Testing DeepProbLog on Sudoku

Rohan Sambidi (1225571620)

April 29, 2023

Abstract

DeepProbLog is a Neuro-Symbolic (Ne-Sy) paradigm that incorporates deep learning into an existing probabilistic logic programming language, ProbLog, through neural predicates. DeepProbLog has been tested on various complex problems that involve learning and reasoning such as MNIST addition, sorting, word algebra problems, etc. In solving these problems, DeepProbLog achieved state-of-the-art results. However, to the best of my knowledge, there has been no implementation of DeepProbLog for solving Sudoku. So, for this project, I will be testing DeepProbLog on Sudoku. I aim to analyze the performance of learning and inference by DeepProbLog for two tasks: classifying visual Sudoku, and solving visual Sudoku. My goal is to gain deeper insights into the integration of neural nets with ProbLog, and compare DeepProbLog with various Ne-Sy frameworks.

1 Introduction

Neural networks are best known for their incredible capability of low-level perception, whereas symbolic AI is capable of high-level reasoning. With incredible strides being made in the field of deep learning, there are still numerous drawbacks that have still not been addressed. Some of these are based on explainability of the deep learning black-box, excessive computation usage, and requirement of high amounts of data for diminishing performance improvements. Symbolic AI (logical and probabilistic representations) can counter these issues but is restricted by the necessity of manually building knowledge graphs. Ne-Sy AI integrates these opposing ends of the spectrum and tries to be the best of both worlds by offering performance comparable to deep learning models, while also being explainable. Integration of neural networks and symbolic reasoning for machine level perception is being studied for over a decade now, and has proven to give promising results.

Various studies have proven that incorporating explainability into deep learning can yield problem-specific solutions, but is usually challenging and would lack scalability and generalizability. Hence, Manhaeve et al. [2018] approached the problem from the other end. They tried to incorporate perceptibility into symbolic AI. This resulted in DeepProbLog, a Ne-Sy paradigm capable of representing and reasoning.

Training AI to solve puzzles is an age-old approach for validation. Puzzles like Sudoku have proven to be challenging for deep learning models, since they lack explainability. Since Ne-Sy frameworks can incorporate reasoning, testing these models on Sudoku would be a good approach to validate them. However, DeepProbLog was not tested on Sudoku in the original work by Manhaeve at al. They released an extended version of the original DeepProbLog publication [Manhaeve et al., 2019] that doesn't included tests on Sudoku either. On the other hand, Sudoku has been used for testing by the authors of other Ne-Sy paradigms. Few examples include NeuPSL, NeurASP, and SATNet. NeuPSL [Pryor et al., 2022] was tested on visual Sudoku classification and achieved an accuracy of 85.05 ± 02.65% when trained on 200 puzzles. NeurASP [Yang et al., 2020] was tested on solving Sudoku and achieved an accuracy of 100% on when trained on 25 puzzles. SATNet [Wang et al., 2019] was also tested on solving Sudoku and achieved an accuracy of 63.2% on visual Sudoku and 98.3% on original Sudoku, when trained on 9000 puzzles. Since many paradigms use Sudoku for validation, testing DeepProbLog on Sudoku would allow fair comparison with those paradigms.

To the best of my knowledge, there have been no studies on testing DeepProbLog on Sudoku. Hence I chose to test DeepProbLog on Sudoku for this project. I propose two problems for testing that involves perception and reasoning:

- **T1**- Classifying visual Sudoku: Given a Sudoku board, determine whether the puzzle is complete (solved) or incomplete (unsolved).
- **T2**- Solving visual Sudoku: Given an incomplete Sudoku grid, solve the puzzle.

According to Pryor et al. [2022], since Ne-Sy AI is a fairly new field, there is no proper benchmark to evaluate Ne-Sy frameworks. Hence, they created a database called ViSudo-PC for the test on classifying visual Sudoku which can serve a benchmark for this particular test. I will use this dataset to evaluate DeepProbLog for T1 and T2.

2 Background and Related Work

3 Technical Contribution

The main idea of this project was to implement a DeepProbLog program that can solve a Sudoku puzzle given as an image. However, due to various challenges I was not able to achieve the results. In this section, I describe the tasks I have completed so far.

