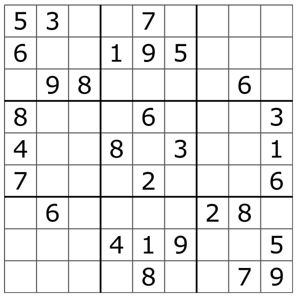
Solving Sudoku with Neural Networks

Trieu Hung Tran, Grahm Manley
Department of Computer Science and Engineering
{ttran|gmanley}@cse.unl.edu

***Abstract*—We investigate the application of Neural Network models on logical inference, more specifically, Sudoku puzzles. The task of solving a Sudoku puzzle of size N is known to be NP-Complete and is usually solved using Constraint Processing approach. As part of our project, we adopt and implement a Neural Network approach to solve Sudoku puzzles using simple architectures of dense and convolutional layers. We train the models using staged learning and evaluate them on three different datasets containing puzzles covering a wide range of difficulty. Finally, we conclude that our approach is not good enough given its limitations and propose suggestions for future improvement.**

I. Introduction

SUDOKU is a puzzle consisting of 81 cells organizing in a nine by nine grid made up of nine three by three sub-sections. Each cell can have an integer from one to nine subjecting to the constraint that each integer is unique to its row, column, and sub-section. The unsolved puzzle as illustrated in Fig. 1 contains some cells already filled with integers while the remaining are left blank to fill in. Because of its simplicity, Sudoku puzzles are very popular and can easily be found in many magazines or the Internet. For a very long time, it has been used as a benchmark for Artificial Intelligence (AI) algorithms, especially in the field of Constraint Processing.



5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

Fig. 1. An example of a Sudoku puzzle. [Image from Wikipedia]

Machine Learning (ML), on the other hand, has been rising from a sub-field of AI and eventually becomes synonymous or even larger than its parent. Thanks to the invention of better hardware, Deep Learning and Neural Networks, sub-fields of ML, have recently become among the most powerful techniques with various applications outperforming state-of-art solutions in many areas. Following that trend, we investigate the application of Neural Networks on classic AI problems, more specifically, in logical inferences starting from the well-known Sudoku puzzles.

II. Problem Definition

Usually, a Sudoku puzzle is generated so that one and only one solution exists. However, there are some exceptions when a Sudoku can have more than one solutions. Still, the basic techniques of solving a Sudoku puzzle remain the same. The state-of-art technique of solving a Sudoku puzzle is to utilize Constraint Processing techniques including Consistency Enforcing algorithms and Search. Moreover, with strong enough Consistency algorithms, the hardest known Sudoku instances can even be solved without search [1]. We are curious to see if ML can also be applied to solve Sudoku puzzles.

The task of solving a Sudoku puzzle can be modeled as a classification problem of all 81 cells with the accepted classes ranging from one to nine. The first inspiration for this idea comes from the fact that a Sudoku puzzle can be interpreted as a nine by nine image while Deep Learning and Neural Networks have been proven to be among the most effective techniques in image-related problems, i.e., with Convolutional Neural Networks. The second inspiration is from related works in [2] and [3] where they achieved promising results of 86% and 99.76% accuracy, respectively, on a large dataset with simple network architectures. We adopt some of their approaches and would like to see if we can achieve similar results along with further evaluation.

III. Datasets

We use three different datasets for our implementation and evaluations of the Neural Network models. All of them use the same string presentation of the Sudoku puzzle which consists of 81 characters representing each of the cells. Each character is a digit from zero to nine in which, zero represents a blank cell while other digits represent a cell that has been filled. A solution is also presented in the same format with all non-zero digits. Different datasets come with different sizes and additional features that can be used during data analysis and evaluation. They are detailed below.

A. Kaggle

The primary dataset that we use during the training process is obtained from Kaggle at [4] with one million instances containing a pair of a Sudoku puzzle and its solution. According to the author, the puzzles and solutions were created using the Sudoku generator from [5] written in Python in `numpy` array format before being converted to strings for CSV file exporting.

