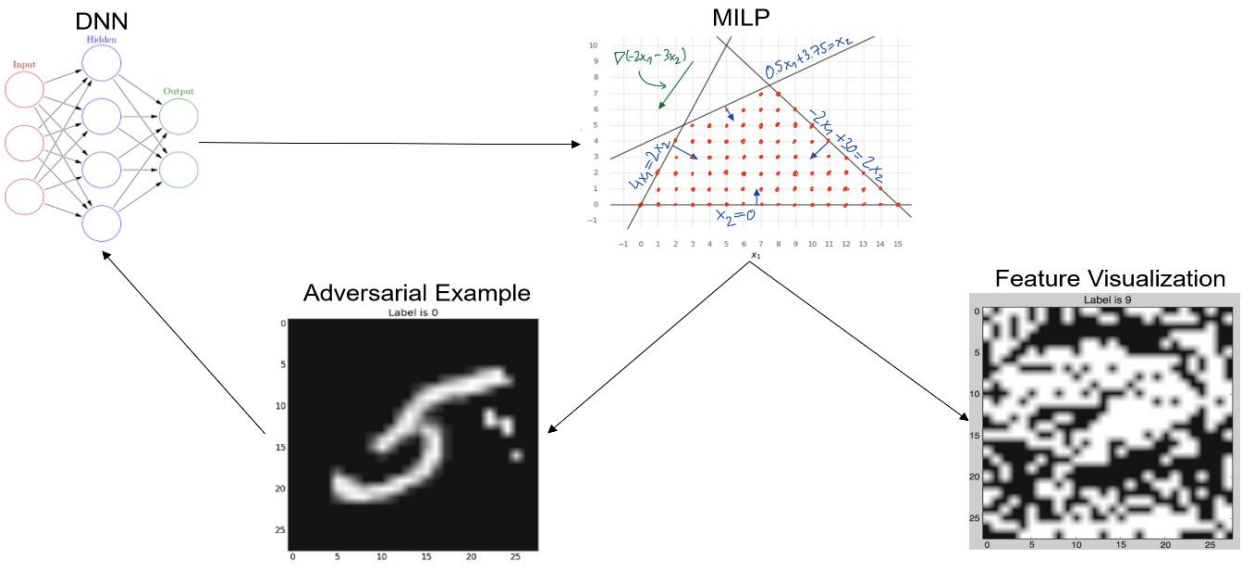
Deep Neural Network & Mixed-Integer Linear Programming

Anuraag Mishra



Abstract: This paper explores Deep Neural Networks (DNNs) and Mixed Integer Linear Optimization (MILP). It aims to consolidate essential information about DNNs, including activation functions and MILP, focusing on Branch-n-Bound and Bound-Tightening techniques. The paper also covers MILP's application in deep learning, particularly in feature visualization and adversarial examples. A basic understanding of linear inequalities and neural networks is beneficial.

I. DNNs- A Brief Overview:

Concept: DNNs are inspired by the human brain, functioning through interconnected neurons to process information and learn.

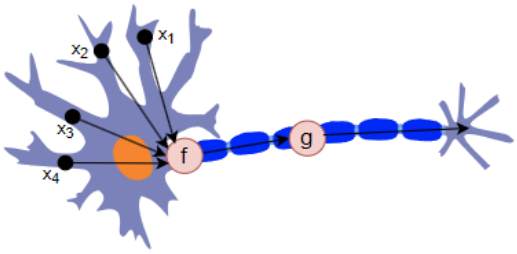


Figure A: Neuron

Artificial Neuron: Inputs received by a neuron are weighed and summed. An activation function, analogous to a neuron's threshold, is applied to this sum to generate an output, which is passed to subsequent neurons.

