Neuro-Symbolic Sudoku Solver

Ashutosh Hathidara[1], Lalit Pandey[1], and Dr David Leake[1]

Indiana University Bloomington, USA {ashuhath,lpandey,leake}@iu.edu

Abstract. Presently, Deep Neural Networks have achieved great success in some of the complex tasks that humans can do with ease. These include image recognition/classification, natural language processing, game playing etc. However, modern Neural Networks fail or perform poorly when trained on tasks (like list sorting, playing Sudoku, finding shortest path etc.) that can be solved easily using backtracking and traditional algorithms. Therefore, we use the architecture of a Neuro Logic Machine (NLM) ([1] Honghua et al., 2019) and extend its functionality to solve a Sudoku of 9X9 grid. To expand the application of NLM, we generate a random grid from the dataset and assign empty cells in it which need to be filled. The goal of this experiment is to find a target value ranging from 1-9 and fill the empty cells of the sudoku grid while maintaining a valid configuration. In our study, we showcase that NLM can achieve 100% accuracy for solving a Sudoku with empty cells ranging from 3 to 10. One of our goals is to demonstrate that NLM's can also be used for solving complex problems and games like Sudoku. We also analyse the behaviour of NLM with backtracking algorithm by comparing the convergence time using a graph plot on the same problem. With this study we show that Neural Logic Machines can be trained on the tasks that traditional Deep Learning architectures fail using Reinforcement Learning. We also aim to propose the importance of symbolic learning in explaining the systematicity in the hybrid model of NLM.

Keywords: Neural Logic Machines · Symbolic Learning · Neuro Symbolic · Sudoku Solver · Neural Networks · Deep Reinforcement Learning.

1 Introduction

The groundbreaking results from the modern deep learning models have proved that they are the ideal tools to solve complex problems but the lack of systematicity in these models have been a problem for a long time. Recent research has focussed on this issue and has come up with hybrid models of Neural Networks with symbolic learning. By testing one such model called Neural Logic Machine ([1] Honghua et al., 2019), we emphasise on the relevance of symbolic learning in solving complex problems that modern deep learning methods fail to converge for. More specifically, we validate the NLM model on a different mathematical problem to realize their true potential and to analyse their performance. The important aspect to focus here is that NLM is capable of utilizing the knowledge gained from generated rules to achieve a perfect generalization in a number of

tasks. Therefore, we also aim to test NLM's ability to recover the lifted rules and apply them in the later stages of curriculum learning when the complexity of the problem rises. For this, we gradually change the number of empty cells in the grid while training.

We test the architecture of Neural Logic Machines ([1] Honghua et al., 2019) for solving a complex puzzle called Sudoku using our own set of predicates as input. In this experiment, we closely analyse the performance of this model and compare it with traditional algorithms on the same problem. To perform this experiment, our three main tasks are: first, we train the NLM on sudoku grids with pre-defined empty cells which need to be filled. The number of empty cells are increased as the training progresses. This approach is called curriculum learning where the complexity of the problem increases as the model learns successfully. Second, we use symbolic learning with reinforcement rewards to award the model every time a valid configuration of the empty cell is achieved. Finally, the convergence time of NLM and Backtracking is compared using a graph plot. Towards the end of the experiment, the trained model is tested with a random sudoku grid and a fully filled solved sudoku is obtained.

In the later sections, we elaborate upon the robustness and systematicity of the network layers.

1.1 Key Contributions

Our major contributions in this paper are:

Extending Application of Neural Logic Machine: The NLM is trained and tested on a completely different problem set (Sudoku puzzles) to expand its scope in wide areas of applications. Instead of function approximators, the reinforcement training algorithm 'Reinforce' is used to estimate the gradients using the policy-gradient mechanism and calculates the gradients in a non-differentiable trainable setting.

Time Complexity Comparison with Backtracking: Upon successful implementation of NLM on a 9X9 Sudoku grid, their convergence time is compared with Backtracking algorithm and demonstrated using a graphical representation. A thorough comparison of NLM with backtracking is also mentioned in the result section.

