COMPSCI 602
Final Report

Frank Chiu
College of Information and Computer Science
University of Massachusetts
Amherst, MA 01003
fchiu@umass.edu

Project Title: Complementary Analysis of "Learning Where and When to Reason in Neuro-symbolic Inference" [3]

1 Introduction

Neural Attention for Symbolic Reasoning (NASR), proposed by Cornelio et al., is a neuro-symbolic pipeline designed to enforce hard logical domain constraints at inference time[3]. NASR was designed to solve predicate classification tasks such as the Visual Sudoku Task and Visual Scene Graph Prediction Task[3]. NASR has two major components (1) a neural attention module that learns to direct the reasoning effort to focus on potential prediction errors, while keeping other outputs unchanged, and (2) a symbolic reasoning module capable of correcting structured prediction errors. The NASR pipeline achieves efficiency and accuracy by predicting solutions and identify potential errors to reduce the solution search space through the neural attention module and generate the final solution through the symbolic reasoning module. In this work, we replicate and extended Cornelio et al.'s paper "Learning Where and When to Reason in Neuro-symbolic Inference" under the context of Visual Sudoku Task. We verified the performance of NASR presented in the original paper, identified discrepancies between a pipeline component's design and performance and further analyzed its robustness to out-of-distribution data, modular interactions, and designed a contrastive constraint loss to complement the original paper's analysis.

2 Related Work

Several approaches have been used in an attempt to enforce constraints on neural network output. Xu et al. proposed a semantic loss that quantifies the level of disagreement between the output and the constraints [6]. In the work of Ashok et al., violation function was used to generate new data for adversarial training[1]. Both of these approaches attempts to enforce constraints through training. In some other similar neuro-symbolic approaches, Mulamba et al. combined a neural perception model with a symbolic solver to directly solve the visual sudoku problems[5]. NESTER, proposed by Dragone el al., enforces constraints by limiting the output space of the neural network. The outputs of a neural network are passed to a constraint program that enforces the constraints [4]. Broader et al. devised methods to learn criterion and definitions of a domain as a set of cost functions and constraints represented as Cost Function Networks (CFN)[2]. The CFN along with a Weight Constraint Satisfaction Solver is used to optimize graphical models[2]. Broader et al. showed that the combination of differential and non-differential architectures can provide excellent performance [2]. Latest approaches to the constraint satisfaction problems include applying soft constraints during the training process; however, these methods are inexact and whether these constraints were satisfied by the output during inference is not addressed and under-studied. The few methods that are capable of impose exact constraints often require to invoke a full symbolic solver, which is inefficient during inference time. The above work depicts the problem and trade-offs that needs to be addressed within constraint satisfaction problems.

3 Pipeline Components and Dataset

3.1 Dataset

The NASR pipeline was evaluated on 4 different sudoku datasets, each with different characteristic.

- **big_kaggle**: This dataset contains 100,000 puzzles, which is a subset of a bigger dataset containing 1 million boards hosted on Kaggle. These puzzles have an average of 33.82 hints per board (between 29 and 37).
- **minimal_17**: This dataset contains sudoku boards with minimal hints (17 hints).
- **multiple_sol**: This dataset contains 10,000 sudoku boards with two or more solutions.
- **satnet_data**: This dataset contains 10,000 Sudoku boards with an average number of hints equal to 36.22.

In this replication, we replicated the results for big_kaggle, minimal_17, and multiple_sol.

3.2 Pipeline Components

NASR pipeline can be broken into 3 sub-systems: a Neural Solver, a Mask Predictor, and a Symbolic Solver.

The Neural Solver takes an image of a sudoku board and proposes a solution by outputting probability distribution over numbers 1-9 for each cell in the sudoku board. The Neural Solver can be divided into even smaller sub-systems: Perception and SolverNN. The Perception is a CNN that takes in the sudoku image and outputs a distribution over 0-9 for each cells as an initialized empty board. The number 0 represents the empty cell in the original image. The SolverNN is a transformer that takes in the distribution generated by the Perception and attempts to output a distribution over numbers 1-9 for each cell as the proposed solution to the sudoku board.

The Masked Predictor has the same architecture as the SolverNN. It takes in the proposed solution generated by the Neural Solver and attempts to output a distribution over numbers 0 and 1 as an error prediction for cells in the sudoky board.

The Symbolic Solver is a deterministic program powered by Prolog, a logic programming language. With the specified sudoku rules, the symbolic Solver attempts to solve the sudoku board by reasoning over the proposed solution and error predictions generated by the Neural Solver and the Mask Predictor.

Since the pipeline is modular, the components can be trained in parallel and further refined using Reinforcement Learning (RL).

4 Replication Results and Analysis

Table 1 shows the NASR experiment results presented in the original paper. We attempted to replicate the results corresponding to the datasets, big_kaggle, minimal_17, and multiple_sol and were able to verify all statistics presented in the first three columns using the code and data provided by the authors. We verified:

- The Neuro-Solver has the ability to completely solve sudoku boards.
- The Mask-Predictor has the ability to mask out errors while preserving correct solutions to some extent and that the Symbolic Solver is capable of generating correct solutions based on Neuro-Solver and Mask Predictor output.
- NASR without RL struggles with difficult datasets such as minimal_17 and multiple_sol and that refining NASR with RL boosts model performance more on challenging datasets.
- The SolverNN is able to preserve 100% of the input (hint) cells across different datasets and have high correctnes of solution cells except on minimal_17.
- The Mask Predictor is able to preserve nearly 100% of SolverNN's correct solutions.