This linear function can be written as:

$$f(x) = w^T x + b$$

$$y = g(f(x))$$

$$y = g(w^T x + b)$$

where $g(f)$ is the activation function, w^T is transpose of the vector w which to represent weights for each input, x represents input variables, and b is a constant bias.

$$x = [x_1, x_2, x_3]$$

$$w = [w_1, w_2, w_3]$$

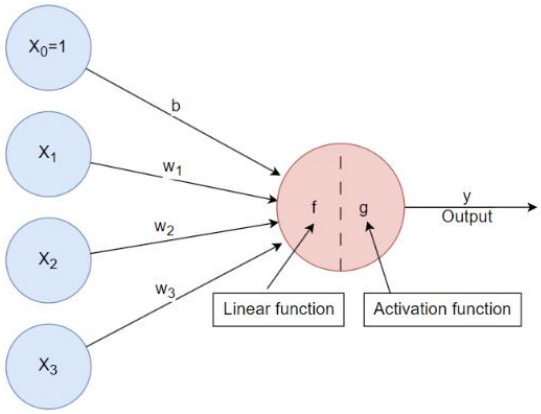


Figure B: Artificial Neuron

Neural Layer

This network can be extended to having more than one neuron in the same layer. That means the same input can contribute differently to different neurons based on the unique weightage applied on each directed node. Here the definition of *g* and *x* remains the same, but the definition of *w* and *b* changes due to the increase in the number of branches.

$$w = \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \end{bmatrix}$$

$$b = [b_1, b_2, b_3]$$

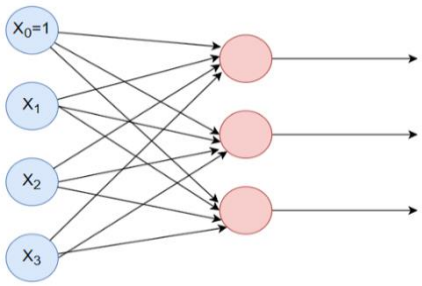


Figure C: Single Layer of 3 Neurons

(Deep) Neural Network

This idea can be further raised by having multiple neural networks connected. Each hidden neural layer's output works as input to the next layer.

For any inner layer k, the output can be defined as:

$$x^k = g((W^k)^T x^{k-1} + b^k)$$

where g is the activation function, W^T is transpose of W, m is the number of neurons in the hidden layer-k, n is the number of inputs to the hidden layer-k.

$$x^k = [x_1^k, x_2^k, \ldots . x_n^k]$$

$$w^k = \begin{bmatrix} w_{11}^k & \cdots & w_{1n}^k \\ \vdots & \ddots & \vdots \\ w_{m1}^k & \cdots & w_{mn}^k \end{bmatrix}$$

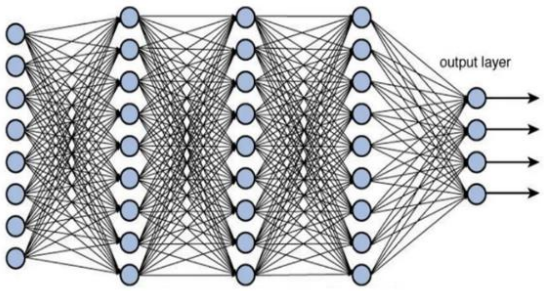


Figure D: Deep Neural Network

Such a deep network of neural layers is called Deep Neural Network (DNN). In today's world, such models set the base of how many complex DNNs are created, which are trained over a dataset, to learn features hidden inside the data and try to predict the output correctly.

Activation Function

Activation functions enable artificial neurons to mimic brain neurons, crucial for neural networks to understand non-linear patterns. This capability significantly enhances their ability to model complex data. Our focus for this paper is on ReLU, a popular non-linear function defined as :

$$ReLU(x) = max(x, 0)$$

The Output of the ReLU function is x if x is positive else zero. The entire layer can be represented with ReLU as an activation function:

$$x^k = ReLU((w^k)^T x^{k-1} + b^k)$$

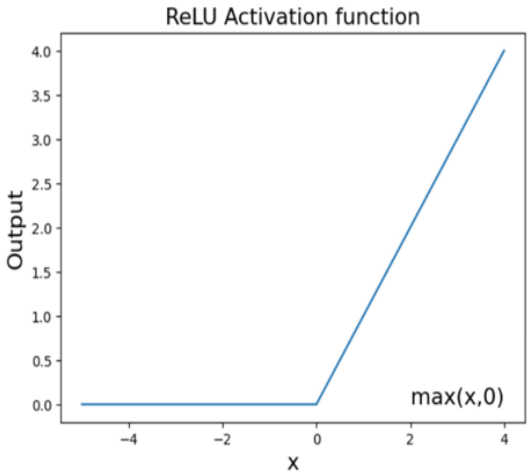


Figure E: ReLU vs x

II. MILP

What is MILP?

