linear Programming and Extensions*, Princeton University Press, 1963.

[7] Barbosa, H. et al. (2022). cvc5: A Versatile and Industrial-Strength SMT Solver. In: Fisman, D., Rosu, G. (eds) Tools and Algorithms for the Construction and Analysis of Systems. TACAS 2022. Lecture Notes in Computer Science, vol 13243. Springer, Cham. https://doi.org/10.1007/978-3-030-99524-9_24