The replication results were mostly promising. However, we identified several ambiguities and opportunities to complement the original analysis.

	big_kaggle		minimal_17		multiple_sol		satnet_data	
Perception	99.64	74.56	99.84	87.70	99.48	65.70	99.32	63.20
SolverNN	100-98.33	62.68	100-61.56	0.00	100-93.72	46.70	100-94.84	24.00
Mask-Predictor	99.92-99.71		99.54-35.26		99.02-76.12		99.90-96.06	
Neuro-Solver		47.03		0.00		28.80		14.40
NASR w/o RL		80.02		1.59		60.00		76.40
NASR with RL		**84.24**		**87.00**		**73.00**		**82.20**
NASR-Heur.Mask		73.11		66.99		55.60		42.80

Table 1: Table from the original paper. Results of NASR pipeline on the visual Sudoku datasets, divided by the different modules. The metric used in the right column for each dataset correspond to the percentage of completely correct predicted boards. In addition to that we report (in the left column of each dataset): for the **Perception** *the number of correctly predicted cells on average*; for the **SolverNN** *preservation of input cells - correctness of solutions cells*; for the **Mask-Predictor** *true negative - true positive* (that corresponds to *correct solution cells that are not masked - error solution cells that are masked*).

5 Research Questions and Hypotheses

5.1 Research Question 1: Why does the Mask Predictor fail at Identify Errors on Challenging Datasets?

The Mask Predictor was designed to mask out the error cells of the proposed solution at the cost of masking the correct cells to guarantee the correctness of the input to the symbolic solver[3]. According to the paper, it was trained to put more focus on the negative examples, i.e. error cells, by weighing more on the training loss from the negative examples. However, the evaluation results on the challenging datasets show that the Mask predictor still retains high accuracy (99%) at preserving correct solution cells yet performs poorly at identifying error cells.

While the vanilla NASR pipeline struggled at identifying errors on challenging datasets such as minimal_17 and multiple_sol, refining the NASR pipeline with reinforcement learning (RL) greatly boosts its accuracy. With RL, the NASR pipeline's accuracy on minimal_17 improved from 2% to 86% and on multiple_sol from 70% to 78%. Interestingly, the SolverNN's performance within the NASR pipeline with RL somehow decreased, with the percentage of correct solution cells in the proposed solution decreased from 72% to 49%, whereas the performance of the Mask Predictor with RL skyrocketed, which masked 90% of incorrect solutions cells compared to only 22 percent without RL. This seems to suggest that the performance of the Mask Predictor more important than the performance of the Neural Solver. Since RL takes a significant amount of time to train, and the individual components within the pipeline can be trained separately, this raises the question:

- Why isn't the Mask Predictor performing as expected?
- How can we improve the Mask Predictor's accuracy at identifying errors without using reinforcement learning?

The biggest difference between the basic NASR pipeline and the NASR pipeline with RL is that the basic NASR pipeline is trained in parallel, where the training of one component doesn't effect the other, whereas the NASR pipeline with RL is trained as a whole. Specifically, the Mask predictor of the basic NASR pipeline was trained with manually one-hot encoded input distributions with noise that mimics the output distribution from the Neural solver. This lead to the hypothesis: **The mask predictor of the basic NASR pipeline will increase in performance if trained on real distributions (proposed solutions) generated by the Neural Solver.**

Another possible hypothesis is that **the hyper-parameter "pos_weight" for the training loss wasn't tuned to the optimal value**. As mentioned, the pretrained mask predictor was trained with binary cross entropy logit loss with a pos_weight of 0.01. This degree of pos_weight is very extreme. It essentially ignores the loss contributed by incorrectly masking correct solution cells and heavily weighs the loss contributed by mistakenly not masking the error solution cells. However, the behavior of the Mask Predictor contradicts what was intended during training. It may be that the pos_weight value of 0.01 wasn't extreme enough or the original authors made an mistake in their source code.

5.2 Research Question 2: How does the SolverNN's confidence in its output affect the Mask Predictor's performance and confidence in its output?

As shown in the original paper, NASR with RL performs much superior than the vanilla NASR pipeline on challenging datasets. The preliminary experiments showed that the performance of NASR heavily depends on the Mask Predictor's ability to mask the error solutions. Since the Mask Predictor takes in the the SolverNN's output distribution over sudoku cells as input and outputs a masking probability for each cell as output, it would be interesting to investigate how the characteristics of SolverNN's output affects the Mask Predictor's ability to mask errors. Specifically, how does the SolverNN's confidence in its output distributions affect the mask predictor's confidence and performance at predicting errors?

1. Hypothesis 1: The Mask Predictor's confidence in its error prediction is positively correlated with the Neural Solver's confidence in its proposed solutions.
2. Hypothesis 2: The Mask predictor's confidence and accuracy at masking errors is positively correlated with Neural Solver's confidence in its correct solutions.
3. Hypothesis 3: The mask predictor's confidence and accuracy at masking the errors is inversely correlated to the SolverNN's confidence in its error solutions.

The SolverNN and Mask Predictor's "confidence" in their output can be quantified by calculating the negative entropy of the output distributions, where a higher entropy indicates a smoother distribution and a lower entropy indicates a sharper distribution, which respectively indicates uncertainty and confidence. Confidence score is defined in section 6.2.1.

5.3 Research Question 3: How robust is the NASR pipeline to out-of-distribution sudoku boards?

In the original paper's results, NASR performed well in all 3 different types of sudoku dataset. However, after further inspection, these performance results were obtained from 3 different NASR pipelines respectively trained and evaluated on 3 different types of sudoku dataset instead of a general-purpose NASR pipeline trained on all types of sudoku dataset and is capable of dealing with different types of sudoku boards. The evaluation method seemed odd as different types of sudoku boards all follow the same logical rules and constraints, where cells in the same row, column, and block must be different from each other. Therefore, it would be interesting to investigate to what extend is the NASR pipeline learning the sudoku rules and how NASR pipeline trained on one datasets performs on the other.