B. 17-given Puzzles

Unlike the Kaggle dataset, we use this dataset for evaluation purpose only, and it contains only Sudoku puzzles without providing any solution. The dataset is obtained from [6] and consists of 49,151 puzzles with exactly 17 clues given for each puzzle. Puzzles with 17 clues are called the minimum Sudokus since the existence of 16-clue puzzles has been disproved in [7]. However, they are not necessarily the hardest puzzles. We use this dataset mainly to add some challenges to the evaluation of Neural Network models because the first dataset mentioned above is shown to be not enough. Detailed analysis and results are described in Sections V and VIII, respectively.

C. UNL Sudoku

Following the usage of the 17-clue puzzles, we also use the dataset mentioned in [1] to further integrating the performance of the new approaches with existing Consistency algorithms in Constraint Processing for comparison. This dataset is currently used for the UNL Sudoku Solver website at http://sudoku.unl.edu/SudokuSet/SudokuSetV5/. It is provided in JSON format consisting of 472 instances with 13 features each. Besides the string presentation of the puzzle itself, each instance also contains additional features such as the source where the puzzle is obtained from, the number of clues, the number of solutions, or the Consistency Enforcing algorithm that can solve the puzzle without search. Among all additional features, only the number of clues and the Consistency algorithm are used later in data analysis and result evaluation.

IV. Data Preprocessing

As mentioned above, the Sudoku puzzles from the chosen datasets are provided in their string presentation which is inconvenient for the process of calculating and finding hidden patterns in the puzzles. Hence, they are all reformatted by applying three preprocessing steps described as follow.

1) The string presentation of each puzzle or solution is split into an array of length 81 where each element is an integer from zero to nine converted from a single digit character from the string.
2) Next, each array is further reshaped to a two-dimensional grid of size nine by nine simulating the original shape of the puzzle. This step is applied to preprocess the array with the expectation that the models can recognize the relation between the values of the puzzle cells and their locations on the grid.
3) Finally, we apply one-hot encoding to each cell of the grid making it a vector of size ten in which, the value at position i is one if the cell is filled with the integer i, otherwise it is zero. For the solution, the one-hot vector for each cell is nine instead of ten because we exclude the possibility of the solution containing zero in its presentation. The purpose of one-hot encoding is to eliminate the implicit ordering between different values of a cell which may be misinterpreted by the models.

After preprocessed, the datasets have the final shape of $(N, 9, 9, C)$ where N is the size of the dataset and C is either nine or ten indicating the number of acceptable classes in a solution or a puzzle.

V. Exploratory Data Analysis

To further understand the chosen datasets, we analyze each of them on two main properties: the number of clues in each puzzle and their difficulty. Out of two properties, the latter was applied after we obtained the results to analyze and understand the best models' performance.

A. Number of clues

For each dataset, we calculate the number of clues given in each puzzle by counting the number of zeros. From there, we investigate their average, minimum, and maximum values summarized in Table I. There is nothing interesting to learn from the statistics of the 17-clue puzzles since all the values for average, maximum, and minimum are the same at 17 which is as expected. On the other hand, it is worth noticing that the Kaggle dataset has the largest average and minimum number of clues among all three datasets. In addition, the range from the minimum to the maximum is much smaller in the Kaggle dataset compared to the one from UNL Sudoku Solver while its size is significantly larger (one million compared to 472). This indicates that the Kaggle dataset may not cover a wide variety of Sudoku puzzles with a hypothesis that its puzzles tend to be easier than the ones given in the other two datasets.

TABLE I
Number of clues in each dataset.

Dataset	Kaggle	17-clue	UNL
Average	33.81	17.00	23.06
Minimum	29	17	17
Maximum	37	17	50

B. Puzzle difficulty

Rating Sudoku difficulty is a subjective problem; thus, there is no standard for that. However, there are many rating systems that are already implemented by evaluating the process of solving a specific Sudoku puzzle. To examine the difficulty of each puzzle, we utilize the C-program Sudoku Solver from [8] which can produce a score presenting the degree of difficulty for a given puzzle. The rating scale of the score produced by that solver is given below in Table II. According to the description from the solver, Hard and Very Hard puzzles will likely require trial-and-error to solve while Diabolical puzzles are reserved for masochists, savants, and automated solvers.

TABLE II
Rating scale for Sudoku puzzle difficulty.