The task of this project was to design a CNN that can recognize the handwritten digits in a Sudoku board. I trained a CNN model that will recognize digits individually. The outputs of this CNN will be used as neural predicated for DeepProbLog program. When I trained my first CNN model, the classification accuracy was around 97%. This led to inconsistency in Sudoku puzzle due to mis-classification of the digits. I realized this might cause poor inference in ProbLog. I learned that the original architecture is probably over-fitting due to its depth. I also realized that code for the normalization step in the pre-processing step was incorrect. So, I modified the architecture to improve the accuracy. The current validation accuracy is 99.11%. The model was trained for 10 epochs. Total training time was around 300 seconds.

The next task was to process the Sudoku image. The major component of this task was to segment the digits. I tried to write a program to segment the digits without compromising any generalizability, i.e., the segmentation did not rely on the position or size of the digit. However, the segmented images were returned in wrong order. So, I then segmented the image by taking advantage of the positions of the digits. Since the data in ViSudo-PC was generated using MNIST, each digit has a fixed size of 28x28 pixels while also being spaced evenly. So, I segmented the Sudoku image into a 9x9 grid by simply dividing the image into 81 chunks[1]. Though the segmentation works on the current data, it may not work on data that was not created using MNIST. After segmenting and pre-processing the images, I passed them individually to the CNN. The CNN returned the classification results that I stored in an array. This array to similar to the neural predicates need for DeepProbLog. Figure 1 shows the classification results on an input. A few digits were being misclassified. The figures show the results I obtained form CNN.

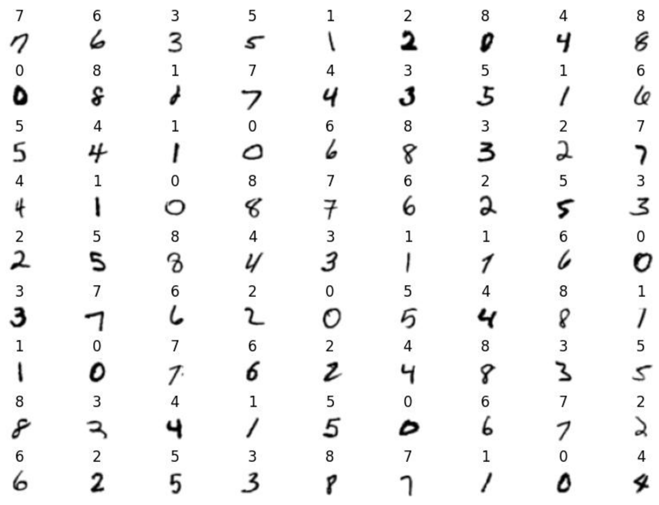


Figure 1: Example of classification result from CNN

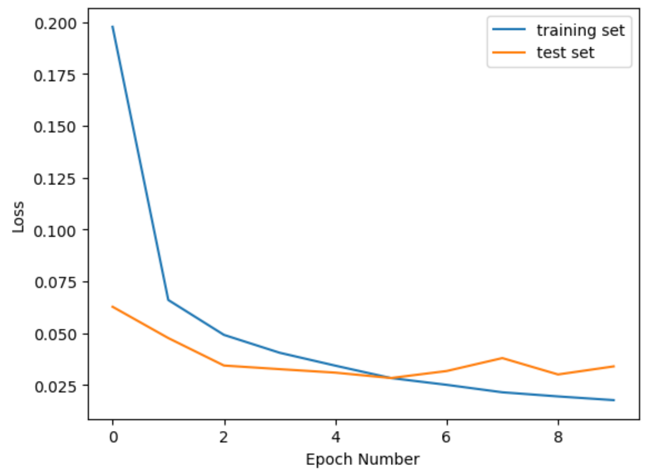


Figure 2: Plot showing train and test accuracy of CNN for each epoch

To write a ProbLog program for solving Sudoku, I referred the official documentation to familiarize myself with the ProbLog environment. Then, I was able to write a program to solve a given 4x4 Sudoku puzzle. Though ot was not very efficient, it produced the outputs in aroung 0.34 seconds. However, when extended to the proper 9x9 Sudoku puzzle, the program was extremely slow. So, tried using packages to improve the performance. I referred SWI-ProbLog website, and one of the tutorials had the program to solve Sudoku. The program was running fast without errors, giving the re-