Testing Robustness of Neural Logic Machine: While the NLM ([1] Honghua et al., 2019) is tested with tasks like list sorting, path finding and BlocksWorld game, we have used a more complex problem of solving a sudoku puzzle of a 9X9 grid with a number of empty cells ranging from 3 to 10 to check for the robustness of the model.

2 Related work

The rising demand for training the Neural Networks for performing complex tasks has caught great attention among the researchers. However, their lack of systematicity and inability to generalize for a greater set of inputs leads them to perform poorly on systematic tasks. To address these challenges, [1] Honghua et al., 2019 proposed Neural Logic Machines which can solve problems that require systems to perform actions by following systematic set of rules. In [1], NLM utilizes the relationships of objects obtained from quantifiers and logic predicates to solve BlocksWorld, Sorting and PathFinding tasks. The study done in [1] Honghua et al., 2019 contrasts between a conventional RNN (Recurrent Neural Network) with the proposed NLM. It states about the training of RNN on a smaller list and its failure to sort a slightly larger list during the testing phase.

An alternate approach to function approximators has been used with NLM called the REINFORCE algorithm (Richard et al., [2]). It is used for policy gradient optimization and estimates the gradient with Monte-Carlo method. This is usually used in deep reinforcement learning where the action is sampled and the neural network can not backpropagate. In such a non-differentiable setting, we use gradient estimation.

Wang et al., 2019 [3] has proposed a novel architecture called SATNet which is a differentiable maximum satisfiability solver. It is an approximate-differentiable solver which works on a fast coordinate descent approach for solving semidefinite programs (SDP) associated with the Maximum Satisfiability problem. SATNet has complete deep learning architecture utilizing CNN whereas the NLM approach proposed in this study is a deep reinforcement learning architecture with Symbolic Learning.

While the previous studies has focussed on using NLM on a different problem set (Honghua et al., 2019 [1]) or solving the Sudoku puzzle with complete Deep Learning approaches (Wang et al., 2019 [3]), our experiment emphasizes combining Symbolic Learning with Deep Learning and using this hybrid architecture to solve a new and complex problem. This experiment also focuses on realizing the true potential of NLM's in different areas of application, Sudoku puzzle in this case.

3 Proposed Methodology

Figure 1 illustrates our proposed model architecture for Neuro-Symbolic Sudoku Solver. It is a Hybrid architecture of Deep Reinforcement with Symbolic Learning. The first half of the model constitutes the Deep Learning phase where the sigmoid function is used in between the layers while the softmax function is used in the output layer. The input layer consists of 4 neurons where each of them accepts a certain type of input parameter. The flexibility to allocate a