Degree of difficulty	Score
Trivial	80 points or less
Easy	81 - 150 points
Medium	151 - 250 points
Hard	251 - 400 points
Very Hard	401 - 900 points
Diabolical	901 and up

We then feed the solver with the puzzles from all three datasets to produce the number of solutions and difficulty score for each of them. Since the running time of the solver is quite long (approximately 30 minutes for 50,000 puzzles), due to time constraint, we cannot feed the whole Kaggle dataset to the solver but a random sample of 50,000 instances instead. We choose 50,000 because it is approximately the size of the 17-clue dataset. The number of solutions is requested for verification purpose from which, we figure out that except for the Kaggle and 17-clue datasets, some of the puzzles from the UNL Sudoku Solver website may have more than one solutions with a maximum of eleven solutions for a single puzzle. The difficulty scores are shown in Table III. According to the result, although both the Kaggle sample and the 17-clue dataset have approximately the same number of instances, their difficulties are significantly different. All puzzles in the sample from Kaggle dataset belong to the Trivial and Easy category which may not require too complex techniques to solve. On the other hand, both the 17-clue and UNL datasets cover a wider range of difficulty from Trivial and Easy to Diabolical puzzles. We understand that the random sample of 50,000 instances from the Kaggle data may not represent the whole dataset. However, given that the sample is generated randomly from the dataset, it may give some hints about the true distribution of the puzzles, which indicates that we may not be able to draw any significant conclusion about the performance of the new models using the Kaggle dataset alone.

TABLE III
DIFFICULTY SCORE OF PUZZLES IN EACH DATASET.

Dataset	Kaggle sample	17-clue	UNL
Average	47.20	1,178.82	390,014.49
Standard deviation	1.02	10,386.98	422,457.61
Minimum	44	144	41
Maximum	119	881,806	2,797,500

VI. EVALUATION METRIC

We evaluate the models' performance using the accuracy of the classification of all empty cells. For a Sudoku puzzle, a prediction is considered to be correct if all blanks are filled correctly. Even if there is only one incorrect cell, the whole prediction is said to be incorrect. Using that intuition, we summarize the percentage of correctly predicted puzzles from each dataset for comparing, analyzing, and drawing conclusions.

VII. METHODS

As mentioned in Section III above, we only use the Kaggle dataset for training and keep the others for evaluating the performance of the models. We design a staged learning approach to train four models differentiate in the Neural Network architectures and validation techniques. We also apply two regularization techniques including dropout and early stopping to prevent the models from over-fitting. Besides careful consideration of some hyper-parameters, most of the hyper-parameters are also selected considering the time it takes to train the models since Neural Network models usually take longer for the training to complete. The detailed process is described in below sub-sections.

A. Models

The main Neural Network model that we use consists of a hidden structure fed with an input of shape $(N, 9, 9, 10)$ obtained from the preprocessing step and a list of outputs. The hidden structure passes the input through two pairs of a hidden layer and a dropout layer before flattening it out to obtain a flat array. The array is then passed through 81 dense layers with softmax activation function to produce the output consisting of 81 one by nine probability arrays, each represents the probability of selecting one value from one to nine as the value of a specific cell. These dense layers ensure that the prediction of each cell takes into account the input from all other cells through which, it is expected to figure out the relationship between them. Since the output is the list of probabilities of multiple classes, the loss function that is used is categorical cross-entropy function calculated on each of the 81 probability arrays. The categorical cross-entropy function is the generalized version of the binomial cross-entropy function by summing over the losses of all classes and can be used on multi-class classification problems.

The hidden layer is either a dense layer or a convolutional layer. They are the two basic Neural Network layers which differ only from the way the neurons connecting to each other. In short, for the dense layer, each neuron in the output connects to all neurons in the input while for the convolutional layer, each neuron in the output only connects to a subset of neurons in the input determining by its parameters, i.e., the kernel size. Both the number of units for hidden dense layers and hidden convolutional layers are set to 64. Hence, the output shapes of them are exactly the same. This number is adopted from [3] since the author achieved an excellent result (99.76% accuracy) on the same dataset. In addition, although the idea of using convolutional Neural Network is inspired from [2], the mentioned architecture is very large with ten convolutional layers of 512 filters each which is definitely too much considering the scope of this experiment. However, we still adopt other hyper-parameters including "same" padding and a kernel size of 3. The output from each hidden layer is also passed through a rectified linear unit (ReLU) to introduce non-linearities to the model. The hidden structure is summarized in Table IV.