1. Hypothesis 1: The NASR pipeline and its components trained on a difficult sudoku dataset, when evaluated on an easier dataset, performs on par with the pipeline and its components trained and evaluated on an easier dataset whereas the converse is not true.
2. Hypothesis 2: The NASR pipeline completely fails to learn the proper sudoku rules and fails at out out-of-distribution sudoku boards
3. Hypothesis 3: Trained on the same sudoku dataset, the vanila NASR pipeline performs better than NASR with RL when evaluated on a different type of dataset.

5.4 Research Question 4: How can we leverage Sudoku Rules to improve SolverNN's raw performance

The SolverNN is trained with incomplete sudoku boards as input and complete sudoku board as labels. However, this approach does not leverage the sudoku rules. Additionally, refining NASR models using RL takes a significant amount of time. It would be interesting to test whether a simple sudoku-rule-based regularization loss can effectively improve SolverNN's performance and in turn extend NASR's ability to enforce constraints from inference to training without expensive compute. The contrastive regularization loss penalizing the similar cell distributions within the same row, column, and block, hoping to improve SolverNN's performance.

1. Hypothesis 1: The cosine-similarity-based constraint regualrization loss will improve NASR's performance while maintaining a reasonable compute cost

2. Hypothesis 1: The KL-divergence-based constraint regualrization loss will improve NASR's performance while maintaining a reasonable compute cost

6 Research Design, Results, and Analysis

6.1 Research Design and Results for Question 1

To verify/falsify the hypothesis that the hyper-parameter "pos_weight" for cross entropy loss wasn't tuned to optimal value, two models were trained using pos_weights of 0.00001 and 100 to evaluate their performance against the original model trained on a pos_weight of 0.01. Extra metric including error rate, recall, precision were added to the pipeline evaluation. Results showed that pipeline trained with pos_weight of 0.01 and 0.00001 both identified more than 98% of the solution errors and the pipeline trained with a pos_weight of 100 only identified 37% of solution errors. The results suggests that a pos_weight of 0.01, used in the original paper, is a reasonable value for training the pipeline. After careful scrutiny in the source code, we found that **the authors made an evaluation error by mixing up the labels when evaluating the Mask Predictors**, thus the discrepancy between Mask-Predictor's performance at identifying errors and its intended design and the contradicting results between the replication and the original paper's evaluation and .

6.2 Research Design and Results for Question 2: Modular Interactions between SolverNN and Mask Predictor

6.2.1 Prelimiaries

To understand how the SolverNN's "confidence" in its outputs affects the Mask Predictor, we need to first quantify "confidence." Since the solverNN and Mask Predictor both output probability distributions, their confidence in their outputs can be defined as the negative entropy of the output probability distributions. The sharper the output distribution (concentrated probability mass), the lower the entropy and uncertainty, thus the higher the confidence; the smoother the output distribution, the higher the entropy and uncertainty, thus the lower the confidence. More concretely, the highest confidence is achieved when all the probability mass for a distribution is concentrated in a single class, and the lowest confidence is achieved when the distribution is uniform.

Below is the softmax operation with temperature, where p_i is the probability for class i, z_i is the raw logits for class i, and T is temperature.

$$p_i = \frac{\exp\left(\frac{z_i}{T}\right)}{\sum_{j=1}^{N} \exp\left(\frac{z_j}{T}\right)}$$

Below is the formula for confidence score (CF) and the entropy (H) of a distribution p, where p_i is the probability mass for class i, and an ϵ for numerical stability.

$$H(p) = -\sum_{i=1}^{N} p_i \log(p_i + \epsilon)$$

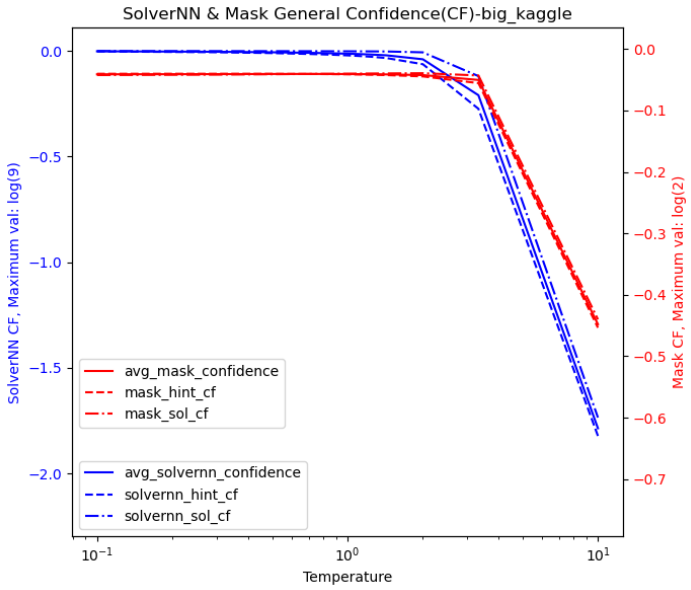
$$CF(p) = -H(p)$$

By lowering the temperature, the probability distribution will gradually converge to a one-hot encoded vector for the class with a dominant logit value, thus decreasing the entropy and increasing the confidence score. By increasing the temperature, the probability distribution will gradually converge to a uniform distribution, thus increasing entropy and decreasing the confidence score.