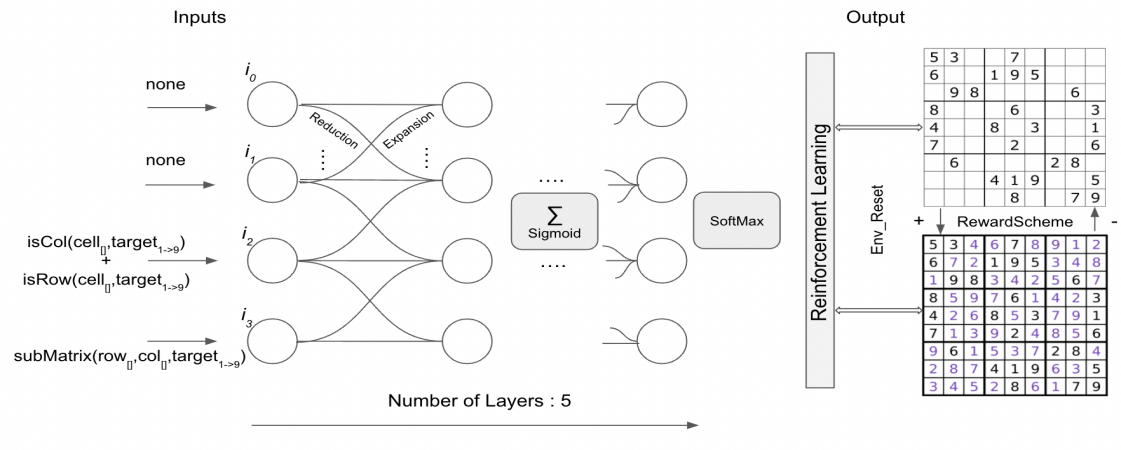

Fig. 1: Model Architecture of Neural Logic Machine

certain type of input to each neuron, leads to more systematicity in the model. For instance, out of the four neurons in the input layer, the first neuron i_0 accepts only nullary predicate, i_1 accepts unary predicate, i_2 accepts binary and i_3 takes ternary predicate as an input.

The problem of sudoku puzzles requires checking for each row, column and submatrix to maintain a valid configuration of the sudoku grid. In order to check that, the coordinates of the rows and columns along with the target values are passed in the input layer. Therefore, neurons i_2 and i_3 receive input as binary and ternary predicates, while neurons i_0 and i_1 get Null input. In contrast to this experiment, [1] shows how their problems require predicates for a different set of neurons in the NLM. Once the set of predicates is given to the input layer, the input for the following layers are reduced or expanded based on the arity of the previous layer.

The output from the softmax layer is fetched to the Reinforcement Learning (RL) module which constitutes the phase 2 of the architecture shown in figure 2. The RL module takes care of three main functionalities: Allocating 1 positive reward for getting a fully solved grid, rewarding -0.01 for every move and performing an environmental reset after checking with all the target values. The RL triggers an environmental reset if none of the target values from 1-9 can fill an empty cell while maintaining a valid configuration of the sudoku grid. During this reset, all the filled values are emptied and the sudoku grid is initialized to its initial state and a new iteration begins.

Given an unsolved sudoku grid, we have implemented a multistage architecture to solve the grid step by step. Figure 2 illustrates the high level architecture of the Neuro-Symbolic Sudoku solver. This high level architecture aligns with the problems experimented as part of the original NLM paper.

The first step in this implementation diagram consists of calculating the boolean predicates. There are three important rules to solve a sudoku grid. The solver needs to put a number in an empty cell such that the resulting grid

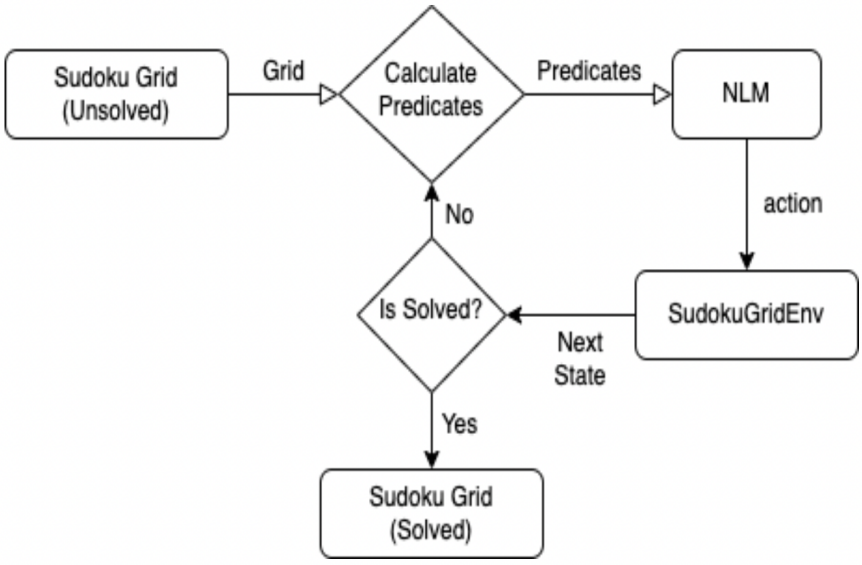

Fig. 2: High Level System Architecture

configuration must be valid. Here, valid configuration refers to the state where all the numbers in each row, column and 3x3 submatrix must be distinct. With the help of these conceptual rules, lifted boolean predicates are created.