TABLE IV
HIDDEN STRUCTURE ARCHITECTURE.

Layer	Output shape
Dense or Convolutional	$(N, 9, 9, 64)$
Dropout	$(N, 9, 9, 64)$
Dense or Convolutional	$(N, 9, 9, 64)$
Dropout	$(N, 9, 9, 64)$
Flatten	$(N, 5184)$

B. Training

Because the Kaggle dataset is large, training on even 80% of it is still time-consuming. We decide to use only a subset of it containing 50,000 instances only for training, validating, and temporarily testing. The rest consisting of 950,000 other instances is used for final evaluation. The sample of 50,000 instances are the same sample that is used for puzzle difficulty analysis in Section V above. Out of 50,000 instances, 32,000 of them are used as the training set, 8,000 are used as the hold-out validation set, and the rest 10,000 instances are used as a temporary test set. We choose to use a single hold-out validation set instead of the k-fold technique because the dataset is quite large and learning in Neural Network takes a longer time. Therefore, k-fold validation may not be possible given the time constraint. We take the risk of less accurate estimating the result.

1) Staged Learning: Instead of feeding one training set with puzzles of different difficulties for multiple epochs to the models, we adopt the stage learning approach from [3]. The general idea is to feed the model with puzzles of increasing difficulty over time with the expectation that the model can learn from the easiest to the hardest in order leading to better performance. For simplicity, we assume that puzzles with more clues tend to be easier to learn and to solve. With that approach, we no longer need the puzzles from the training set but instead passing the solutions with randomly deleted cells to the model. By doing that, we not only control the number of blanks in the training set but also train the model on more instances. As we increase the number of blanks in the puzzles, we also increase the number of epochs to prevent the model from over-fitting since learning how to solve puzzles with fewer blanks is easier according to our assumption. A pair of one number of blanks and one number of epochs is called a stage. To limit the number of stages, we further increase the difference between two consecutive stages from one to five and end up with 19 stages in total. Throughout our experiments, we figure out that for later stages, the model never exceeds ten epochs because of early stopping; thus, the maximum number of epochs is set to ten. Detailed configuration for staged learning is illustrated in Table V below.

TABLE V
STAGED LEARNING CONFIGURATION.

Order	#blanks	#epochs
1	0	1
2	1	1
3	2	2
4	3	3
5	5	5
6	7	7
7	9	9
8	12	10
9	15	10
10	20	10
11	25	10
12	30	10
13	35	10
14	40	10
15	45	10
16	50	10
17	55	10
18	60	10
19	64	10

Based on the staged learning idea, we come up with an approach of applying similar modification to the validation set. Our hypothesis is that by validating the model on the validation set that has puzzles with the same number of blanks as the training set, due to the early stopping technique which is discussed in the following sub-section, the model may stop earlier and less prone to over-fitting because it does not need to take into account different difficulty levels at every stage. Therefore, we try both approaches of using the same validation set or using staged validation set to verify our hypothesis.

2) Regularization: During the training process, we apply two different regularization techniques including dropout and early stopping to prevent our models from over-fitting.

- Dropout: The general idea of dropout is to randomly "forget" a portion of the neurons in the transition between two consecutive layers by setting the outputs to zero. Since a Sudoku puzzle is not complex with a small shape of (9, 9, 9), it is easy for a model to over-fit the training data. For generalization purpose, we want the model to remember what it learns from previous stages but not everything. We decide to use the dropout rate of 0.4 which means allowing the model to "forget" 40% of the previous layer outputs. Since we do not do hyper-parameter tuning on the dropout rate, that value is adopted from [3] because it is shown to be effective.
- Early stopping: Early stopping technique is also applied to stop the training process after two epochs of no improvement on the validation set. We choose two as the patience number after considering the number of epochs in each stage. Since the number of epochs for each stage is small, choosing a large patience might not be effective. In addition, the patience of two is also shown to be effective from [3].