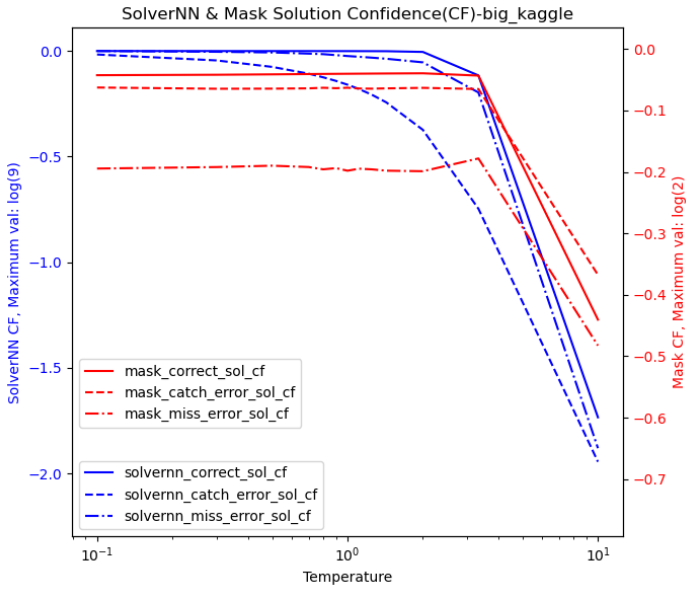
6.2.2 Experiment design and results

In the experiments, the temperature for the softmax operation on SolverNN's raw output logits are systematically varied to control the solverNN's confidence in its output probability distributions. These probability distributions are then passed through the Mask Predictor to obtain the Mask Predictor's output distribution. The confidence statistics for both output probability distributions and the performance statistics of the Mask predictor were then calculated and visualized in several plots. Specifically, the confidence and performance of the SolverNN and Mask Predictor of NASR with RL pipelines are evaluated and measured on both big_kaggle and minimal_17.

Manipulating the Entire Board Figures 1 and 2 show the average confidence of SolverNN and Mask Predictor in their output distributions across different values of temperature.



(a) The SolverNN and Mask Predictors' average confidence for big_kaggle on the hint cells, solution cells, and all cells



(b) The SolverNN and Mask Predictors' average confidence for big_kaggle on correct solution cells, masked error solution cells, and unmasked error solution cells

Figure 1: The SolverNN and Mask Predictors' average confidence for big_kaggle. Note that temperature is on a log scale and that the lowest possible confidence ($-\log(\text{num\_class})$) is different for SolverNN and Mask due to how entropy is calculated.

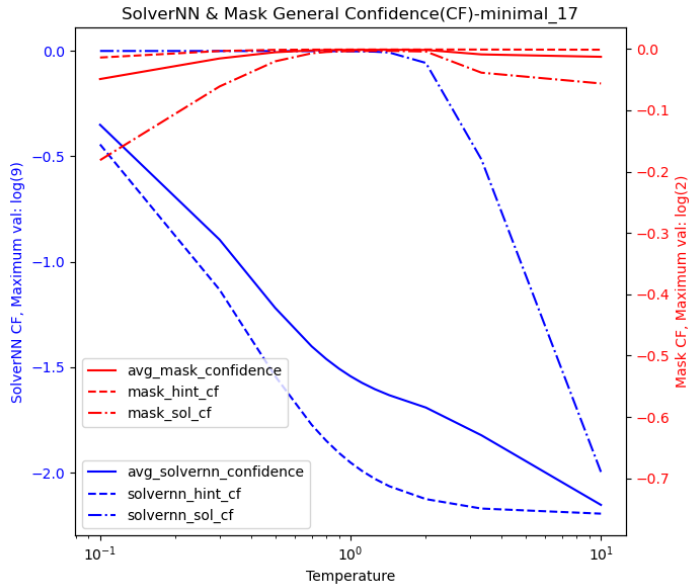
In figure 1 plot 1a, for the big_kaggle dataset, the average confidence for all cell categories(hint, solution, correct solution, masked & unmasked error solutions) all decreased as expected as the temperature increases. Plot 1b shows that the Mask Predictor is generally confident when identifying and masking the error cells and is more uncertain when it misses an error cell. It also shows that the SolverNN is confident at the (error) solutions where the Mask Predictor fails to identify and less confident in (error) solutions where the Mask Predictor successfully identifies.

In Figure 2, for the minimal_17 dataset, the plot 2a SolverNN surprisingly has more confidence in the solution cells than the hint cells, and the Mask Predictor stays confident regardless of the SolverNN's decrease confidence in its otuputs. Plot 2b shows that errors solution cells are more likely to be identified when SolverNN is less confident in its error solutions. It also shows that the SolverNN is much more confident when generating correct solutions than error solutions and that the Mask Predictor is confident at preserving correct solution cells.