Mixed-integer linear Programming (MILP) is a mathematical method used to optimize problems with both integers as well as continuous variables. A special case of MILP is 0-1MILP or binary MILP, which deals with decision variables restricted to values of 0 or 1. The problem involves determining the optimal values of these variables based on linear constraints and an objective function.

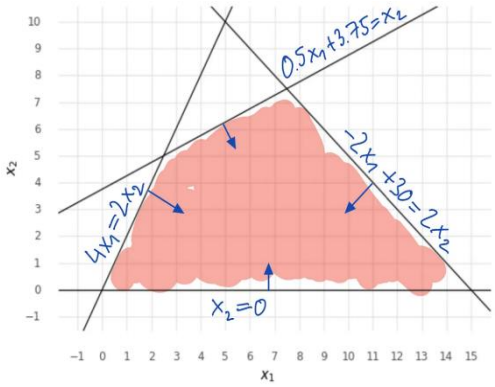


Figure F: Bounding region for the given constraints. Source: 4

In the provided graph, there is an area defined by two decision variables, x_1 and x_2, along with four linear constraints that bound the colored region. The objective is to minimize the objective function, such as $-2x_1 - 3x_2$ while considering only integer solution for x_1. MILP is an essential component of linear programming since real-world scenarios often involve integer values. To get a good overview about MILP we will try to get an overview of two concepts:

- Branch-and-Bound algorithm
- Bound-Tightening.

Branch and Bound (BnB) Algorithm

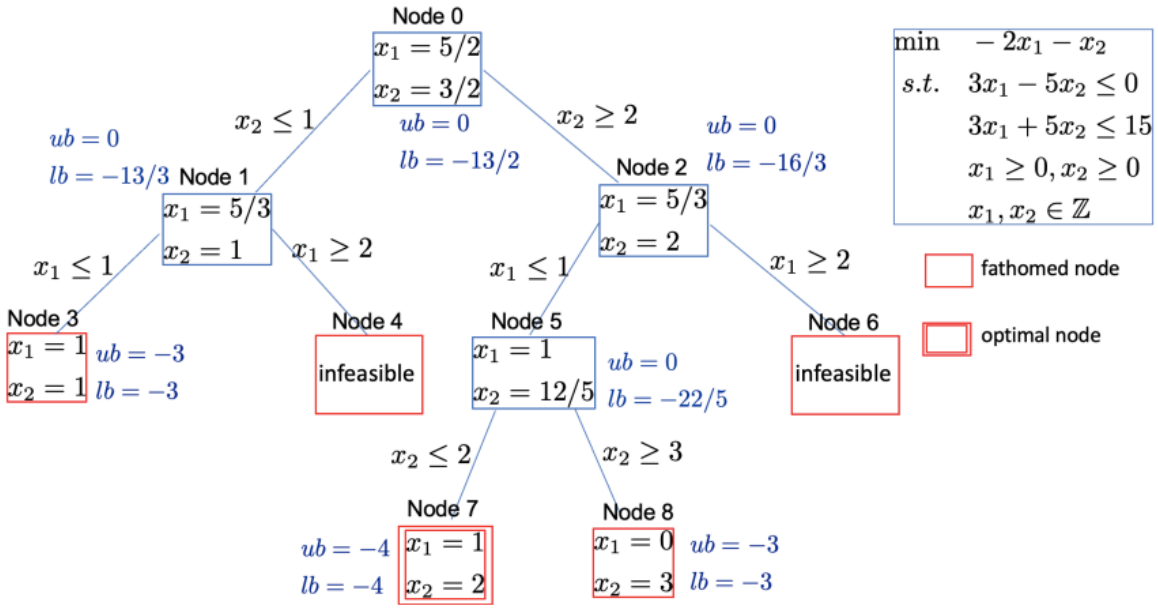


Figure G: A minimization problem of two decision variables x_1, x_2, Source: 2

The BnB algorithm is a systematic approach to solving MILP problems by partitioning the original problem into smaller subproblems, solving these, and then verifying if their solutions satisfy the original problem's constraints.