Lifted rules are generalized rules which apply to a task to crack its solvability. These rules are applicable to any example in that task domain irrespective of its configuration or complexity. These rules can be seen as the simplest fundamental rules of solving the system. We define the predicate **isRow(r, x)** which computes whether number x exists anywhere in row r. Similarly, predicate **isColumn(c, x)** computes whether number x exists anywhere in column c and predicate **isSubMat(r, c, x)** computes whether number x exists in the 3x3 submatrix containing the cell (r, c). Based on above definition of the predicates, isRow(r, x) and isColumn(c, x) are binary predicates and both result in [9, 9] shaped tensors whereas isSubMat(r, c, x) is a ternary predicate that results in [9, 9, 9] shaped tensor.

It is also worth mentioning that in this study we do not input the unsolved grid into the input layer of the neural network. Instead, we compute the predicates as described above and pass those predicates as a set of input. Since there are two binary predicates, we concatenate the values of isRow(r, x) and isColumn(c, x) and call the resultant tensor as a stacked binary predicate. Now, these predicates can be feeded into the input layer of the neural network.

The last layer of the NLM logic machine computes the softmax value and provides the empty cell position (r, c) as well as the target value x to put in that empty cell. Here comes the role of the reinforcement module. Reinforcement environment checks if putting x into (r, c) makes a valid sudoku grid or not. Based on the previous assertion, it generates the next state and computes the

positive (if the next state is not a valid sudoku grid) or negative (if the next state is not a valid sudoku grid) reward. The reinforcement also checks if the next state generates a solved sudoku grid. In that case, we break from the architecture and print the output, otherwise, we perform the same steps in each iteration.

However, the above strategy may take indefinite steps to find a solution. To prevent that, we have defined limited proxy steps to forcefully stop the algorithm loop after a certain number of steps. The proposed algorithm yields a success rate of 1 if it solves the grid otherwise 0.

3.1 Training Details

Problem	**Epochs**	**Batch Size**	**Loss**	**Optimizer**	**LR**	**RL Rewards**	γ
Sudoku Puzzle	50	4	Softmax-CE	Adam	0.005	+1,-0.01	0.99

Table 1: Training Details of Neuro-Symbolic Sudoku Solver

The hyper parameters and the training details of the NLM for solving a sudoku puzzle of 9X9 grid are shown in table 1.

4 Result and Analysis

No. of Empty Cells	**Max. Steps**	**Success Rate**
3	81	0.94
3	150	1.00
3	400	1.00
3	729	1.00
5	81	0.80
5	150	0.96
5	400	0.99
5	729	1.00
8	80	0.68
8	150	0.92
8	400	0.98
8	729	1.00

Table 2: Comparison of different training parameters

Our experiment involves multiple settings (number of empty cells, dimensions and optimal steps) of the grid, which are often modified during the testing phase to understand the performance of the model and obtain a pattern from the results. To begin the experiment, the number of empty cells and the maximum steps in the sudoku grid are limited to 3 and 81 respectively. These are then gradually incremented as the model trains. As these parameters are changed over time, the complexity to solve the problem also increases. However, our result suggests that NLM can perfectly confront this complexity and yields 100% accuracy in most of the cases.

To give a better understanding on how the model performs with different parameters, we have concluded the success rate for each parameter that was tested on the model. Table 1 shows the comparison and performance under each setting that is tested on our modified version of the NLM. The model when tested with the minimum number of empty cells (nr_empty) i.e. 3, gives a success rate of 0.94. However, when the maximum number of steps (max_steps) are increased from 81 to 150, the model gets a perfect score of 1.