VIII. RESULTS

Inspired by the staged learning approach, we also use a different way to generate the prediction for a Sudoku puzzle. Instead of generating the predictions for all blank cells at once, we only predict one cell with the highest probability at a time before feeding the new grid to the model again to predict the following cells. With this approach, we expect the model to produce more consistent predictions because it can take into account its previous predictions. Because there may exist puzzles with more than one solution, a prediction does not need to be exactly the same as the provided solution but is said to be correct if it satisfies the requirement of a Sudoku for a corresponding puzzle, meaning filling all blanks so that the value in each cell is unique to its row, column, and sub-section.

A. On validation set

During the training process, we record the total loss and accuracy of the model at each stage on the validation set. We also have the total loss of the model at each stage on the training set because calculating the accuracy on the training set takes a lot of time with our predicting technique mentioned above. The results are then plotted to select the best model among four different ones. The total losses of four models on the validation set are illustrated in Fig. 2 and 3 in which Dense and Conv indicate the architecture and the suffix _Val indicates that the model uses staged validation set.

According to the graphs, all models converge to produce approximately the same loss on the training and validation set grouped by the usage of the validation set. The trending of the total losses of all four models on the training set is as expected because for puzzles with fewer clues, the losses should be larger because it is harder to predict. Similarly, the total losses of the two models using the staged validation set behaves the same with slightly larger losses compared to the training ones. That also makes sense because the model is trained on the training set; hence, it is expected to perform better. On the other hand, the total losses for models using the same validation set decrease over stages indicating that the models are improving after each stage. However, the losses start flattening out when the number of blanks reaches around 30 to 40. From there, it looks like the models have been near the local optimum and the following stages of training are not as effective as the initial ones.

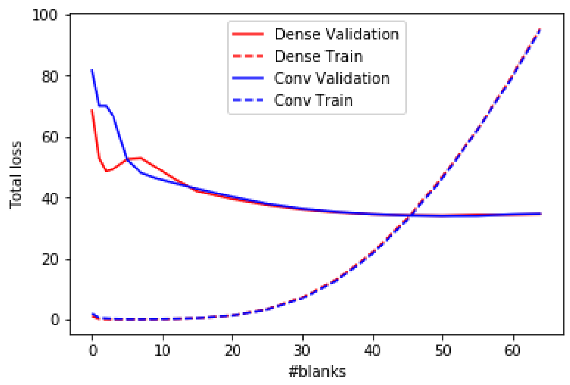


Fig. 2. Loss over number of blanks for models with the same validation set.

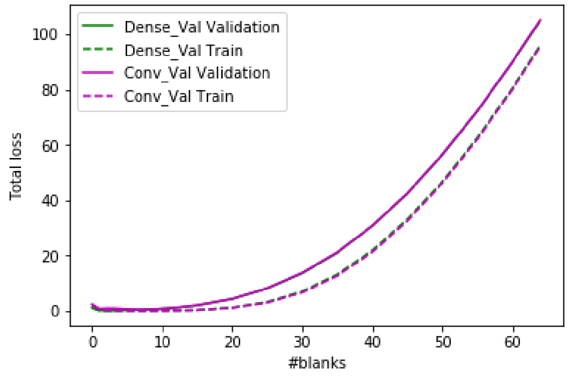


Fig. 3. Loss over number of blanks for models with staged validation set.

The same behavior can be observed from the plot of accuracy over stages illustrated in Fig. 4. The accuracy is calculated on the same validation set for all models because we need to estimate their performance on the unseen testing data. Using the validation set with the same number of blanks for all puzzles cannot serve that purpose. According to the figure, the accuracy of all models on the validation set also start to flatten out when the number of blanks reaches around 30 to 40. It is also worth noticing that although the convolutional model which uses staged validation set has the highest accuracy before losing its place to the dense model with staged validation set at around the same set of stages. The same behavior is also observed for models using the same validation set. It shows that dense models learn better on puzzles with more blanks while convolutional models perform better with puzzles having fewer blanks. Moreover, models using staged validation set also tend to perform better compared to models using the same validation set for all stages. Examining the number of epochs at each stage that all models are trained on from Fig. 5, our previous hypothesis in Section VII seems to be correct when models with staged validation set train on fewer epochs compared to their rivals.