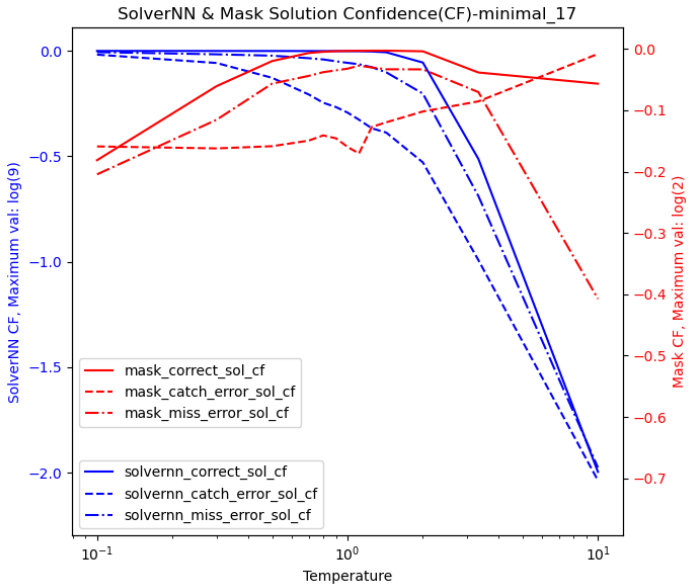
Manipulating the Correct Solution Cells In this section, only SolverNN's confidence in solution cells are being manipulated. This experiment was designed to investigate how the Mask Predictor would perform under fine-grained and meticulous variations in SolverNN correct solution. The softmax operation with varying temperatures is applied to only SolverNN's correct solution cells, the normal softmax operation (temperature =1) is applied to all other cells.

Figures 3 and 4 shows the the Mask Predictor's confidence in solution cells and its ability to mask out errors and preserve correct hint and solution cells and SolveNN confidence in correct solution cells drop.

Figure 3 shows the SolverNN and Mask Predictor's confidence in solution cells as the temeprature varies. For big_kaggle, plot 3a, the Mask Predictor performs as expected in **Hypothesis 2**, where its confidence in the solution cells decreases as SolverNN's confidence in correct solution cell decreases. However, this phenomenon doesn't hold in minimal_17. In plot 3b, the Mask Predictor is highly

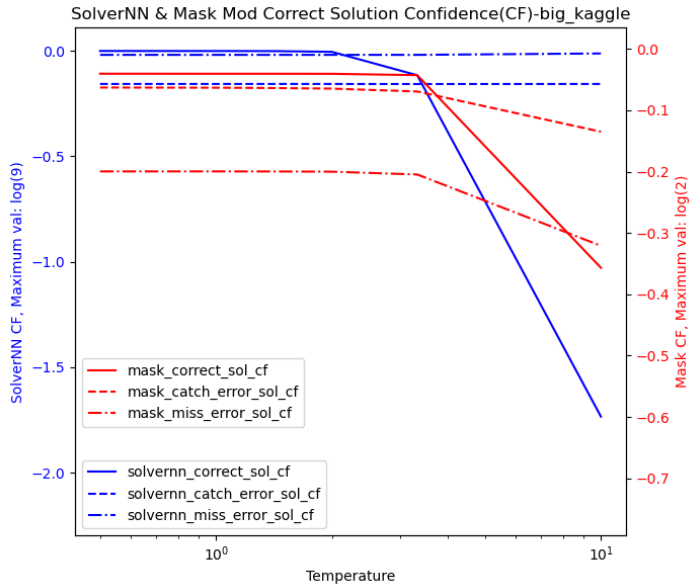


(a) The SolverNN and Mask Predictors' average confidence for minimal_17 on the hint cells, solution cells, and all cells

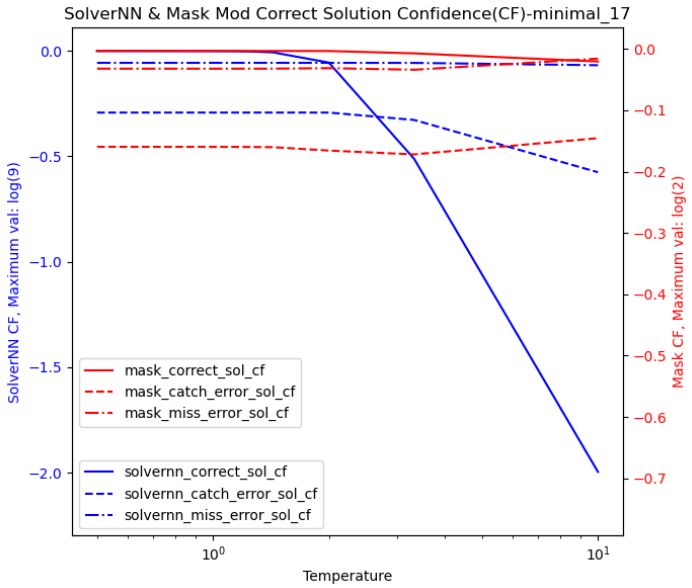


(b) The SolverNN and Mask Predictors' average confidence for minimal_17 on correct solution cells, masked error solution cells, and unmasked error solution cells

Figure 2: The SolverNN and Mask Predictors' average confidence for minimal_17. Note that temperature is on log scale and that the lowest possible confidence ($-\log(num\_class)$) is different for SolverNN and Mask due to how entropy is calculated