The process begins with an **LP relaxation**, where the integer constraints on the variables are temporarily removed, allowing the entire solution space to be considered. From the root node, the

algorithm progresses by setting a decision variable to a specific value, and then solving the linear constraints to find the optimal values for the remaining variables. If this results in non-integer values, two new subproblems are generated by branching—rounding the variable's value up and down. In the above image, the first two subproblems created are $x_2 \leq 1$ & $x_2 \geq 2$. Many MIP-solvers used various techniques to pick an optimal starting value for the decision variable, as this makes sure that the solution is realized earlier with less branching.

Subsequent branches are created at any node where the decision variables have not yet attained integer values, with the exception of *fathomed nodes*—those that cannot yield a better solution or are infeasible.

At each node, the solver can keep track of the **Bounds** (*upper bound (ub)* & *lower bound (lb)).* Upper bound is the best (lowest in case of minimization) objective function value for the problem at a given node. And lower bound is obtained by solving the linear constraint at a node with LP-relaxation.

Fathomed Nodes are identified and pruned from further branching (pruning) or as an optimal solution if the following conditions are met:

- The node is infeasible (breaks the original constraint with values realized under the subproblem for the decision variables).
- The solution found is worse than the best-known solution.
- Lb is higher than ub.

The **optimal solution** is when all nodes have been either explored or pruned and the best feasible solution found is the global optimum. Unlike other heuristic algorithms which may get stuck at local optima, BnB always finds the optimal solution given enough time to run (NP-hard).

Bound Tightening

Bound Tightening (also known as Preprocessing), is a technique used by many to reduce the solution space of the MIP before even applying Branch-n-Bound. The idea is to find even more tighter bounds (ub & lb) for the decision variables with the help of the existing conditions. Having knowledge of reduced solutions space for the decision variables can significantly reduce the runtime of the BnB algorithm as it is an NP-hard problem. The impact of bound tightening on the BnB algorithm is much more pronounced with an increase in the number of decision variables (as solution space is in a higher dimension). We will observe the impact during the analysis part of this paper.

III. Formulate DNN for MILP

In formulating a DNN for MILP, ReLU is linearized (Equation 3). This delineates ReLU's positive and negative parts. However, this formulation does not guarantee uniqueness due to the possibility of adding a scalar δ ≥ 0, yielding multiple feasible solutions $(\boldsymbol{x} + \boldsymbol{\delta}, \boldsymbol{s} + \boldsymbol{\delta})$. To ensure uniqueness, binary constraints (Equations 4a, 4b, 4c) are introduced, making this a 0-1 MILP problem. These constraints enforce that either x or s is zero, aligning with ReLU's nature.

$$x^k = ReLU\left(\left(w^k\right)^T x^{k-1} + b^k\right) \quad (1)$$

$$ReLU(y) = max(y, 0) \quad (2)$$

$$w^T y + b = x - s,\ x \geq 0,\ s \geq 0 \quad (3)$$

$$z = 1 \rightarrow x \leq 0 \quad (4a)$$

$$z = 0 \rightarrow s \leq 0 \quad (4b)$$

$$z \in \{0, 1\} \quad (4c)$$

This binary condition is what makes this DNN problem a 0-1MILP problem. This z variable can be introduced for each neuron for a layer-k. This can be further extended to a generalized formulation with respect to a layer and each neuron within a layer with equation 5, where $k(hidden\ layer) = 1, \ldots, K-1$ & $j\ (neuron\ at\ layer\ k) = 1, \ldots, n_k$.