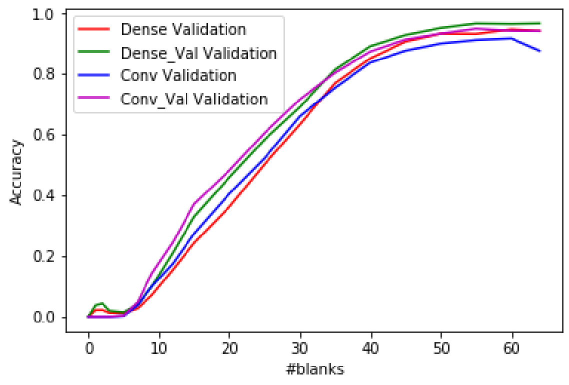


Fig. 4. Accuracy over number of blanks for all models with the same validation set.

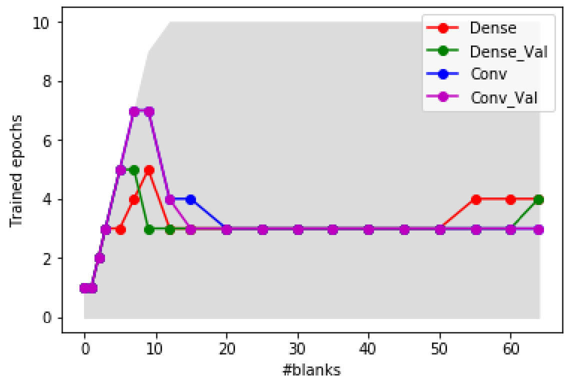


Fig. 5. Number of epochs over number of blanks for all models.

The common behavior for evaluations using the same validation set for all stages is that they all struggle a little bit after some initial stages when the number of blanks is around three to five. One possible explanation is that it is the stage when the number of blanks is doubled (from one to two) and the models had not got used to it yet. Hence, the total losses increase and the accuracy decreases. After a few stages, the models start to improve again.

Although the dense model with staged validation set has the highest accuracy, we cannot conclude that it is the best among all four models because we only calculate on one validation set. In addition, the differences between all models' performance are not significant. Therefore, we decide to use all four models for the evaluation on test sets.

B. On test sets

We first evaluate four models on the 10,000-instance temporary test set and obtain similar result as on the validation set. Details are in Table VI in which dense model with staged validation set performs the best, convolutional model with the same validation set performs the worst, and the other two models perform similarly to each other. The accuracy is approximately the same as what the author originally has from [3]. The only significant difference in our implementation is that we use a different stage configuration. To further verify if the model performs similarly on the whole dataset using only 50,000 out of one million puzzles, we test the best model - dense model with staged validation set against the unseen 950,000 instances and obtain an accuracy of 96.54%. This indicates that puzzles in the Kaggle dataset do not vary much in terms of difficulty which also strengthens what we have from the analysis in Section V.

TABLE VI
MODELS PERFORMANCE ON DIFFERENT TEST SETS.

Model	Kaggle temporary	17-clue	UNL
Dense	94.31%	0.045%	3.39%
Dense_Val	96.37%	0.037%	4.66%
Conv	87.76%	0.047%	2.97%
Conv_Val	94.43%	0.053%	3.18%

In addition, we test all models on the 17-clue dataset and the dataset from UNL Sudoku Solver website. The results are also consistent with what we have in Section V. All models perform poorly on the two additional datasets as expected because they contain harder puzzles compared to the Kaggle dataset. Further analyzing the result from UNL Sudoku Solver website, we figure out that most of the puzzles solved by our models can be solved by the easiest Consistency algorithms. Fig. 6 shows the number of puzzles that are predicted correctly by at least one of our models grouped by the Consistency algorithm that can be used to solve the puzzle without search. The algorithms are listed in the increasing order from left to right. The Unsolved category consists of puzzles with more than one solution. These puzzles must require search to figure out all solutions; thus, cannot be solved by any Consistency algorithm. Our models although are able to predict one puzzle in that category correctly, we can predict only one solution and cannot be counted. It is obvious that the solvable puzzles are the easy ones that can be solved using simple Consistency algorithms such as AC, GAC, or SAC.