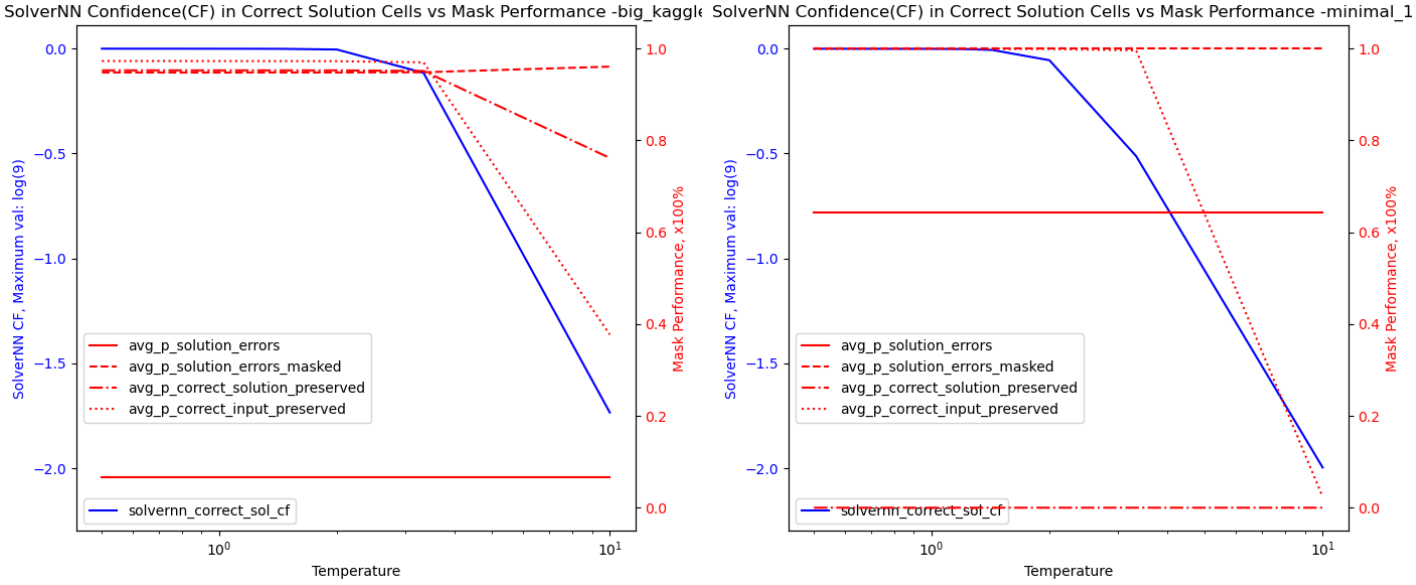
(a) big_kaggle



(b) minimal_17

Figure 3: SolverNN and Mask Predictor's confidence in correct solution cells, masked error solution cells, and unmasked solution cells as temperature varies when applying softmax to correct solution cells.

confident in its predictions regardless of SolverNN's confidence. Interestingly, in plot 3a, the model for big_kaggle is more confident in the errors masked than in the errors missed, which is a good characteristic for a predictor. The Mask Predictor for minimal_17 remains highly confident in all metrics, which is counterintuitive as we would expect its confidence in correct solutions would drop if SolverNN is less confident in the correct solutions.



(a) big_kaggle (b) minimal_17

Figure 4: Mask Predictor's Performance in masking errors and preserving correct hint and solution cells as SolverNN's confidence in correct solutions drops

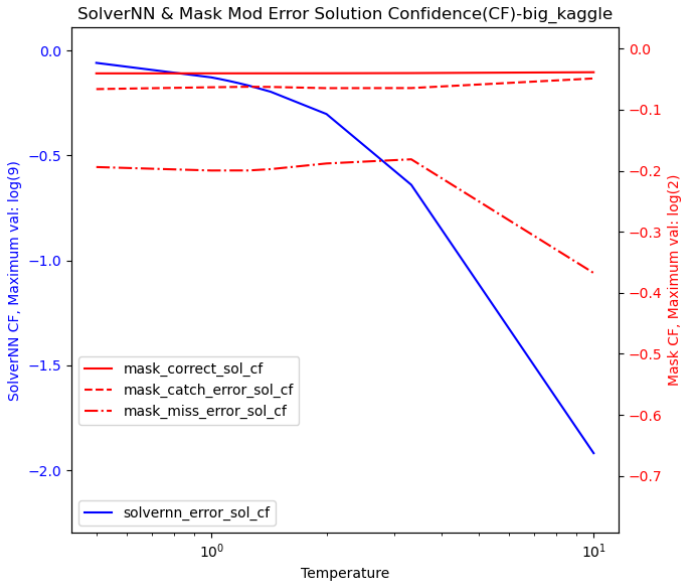
Figure 4 shows the Mask Predictor's ability to Mask out errors and preserve correct hint and solution cells as SolveNN's confidence in correct solution cells drop. Interestingly, the Mask Predictor for big_kaggle preserves more correct solution cells than correct input cells as SolverNN confidence drops in plot 4a. Plot 4b shows that Mask Predictor for minimal_17 simply masks out all correct solution cells regardless of SolverNN's confidence and performance.

Manipulating Error Solution Cells In this section, only SolverNN's confidence in error cells are being manipulated. This experiment was designed to investigate how the Mask Predictor would perform under fine-grained and meticulous variations in SolverNN error solution. The softmax operation with varying temperatures is applied to only SolverNN's error solution cells, the normal softmax operation (temperature =1) is applied to all other cells.

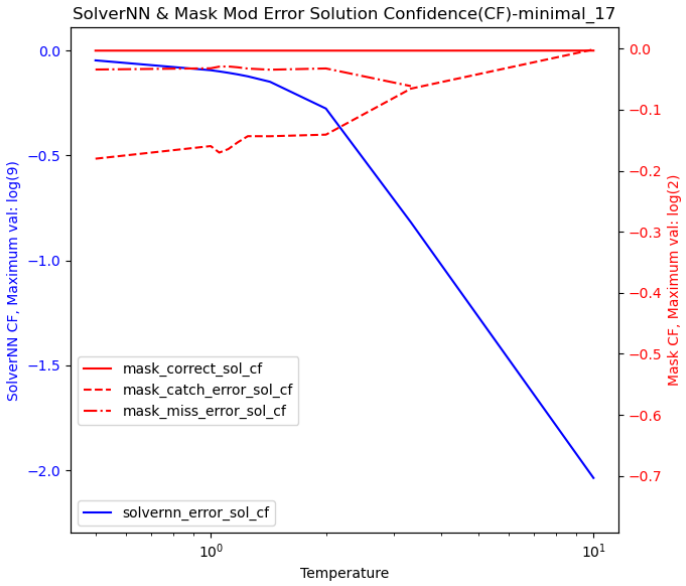
Figures 5 and 6 shows the the Mask Predictor's confidence in solution cells and its ability to mask out errors and preserve correct hint and solution cells and SolveNN confidence in error solution cells drop.