A similar case is also observed when the model receives 5 empty cells with 81 and 729 max_steps. For less optimal steps, the model yields a comparatively lower score as compared to 3 empty cells. However, when the model does not have a restriction on the optimal steps (set to a max of 729), it performs fairly well and gets 100% accuracy. From this we conclude that success rate is directly related to the maximum number of steps allowed to the model. Another inference obtained from table 2 is that when the number of empty cells are kept below 5 (i.e. 3) and the optimal steps are less, the drop in the success rate is significantly low as compared to higher number of empty cells (5 or 8) with the same optimal steps. Therefore, the success rate of NLM model is strongly determined by the relation of:

$$Success\ Rate \propto nr\_empty * max\_steps \tag{1}$$

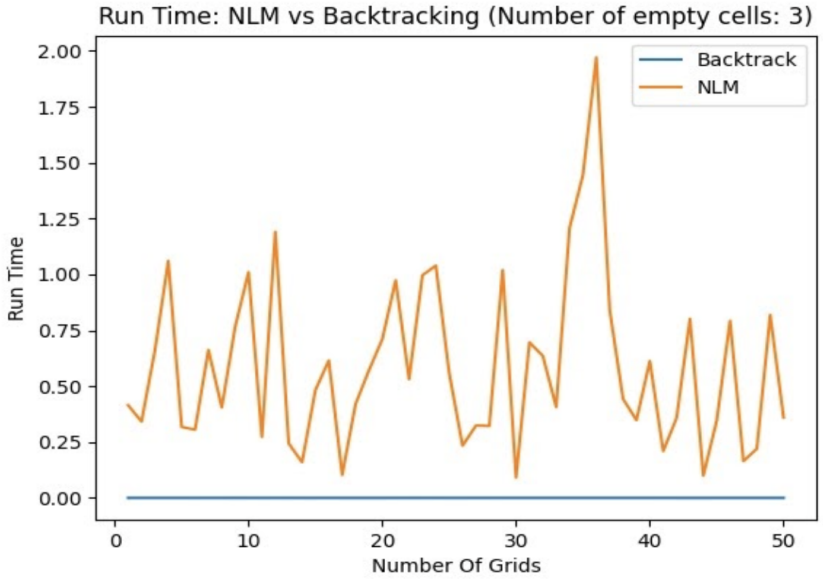

Fig. 3: Comparison of NLM and backtracking convergence time

In addition to fine-tuning the model, we have also drawn a time complexity analysis of NLM with a traditional backtracking algorithm for solving sudoku puzzles. Both, NLM and the backtracking algorithm are provided with the same set of grids and their time to solve the complete grid is highlighted in figure 3. The motivation behind this is to showcase the difference in the principle working

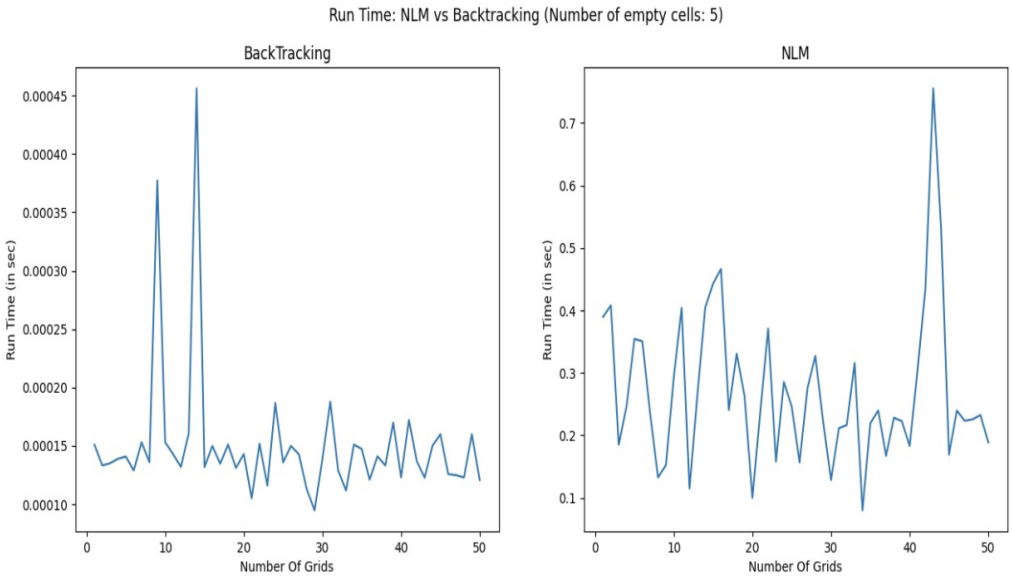

Fig. 4: Separate convergence time for Backtracking and NLM for same problem of both the algorithms and analyse their convergence time (with limited training in case of NLM).