The limitation may come from multiple reasons. First of all, the models themselves are too simple to capture more complex patterns. Implementing grid search on possible hyper-parameters along with better hardware may produce better models. Secondly, the training set does not cover a wide range of puzzle difficulties; hence, models cannot learn from the hard ones. Some people can argue that the models will eventually train on harder puzzles since the number of blanks increases across stages. However, that is only correct if our assumption about the proportionality of the number of blanks and puzzle difficulty is correct. However, it is less likely to be true since 17-clue puzzles also cover a large range of difficulties from 144 to 881,806 according to Table III in Section V.

IX. Conclusion and Future work

In conclusion, we have successfully implemented four Neural Network models for solving Sudoku puzzles and they

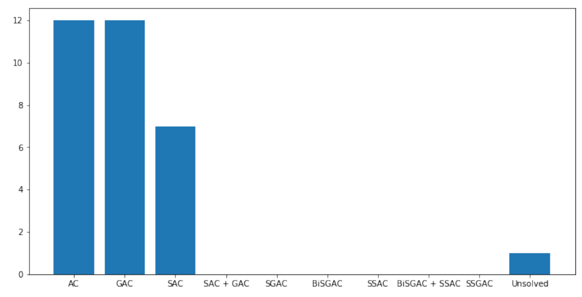


Fig. 6. Number of solvable by at least one of our models over Consistency algorithms.

all work as expected. We realize that the Kaggle dataset is not good to use for training and evaluating because of its limitation in puzzle difficulty variation. However, the Neural Network approach is still promising because it can correctly predict the solution for some of the easy Sudoku puzzles after being trained for only a small number of epochs on a less-variant dataset. It is also worth trying to address the problems mentioned in Section VIII and examining how the results differ. Staged learning also seems to be effective and can be improved by grouping puzzles by the “actual” difficulty instead of the number of blanks. Two of many possible options for determining a puzzle difficulty are either using the difficulty score as mentioned above in Section V or using the Consistency algorithms as provided from the UNL Sudoku Solver.

One thing that is worth noticing is that our solution does not generalize well to other variations of different size in the class of Sudoku puzzles because the models require a fixed input shape and produce a fixed output shape. One potential future improvement is to re-design the models so that they can be used for Sudoku puzzles of various sizes. In addition, ones may try implementing the Recurrent relational network proposed in [9] and compare the results. It is one of the papers that we are interested in and planning to work on in the future.

References

[1] I. Howell, R. J. Woodward, B. Y. Choueiry, and C. Bessiere, “Solving Sudoku with Consistency: A Visual and Interactive Approach,” in *Proceedings of the 27 th International Joint Conference on Artificial Intelligence*, 2018, pp. 5829–5831.

[2] K. Park, “Can Convolutional Neural Networks Crack Sudoku Puzzles?” Dec 2016. [Online]. Available: https://github.com/Kyubyong/sudoku

[3] Dithyrambe, “Neural nets as Sudoku solvers,” Dec 2017. [Online]. Available: https://www.kaggle.com/dithyrambe/neural-nets-as-sudoku-solvers

[4] K. Park, “1 million sudoku games,” Dec 2016. [Online]. Available: https://www.kaggle.com/bryanpark/sudoku

[5] A. L. Cordero, “Arel’s sudoku generator.” [Online]. Available: https://www.ocf.berkeley.edu/~arel/sudoku/main.html

[6] G. Royle, “Minimum sudoku.” [Online]. Available: http://staffhome.ecm.uwa.edu.au/~00013890/sudokumin.php

[7] G. McGuire, B. Tugemann, and G. Civario, “There is no 16-clue Sudoku: solving the Sudoku minimum number of clues problem via hitting set enumeration,” *Experimental Mathematics*, vol. 23, no. 2, pp. 190–217, 2014.

[8] B. DuPree, “Sudoku Solver in C.” [Online]. Available: https://github.com/attractivechaos/plb/blob/master/sudoku/incoming/sudoku_solver.c

[9] R. Palm, U. Paquet, and O. Winther, “Recurrent relational networks,” in *Advances in Neural Information Processing Systems*, 2018, pp. 3368–3378.