[1]Acknowledgement: This implementation idea was suggested by Adam Ishay

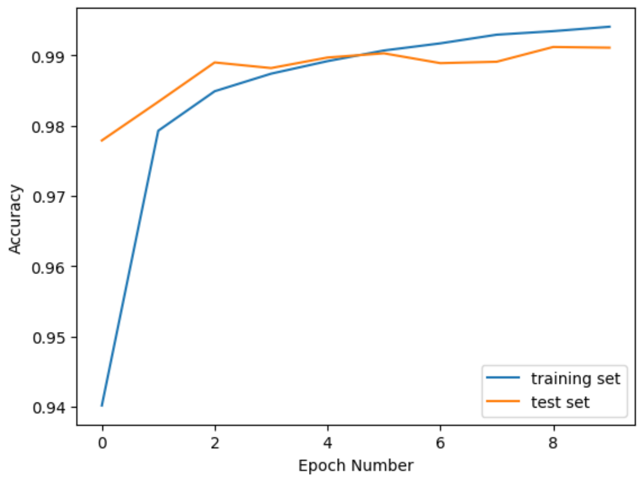


Figure 3: Plot showing train and test loss for CNN for each epoch

sults in less than 0.4 seconds. However, I was not able to integrate it with Python, where I could use the neural network output as a query.

That being said, I was able to test the DeepProbLog on MNIST addition task mentioned the original work. The program took around 406 seconds for 1 epoch. The resulting accuracy was 95.12%.

4 Conclusion

DeepProblog is an interesting approach to combine neural networks with symbolic models. It made me curious to explore more about the possibilities of such approach. Though this project, I also realised its difficult to work with without in-depth understanding. I was able to manage getting the results from the neural network side, as well as the logic programming. However, I failed to combine these two and achieve an end-to-end architecture to perform the tasks originally expected in the proposal.

5 Self-assessment

One of the reasons I chose this project was that I have never directly worked on probabilistic logic programming before. I wanted to explore, understand and gain some experience with the semantics of logic programming. The problems I faced were not with the programming language itself, but with integrating it with a general-purpose programming language like Python. I should have spend more time in understanding the configurations of various libraries. Though the theory behind my project was clear, I faced challenges in the implementation. I failed in understanding software dependencies and configurations. Most of the errors I encountered during my work were very tricky to resolve. Even after searching online, and sites like StackOverflow and GitHub, I couldn't find the ideas to overcome these errors. I was not able to achieve the alternative goals I set during the project check-in either.

References

Robin Manhaeve, Sebastijan Dumancic, Angelika Kimmig, Thomas Demeester, and Luc De Raedt. Deepproblog: Neural probabilistic logic programming. *CoRR*, abs/1805.10872, 2018. URL `http://arxiv.org/abs/1805.10872`.

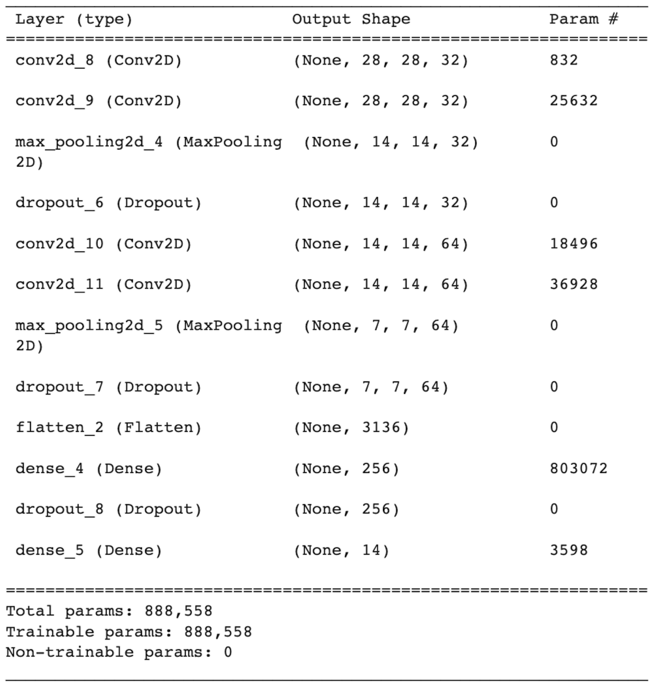
Robin Manhaeve, Sebastijan Dumancic, Angelika Kimmig, Thomas Demeester, and Luc De Raedt. Deepproblog: Neural probabilistic logic programming. *CoRR*, abs/1907.08194, 2019. URL `http://arxiv.org/abs/1907.08194`.