While the backtracking algorithm takes a constant time of nearly 0.00045 sec to solve the same set of 9X9 grids that are given to NLM, on the other hand, NLM takes a considerably higher amount of time to converge. It is also worth mentioning that figure 3 demonstrates the time taken by NLM with 729 maximum number of steps. The reason for backtracking to converge faster is that it solves the grid in an optimal number of steps i.e MaxStepsbacktracking $\simeq$ Number of empty cells. In figure 3, the peak in time in case of NLM denotes that instance when the environment is reset due the the formation of an invalid configuration of the sudoku grid. During this instance, the model first receives a negative reward through reinforcement learning and then resets the environment as there are no possible target values to test in the empty cells. In this case, NLM again tries to fill the empty cells but with a different set of target values from the beginning. However, even with 10 empty cells, our modified version of NLM always takes less than 2.0 sec to converge.

5 Conclusion and Future Work

The main focus of this study is to tackle one of the drawbacks of the traditional Neural Networks i.e. 'systematicity'. While the Neural Networks fail and perform poorly, NLM's can solve the same task with 100% accuracy. NLM has been trained and tested by Honghua et al., 2019 [1] on various tasks that Deep Learning models failed to solve and converge. In our paper, we added an extra

application to their architecture and solved a more complex problem to test the robustness of Neuro Logic Machines.

While Neuro Logic Machines failed to converge for sudoku puzzles faster than the backtracking algorithm, however, it is evident from this study that a Neuro Logic Machine can be trained to solve tasks that a conventional Deep Learning model will fail to do. Also, NLM receives a random combination of grid and nr_empty cells from a data set of one million grids, thereby supporting the fact that the high success rate of NLM is not due to model's over-fitting. Thus, with this experiment, we have been able to strengthen the argument (Honghua et al., 2019 [1]) that NLM can solve tasks with 100% accuracy without being over-fitted. An inference is also deduced in section 4 that the success rate is directly associated with the number of empty cells and the maximum number of steps that model is allowed to traverse.

To conclude, Neuro Logic Machines are capable of solving complex problems using Reinforcement Learning. The fine tuning of this model to converge for the current and other problems in the optimal number of steps is expected to be covered in the future work. Their applications can be extended further in the future with more games (like Ken Ken puzzles) and mathematical problems (like searching tasks). It is also expected to cover problems where NLM doesn't yield a success rate of 1.

6 Acknowledgment

We thank Dr.Leake, professor at Indiana University Bloomington, for being our instructor and guiding us through this experiment and thereby supporting our work.

References

1. H.Dong, J.Mao, T.Lin, C.Wang, L.Li and D.Zhou, "Neural Logic Machines", In ICLR, 2019.
2. R.Sutton, D.McAllester, S.Singh, Y.Mansour, "Policy Gradient Methods for Reinforcement Learning with Function Approximation", AT&T Labs - Research, 1057-1063, 1996.
3. J.Andreas, M.Rohrbach, T.Darrell, and D.Klein, "Neural module networks", In CVPR, 2016.
4. D.Bahdanau, K.Cho, and Y.Bengio, "Neural machine translation by jointly learning to align and translate", In ICLR, 2015
5. Y.Bengio, J.Louradour, R.Collobert, and J.Weston, "Curriculum learning", In ICML, 2009.
6. J.Cai, R.Shin, and D.Song, "Making neural programming architectures generalize via recursion", In ICLR, 2017.
7. X.Chen, C.Liu, and D.Song, "Towards synthesizing complex programs from input output examples", In ICLR, 2018.