$$\sum_{i=1}^{n_{k-1}} w_{ij}^k\, x_i^{k-1} + b_j^k = x_j^k - s_j^k, \quad (5a)$$

$$x_j^k \geq 0, s_j^k \geq 0$$

$$z_j^k = 1 \rightarrow x_j^k \leq 0 \quad (5b)$$

$$z_j^k = 0 \rightarrow s_j^k \leq 0 \quad (5c)$$

$$z_j^k \in \{0, 1\} \quad (5d)$$

The equation 6a is the lb and ub of the initial input from the actual data. These bounds are based on the domain (remain fixed), for example, if our input is a grayscale image with normalized pixel values ranging from 0(lb) to 1(ub. Whereas, for the internal layer, the lb and ub can change based on the objective function for each layer. Bound-tightening techniques can be used to realize these bounds more strictly. To begin with, we can set the $lb = 0$ (because of ReLU) & $ub \in R_+ \cup \{+\infty\}$ for any layer k as described in equations 6d & 6e.

$$lb_j^0 \leq x_j^0 \leq ub_j^0 \quad (6a)$$

$$lb_j^k \leq x_j^k \leq ub_j^k \quad (6b)$$

$$\overline{lb_j^k} \leq s_j^k \leq \overline{ub_j^k} \quad (6c)$$

$$lb_j^k = \overline{lb_j^k} = 0,\ k \geq 1 \quad (6d)$$

$$ub_j^k,\ \overline{ub_j^k} \in R_+ \cup \{+\infty\},\ k \geq 1 \quad (6e)$$

Furthermore, we can introduce an objective function at each layer for the MILP solver to optimize (Eq. 7). c_j^k and γ_j^k can be defined according to the specific problem at hand, based on the activate and inactivate neuron (where $z = 1$).

$$\sum_{k=0}^{K} \sum_{j=1}^{n_k} c_j^k x_j^k + \sum_{k=0}^{K} \sum_{j=1}^{n_k} \gamma_j^k z_j^k \quad (7)$$

IV. Application

To see the above in action, we can take a case where a DNN model has learned to recognize a hand-written digit from the input (a grayscale image with each pixel as input. Each pixel value is normalized in the range of [0,1], where $1 \rightarrow white,\ 0 \rightarrow black$) with an accuracy of at least 90%. An image can have a single digit between the range of 0-9.

The output layer of such a DNN model to predict 10 possible digits, has 10 neurons. The neuron with the highest activation value in the final layer is considered the prediction made by the DNN model. Let's Explore two applications for this DNN setup.

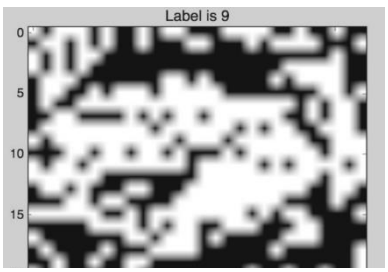


Figure H: All the pixels (white) that contribute to predicting an MNIST image as 9 for the trained DNN model.

Feature Visualization: A MIP solver can be used to find the input (pixel) value that maximum activation x_j^k of each unit for each digit. This leads to an interesting result, adjacent is an image that shows which pixels the DNN thinks are important to predict the digit 9. We can observe that there is no nice visual

pattern that can be recognized, something that even closely resembles the digit.

Adversarial Examples: The most intriguing use case of the MIP solvers can be used for generating newer examples(images) which can be used to exploit the hidden weaknesses of the DNN model. The key idea is to use the understanding of the importance of each pixel for different digits & modify the image slightly so that it still visibly looks like the original image, but the DNN model predicts something completely different. Given original input $\tilde{x}^0$ which DNN correctly classifies $\tilde{l}$. Find x^0 (close to $\tilde{x}^0$) which is classified as l ($\neq \tilde{l}$).