Connor Pryor, Charles Dickens, Eriq Augustine, Alon Albalak, William Wang, and Lise Getoor. Neupsl: Neural probabilistic soft logic. 2022. URL `https://arxiv.org/abs/2205.14268`.

Po-Wei Wang, Priya L. Donti, Bryan Wilder, and J. Zico Kolter. Satnet: Bridging deep learning and logical reasoning using a differentiable satisfiability solver. *CoRR*, abs/1905.12149, 2019. URL `http://arxiv.org/abs/1905.12149`.

Zhun Yang, Adam Ishay, and Joohyung Lee. Neurasp: Embracing neural networks into answer set programming. In Christian Bessiere, editor, *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI-20*, pages 1755–1762. International Joint Conferences on Artificial Intelligence Organization, 7 2020. doi: 10.24963/ijcai.2020/243. URL `https://doi.org/10.24963/ijcai.2020/243`. Main track.

A CNN architecture

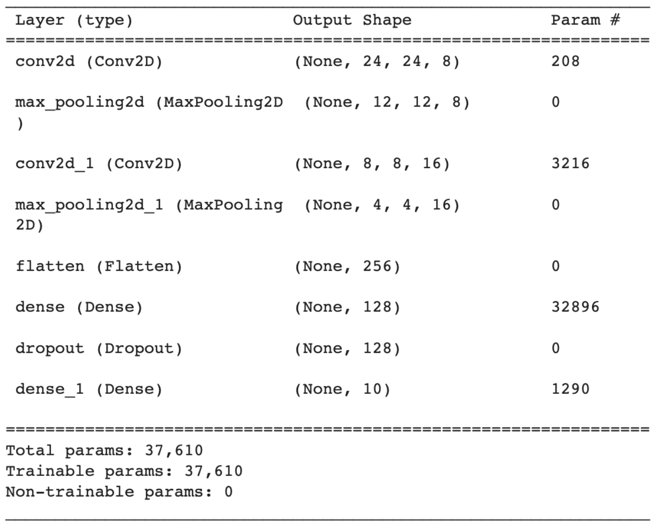


```
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_8 (Conv2D)           (None, 28, 28, 32)        832

 conv2d_9 (Conv2D)           (None, 28, 28, 32)        25632

 max_pooling2d_4 (MaxPooling  (None, 14, 14, 32)       0
 2D)

 dropout_6 (Dropout)         (None, 14, 14, 32)        0

 conv2d_10 (Conv2D)          (None, 14, 14, 64)        18496

 conv2d_11 (Conv2D)          (None, 14, 14, 64)        36928

 max_pooling2d_5 (MaxPooling  (None, 7, 7, 64)         0
 2D)

 dropout_7 (Dropout)         (None, 7, 7, 64)          0

 flatten_2 (Flatten)         (None, 3136)              0

 dense_4 (Dense)             (None, 256)               803072

 dropout_8 (Dropout)         (None, 256)               0

 dense_5 (Dense)             (None, 14)                3598

=================================================================
Total params: 888,558
Trainable params: 888,558
Non-trainable params: 0
```