Figure 5 shows how Mask Predictor's confidence changes as SolverNN's confidence in error solution drops. Both Mask Predictors for big_kaggle and minimal_17 remains highly confident in correct solutions as SolverNN's confidence in error solutions drop. The Mask Predictor for big_kaggle performs somewhat as expected in plot 5a. The Mask Predictor is less certain when missing the error solutions and remains high confidence in catching error solutions, which is a good quality of an error predictor. The Mask Predictor for minimal_17 also performs as expected in plot 5b. It's confidence at masking errors increases as SolverNN decreases confidence in error solutions, which is also a good quality of an error predictor. These results partially verify the claims in Hypothesis 1 and 3 for Resaerch Question 2.

Figure 6 shows the Mask Predictors ability to mask out errors and preserve correct hint and solution cells. The Mask Predictor for both datasets performs exceptionally well in all metrics as expected,

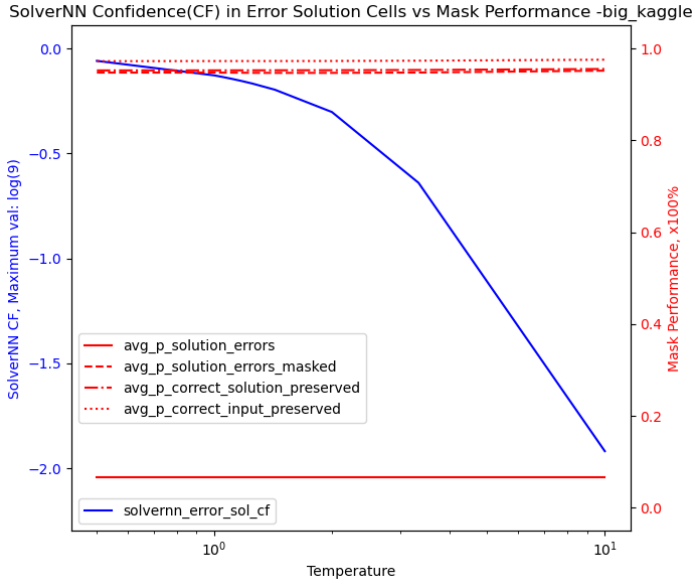


(a) big_kaggle

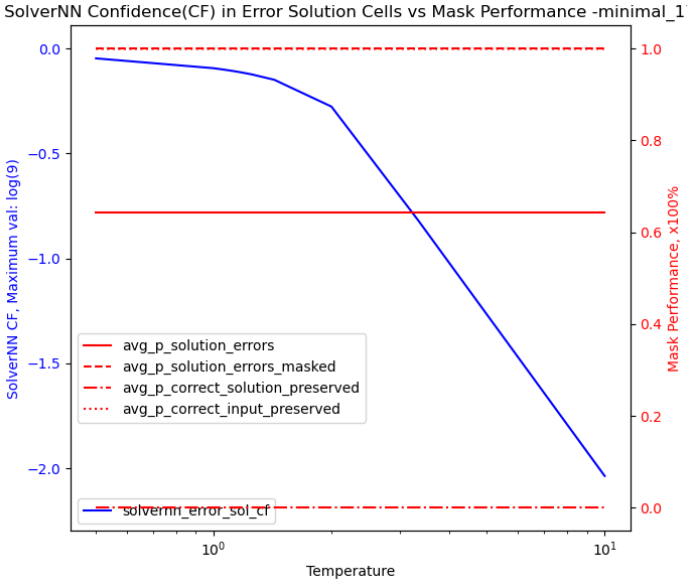


(b) minimal_17

Figure 5: SolverNN and Mask Predictor's confidence in correct solution cells, masked error solution cells, and unmasked solution cells as temperature varies when applying sofmax to error solution cells.



(a) big_kaggle



(b) minimal_17

Figure 6: Mask Predictor's Performance in masking errors and preserving correct hint and solution cells as SolverNN's confidence in error solutions drop

except that minimal_17, preserves none of the correct solutions cells, similar to the case when the correct solution cells were manipulated.

6.3 Research Design and Results for Question 3: Model Robustness to OOD Sudoku Boards

This experiment was designed to investigate the robustness of NASR models trained on a certain sudoku dataset to another sudoku dataset. The pretrained models and RL refined models trained on big_kaggle and minimal_17, respectively, are both evaluated on minimal_17 and big_kaggle.