$$l = (l + A) mod\ 10 \qquad (8)$$

The images above are achieved by using the images for digits 4 & 5 and then setting A = 5. We can observe that the small pixels added (white) at the bottom (digit 4 image), and in the middle of the image were enough for DNN to recognize this image as 9. To make these changes less apparent, we can further add another constraint on the input value change:

$$x^K_{d+1} \geq 1.2\ x^K_{j+1},\ j \in \{0, \dots, 9\}\{d\} \qquad (9)$$

The condition is set such that pixel changes throughout the image are not more than 20% of the original value. We can further add an L1 norm constraint to keep the images visibly similar:

$$-d_j \leq x^o_j - \tilde{x}^0_j \leq d_j,\ \sum_{j=1}^{n_0} d_j \qquad (10)$$

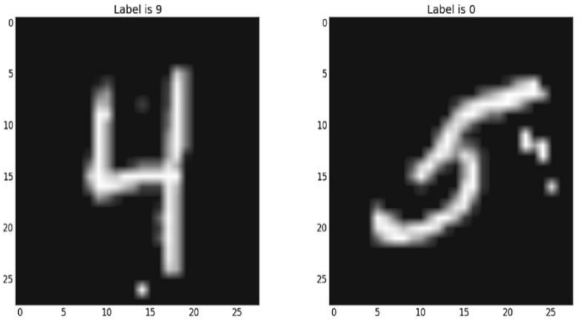


Figure I: Adversarial examples. Predicting image on the left as 9 & on the right as 0.

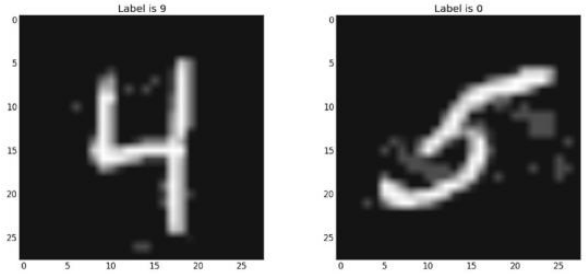


Figure J: Adversarial examples with 20% change. Predicting image on the left as 9 & on the right as 0.

Due to the flexibility of the MIP solvers, all we need to do is just add equation 8,9,10 to the previous equation within the MIP solver. The Adversarial examples have a strong case in the real world, as they can be used to find weaknesses of the DNN networks, create enough of such examples that DNN can be trained on them, and improve its model. Such examples can also be used for human verification on websites in this current age of advanced image recognition models. Newer examples can be created to fool such models.

V. Result & Analysis

MILP is well known to be NP-hard (Non-deterministic Polynomial-time hard). Therefore, the creation of these adversarial examples can take a lot of time given how complex the DNN models are. Therefore, to further illustrate the importance of bound tightening for MIP solvers, the results from a research paper by the author Fischetti on the topic 'Deep neural networks and mixed integer linear optimization' are used to illustrate.

5 different DNNs were used, on each DNN two different models of MILP were used (Basic model & Improved Model). Both models had the same equations mentioned throughout this paper, except that on the improved model, the bounds of x & s for each layer (equation 5) were improved by applying bound tightening (Preprocessing). This needs to be done only once for each DNN once model is trained. The following table is an observation made within the research paper after creating 100 instances of adversarial examples with a time limit of 300 seconds per instance with each model (Basic & Improved).

	Basic model				Improved model			
	%solved	%gap	Nodes	Time (s)	%solved	%gap	Nodes	Time (s)
DNN1	100	0.0	1,903	1.0	100	0.0	552	0.6
DNN2	97	0.2	77,878	48.2	100	0.0	11,851	7.5
DNN3	64	11.6	228,632	158.5	100	0.0	20,309	12.1
DNN4	24	38.1	282,694	263.0	98	0.7	68,563	43.9
DNN5	7	71.8	193,725	290.9	67	11.4	76,714	171.1

Figure K: Table 1 from source 1. Where Nodes - Avg. no. of branching nodes, Time(s) - Computing time on avg. (all instances), %gap - percentage gap between the best ub and the best lb, %solved - percentage of solved instances.

DNN	Neurons with Hidden Layer(s)
DNN1	8+8+8
DNN2	8+8+8+8+8+8
DNN3	20+10+8+8
DNN4	20+10+8+8+8
DNN5	20+20+10+10+10

5 DNNs were used to test the two models. Each DNN described above is by a number of neurons at each hidden layer (separated by a + sign) with ReLU activation. Each DNN further had one input layer for pixels of the image and one more output layer formed with 10 neurons each for 10 digits.

Observation: The Basic model is only able to solve all 100 instances of adversarial examples. As the models get more complex, the average time increases(column: *Time(s)*) as more and more instances remain unsolved. At DNN5, the basic model is only able to solve 7 out of 100 with an average time of 290.9 secs.