Figure 4: Summary of the original CNN

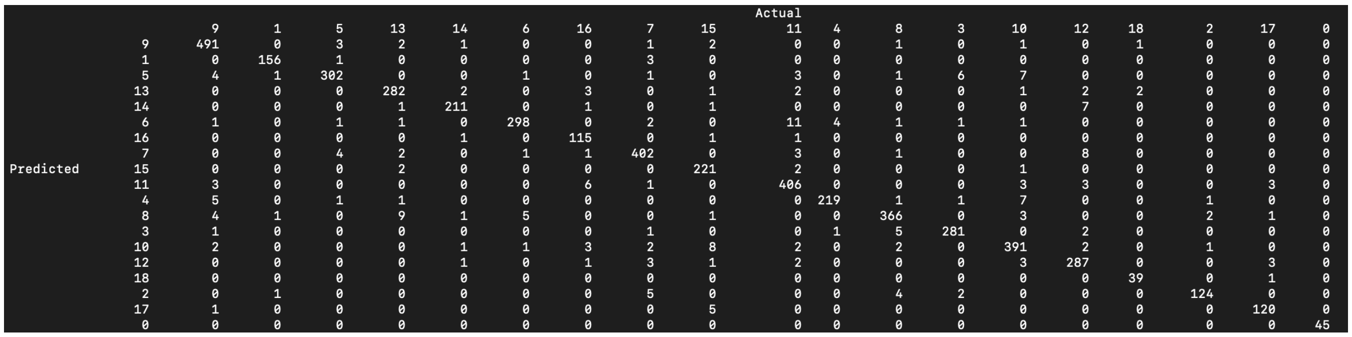
B MNIST Addition

I was able to get the results for MNIST Addition task form the original paper. Figure 6 shows the confusion matrix returned by the program on my terminal.



```
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 24, 24, 8)         208

 max_pooling2d (MaxPooling2D  (None, 12, 12, 8)        0
 )

 conv2d_1 (Conv2D)           (None, 8, 8, 16)          3216

 max_pooling2d_1 (MaxPooling  (None, 4, 4, 16)         0
 2D)

 flatten (Flatten)           (None, 256)               0

 dense (Dense)               (None, 128)               32896

 dropout (Dropout)           (None, 128)               0

 dense_1 (Dense)             (None, 10)                1290

=================================================================
Total params: 37,610
Trainable params: 37,610
Non-trainable params: 0
```

Figure 5: Summary of the updated CNN



```
                                                                               Actual
                      9      1      5     13     14      6     16      7     15     11      4      8      3     10     12     18      2     17      0
              9     491      0      3      2      1      0      0      1      2      0      0      1      0      1      0      1      0      0      0
              1       0    156      1      0      0      0      0      3      0      0      0      0      0      0      0      0      0      0      0
              5       4      1    302      0      0      1      0      1      0      3      0      1      6      7      0      0      0      0      0
             13       0      0      0    282      2      0      3      0      1      2      0      0      0      1      2      2      0      0      0
             14       0      0      0      1    211      0      1      0      1      0      0      0      0      0      7      0      0      0      0
              6       1      0      1      1      0    298      0      2      0     11      4      1      1      1      0      0      0      0      0
             16       0      0      0      0      1      0    115      0      1      1      0      0      0      0      0      0      0      0      0
              7       0      0      4      2      0      1      1    402      0      3      0      1      0      0      8      0      0      0      0
Predicted    15       0      0      0      2      0      0      0      0    221      2      0      0      0      1      0      0      0      0      0
             11       3      0      0      0      0      0      6      1      0    406      0      0      0      3      3      0      0      3      0
              4       5      0      1      1      0      0      0      0      0      0    219      1      1      7      0      0      1      0      0
              8       4      1      0      9      1      5      0      0      1      0      0    366      0      3      0      0      2      1      0
              3       1      0      0      0      0      0      0      1      0      0      1      5    281      0      2      0      0      0      0
             10       2      0      0      0      1      1      3      2      8      2      0      2      0    391      2      0      1      0      0
             12       0      0      0      0      1      0      1      3      1      2      0      0      0      3    287      0      0      3      0
             18       0      0      0      0      0      0      0      0      0      0      0      0      0      0      0     39      0      1      0
              2       0      1      0      0      0      0      0      5      0      0      0      4      2      0      0      0    124      0      0
             17       1      0      0      0      0      0      0      0      5      0      0      0      0      0      0      0      0    120      0
              0       0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0     45
```

Figure 6: Confusion Matrix for MNIST addition task