Robustness results on RL-enhanced models	M_b on D_m	M_m on D_b
Correct solution boards	5.41%	6.02%
Avg. correct cells	49.82%	71.22%
Cells masked by Mask Predictor	24.32%	31.50%

^{M_b} Model trained on dataset *big_kaggle*
^{M_m} Model trained on dataset *minimal_17*
^{D_b} Dataset containing board examples from *big_kaggle*
^{D_m} Dataset containing masked board examples from *minimal_17*

Robustness results on pre-trained models	M_b on D_m	M_m on D_b
Correct solution boards	0%	0%
Avg. correct cells	17.13%	24.61%
Cells masked by Mask Predictor	13.65%	10.60%

^{M_b} Model trained on dataset *big_kaggle*
^{M_m} Model trained on dataset *minimal_17*
^{D_b} Dataset containing board examples from *big_kaggle*
^{D_m} Dataset containing masked board examples from *minimal_17*

Table 2: Comparison of robustness results for RL-enhanced and pre-trained models

Results show that the RL refined models performed horribly when evaluated on out-of-distribution data, solving around 6% of the sudoku boards respectively, with a 49.82% cell accuracy and 24.32% masking rate for big_kaggle-triained model and 71.22% cell accuracy and 31.50% masking rate for minimal_17-trained model. Since there's a possibility that RL refined models are too specialized to the characteristics of the dataset they were trained on, it is possible that the pretrained models may perform better than RL refined models. Results however show that pretrianed models perform far inferior to RL pretrained models, solving 0% of the sudoku boards, with a 17.13% cell accuracy and 13.65% masking rate for big_kaggle-triained model and 24.61% cell accuracy and 10.60% masking rate for minimal_17-trained model. Despite the poor performance, the model trained on the harder dataset, minimal_17, is slightly more robust than the model trained on the easier dataset, big_kaggle. Nontheless, the results verify **Hypothesis 2** for Research Question 3, where the models are properly learning the sudoku rules.

6.4 Research Design and Results for Question 4: Contrastive Constraint Regularization Loss

This experiment was designed to experiment whether a simple regularization loss leveraging sudoku rules can improve SolverNN's raw performance. The design of Constraint Regularization Loss are designed as the following.

6.4.1 Preliminaries

The contrastive constraint regularization loss penalizes the similarity between distributions of cells within the same row, colum, and block within the sudoku board. The contrastive loss is degined as

$$\sum_{n=1}^{\text{\#boards}} \sum_{i,j=0,0}^{81,81} sim(dist[n,i], dist[n,j]) * I[i,j \in \text{same row, col, or 3x3}]$$

where the similarity function can be any function that calculates the similarity/divergence between two vectors/distributions. In this project, cosine similarity and KL-divergence is used:

$$\text{cosine\_similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|}$$

$$\|\mathbf{A}\| = \sqrt{\sum_{i=1}^{n} A_i^2}, \quad \|\mathbf{B}\| = \sqrt{\sum_{i=1}^{n} B_i^2}$$

$$D_{\text{KL}}(P\|Q) = \sum_i P(i) \log \frac{P(i)}{Q(i)}$$

where $\mathbf{A}$, $\mathbf{B}$, P and Q are all cell distributions.

6.4.2 Experiment Design and Results

The SolverNN model is trained on big_kaggle and minimal_17, respectively, with out RL refinement. In addition to the training boards and label boards, a contrastive constraint regularization loss is used in the training process. Results show that this method can perform on par but not exceed the performance of the regular training method. Additionally, the training time doubled and tripled due to amount of computation required to compute the loss, which falsifies the Hypotheses for Research Question 4.

7 Conclusion

After careful experiments and investigation, it was determined that the poor performance of the Mask Predictor was due to the original author's errors in performance evaluation and that the Mask Predictor performs exceptionally well at identifying solution errors.

The decrease in SolverNN confidence had varying effects on Mask Predictor's confidence and various performance and the hypothesis are partially verified and falsified. The phenomena where Mask Predictor confidence decreases as SovlerNN confidence decreases only holds for big_kaggle-trained model, with minimal_17 trained model remain highly confident in its prediction. This may be attributed to the fact that minimal_17-trained Mask Predictor masks out all solutions regardless of SolverNN's confidence, as show in the graphs. Additionally, manipulating SolverNN's confidence in only correct solution cells has limited effect to Mask Predictor's confidence. The minimal_17 trained Mask Predictor remains highly confident and high performing, and the big_kaggle-trained Mask Predictor's confidence decreased slightly. Interestingly, when decreasing SolerNN's confidence, both big_kaggle and minimal_17-trained Mask Predictors exhibited desired qualities of a error predictor. As SolverNN's confidence in error solutions decreases, the big_kaggle-trained mask-predictor becomes uncertain about the error solutions it missed, and the minimal_17-trained Mask Predictor increased confidence in the error solutions it masked.

It was also discovered that NASR model are not robust to out-of-distribution sudoku boards, implying that NASR is not learning proper sudoku rules for generalization. Despite the poor peformance, the model trained on a challenging dataset, minimal_17, is more robust than the model trained on an easier dataset, big_kaggle. In the attempt to leverage sudoku rules during training, both consine-similarity and KL-divergence-based contrastive regularization loss failed to improve SolverNN performance and tripling the training time needed.

References

[1] Dhananjay Ashok, Joseph Scott, Sebastian Wetzel, Maysum Panju, and Vijay Ganesh. Logic guided genetic algorithms, 2020.

[2] Céline Brouard, Simon de Givry, and Thomas Schiex. *Pushing Data into CP Models Using Graphical Model Learning and Solving*, pages 811–827. 09 2020.

[3] Cristina Cornelio, Jan Stuehmer, Shell Xu Hu, and Timothy Hospedales. Learning where and when to reason in neuro-symbolic inference. In *The Eleventh International Conference on Learning Representations*, 2023.

[4] Paolo Dragone, Stefano Teso, and Andrea Passerini. Neuro-symbolic constraint programming for structured prediction, 2021.

[5] Maxime Mulamba, Jayanta Mandi, Rocsildes Canoy, and Tias Guns. Hybrid classification and reasoning for image-based constraint solving, 2020.

[6] Jingyi Xu, Zilu Zhang, Tal Friedman, Yitao Liang, and Guy Van den Broeck. A semantic loss function for deep learning with symbolic knowledge, 2018.