The improved model, on the other hand, finds all 100 instances for the DNN1 DNN2 & DNN3. And only struggles a little with DNN4(98%) & DNN5(67%). The time is also significantly lower for the fact that number of nodes that the MIP solvers had to explore due to the preprocessing (bound tightening) technique. The preprocessing time is not included in the column Time(s) for the Improved model.

Preprocessing can also take a lot of time. For the improved model, the time limit for each variable was set to 300 seconds. If bound tightening process takes full 300 seconds to find bounds, then the preprocessing can itself take several hours. We can reduce this time by only allowing the solvers to find bounds with only 1 second cutoff. This significantly reduces the preprocessing time at the cost of solver only able to find weaker bounds for the variables. If we compare Improved model with exact bound (300 seconds cutoff for bound tightening) vs Weak bounds (1 second), the weaker bound model still performance significantly close to the Improved with exact bound. It is only at DNN4 & DNN5 we see some difference.

	Improved model									
	Exact bounds					Weaker bounds				
	t.pre.	%sol.	%gap	Nodes	Time (s)	t.pre.	%sol.	%gap	Nodes	Time (s)
DNN4	1,112.1	98	0.7	68,563	43.9	69.4	98	0.4	80,180	45.5
DNN5	4,913.1	67	11.4	76,714	171.1	72.6	57	16.9	84,328	185.0

Figure L: Table 2 from source 1

VI. Conclusion & Future Work

MILP can find the best solution given that any problem can be described as LP with high precision. This makes them versatile. They are also favoured over greedy approaches due to their robustness to getting stuck at local/suboptimal solutions. MILP has a strong case to be used in real-world examples. And with the adversarial examples, they make a compelling case to be used to optimize the DNN models.

MILP does have some inherent challenges that make it difficult to use everything:

- Complexity: Can be computationally intensive, especially as the number of variables and constraints increases.
- Solving Techniques: Techniques like branch-and-bound does not perform well or take too much time when compared to approaches like Stochastic gradient descent for complex DNN.

Nevertheless, these challenges are the potential future MILP research areas. Furthermore, the MIP model is not suitable for training (learning w), as the problem will become bilinear (learning both weights/biases and inputs), but good for optimization tasks like feature visualization and adversarial example generation to learn about the DNN models.

For small DNNs, MIP can be solved to proven optimality in a matter of seconds on a standard notebook. However, for larger (realistic) DNNs the computing time can become too large. DNNs of size (30, 20, 10, 10, 10, 8, 8, 8) or (50, 50, 50, 20, 8, 8) lead to computing times of one hour or more. In such cases, it is best to resort to heuristics or use a restricted version of the model itself.

A work-in-progress implementation of the research paper at github.com/mishraanuraagx/DNN_MILP.

Future Work:

- Reduction of the computational effort involved in the exact solution of the model.
- Find new heuristic methods for building adversarial examples for large DNNs.
- Find new application of MILP in DNN.

VII. References

1. Fischetti, M., & Jo, J. (2017). Deep Neural Networks as 0-1 Mixed Integer Linear Programs: A Feasibility Study. arXiv. arXiv:1712.06174. Available at https://arxiv.org/abs/1712.06174
2. Huang, L., Chen, X., Huo, W., Wang, J., Zhang, F., Bai, B., & Shi, L. (2021). Branch and Bound in Mixed Integer Linear Programming Problems: A Survey of Techniques and Trends. arXiv. arXiv:2111.06257. Available at https://arxiv.org/abs/2111.06257
3. Grimstad, B., & Andersson, H. (2019). ReLU networks as surrogate models in mixed-integer linear programs. arXiv. arXiv:1907.03140. Available at https://arxiv.org/abs/1907.03140
4. Módos, István. “Mixed-Integer Linear Programming: Formal Definition and Solution Space." Towards Data Science, Feb 21, 2023, https://towardsdatascience.com/mixed-integer-linear-programming-formal-definition-and-solution-space-6b3286d54892. Accessed 10 Jan. 2024.