Sudoku Studying

Lena Melnikova

Optimization Class Project. MIPT

Introduction

In this project, it is planned to use several methods of automatic Sudoku Solving - using a neural network, using backtracking and using the Naked Twin approach. The accuracy and speed of these methods will be compared to draw the conclusions about which method is better to use for this task.

Sudoku rules

Sudoku is a logic-based, combinatorial number-placement puzzle. The objective is to fill a 9×9 grid with digits so that each column, each row, and each of the nine 3×3 subgrids that compose the grid contain all of the digits from 1 to 9.
Main rule:
Each row, column, and subgrid should contain each number (1 to 9) exactly once.

Backtracking

Backtracking is the simplest algorithm that iterates through all possible options for filling Sudoku using recursion. It is guaranteed that the algorithm solves Sudoku correctly, but it works slowly because it uses the usual brute force.

```
def Solve(sudoku):
    $find = FindEmpty(sudoku)$   #looking for an empty cell
    if $find$ is None :
        return $True$   #sudoku is already filled
    else :
        $position = find$   #index of an empty cell
    for $i$ in range$(1, 10)$ :   #for each digit check whether it is possible
        if $Valid(sudoku,\ i,\ position)$ :   #to insert the digit 'i' at the position 'position' in sudoku 'sudoku'
            $sudoku[position] = i$   #insert digit
            if $Solve(sudoku)$ :   #recursion
                return $True$
            $sudoku[position] = 0$   #erasing a digit to try another one
    return $False$   #no digit to insert(incorrect filling)
```

Naked Twin approach

A "Naked Twin" is a set of two candidates located in two cells belonging to one common block: a row, a column, a square.
It is clear that the correct solutions to the puzzle will be only in these cells and only with these values, while all other candidates from the general block can be removed.
This heuristic is added to the usual brute force and greatly speeds up its work.

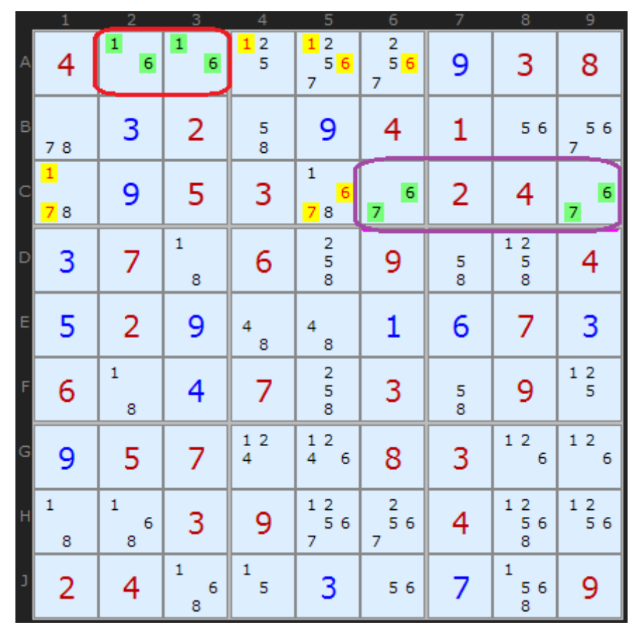


Figure 1: Naked Twin

Sudoku NN in PyTorch

Sudoku can be solved in the framework of a constrained neural network: The input sequence is the available clues, and the output sequence is a sequence of numbers to be entered in the empty cells to complete the puzzle. Predictions should be sequential since filling in numbers help you solve for the remaining empty cells. The problem is furthermore constrained due to the main rule.

Data

The labeled dataset which was used consists of nine million Sudoku puzzles, and two columns: one with the clues, and one with the solution. To handle this data in a network we transform each quiz to a matrix of size $(81, 9)$ where the numbers are one-hot-encoded, and empty cells are given by zero vectors.

Constraints

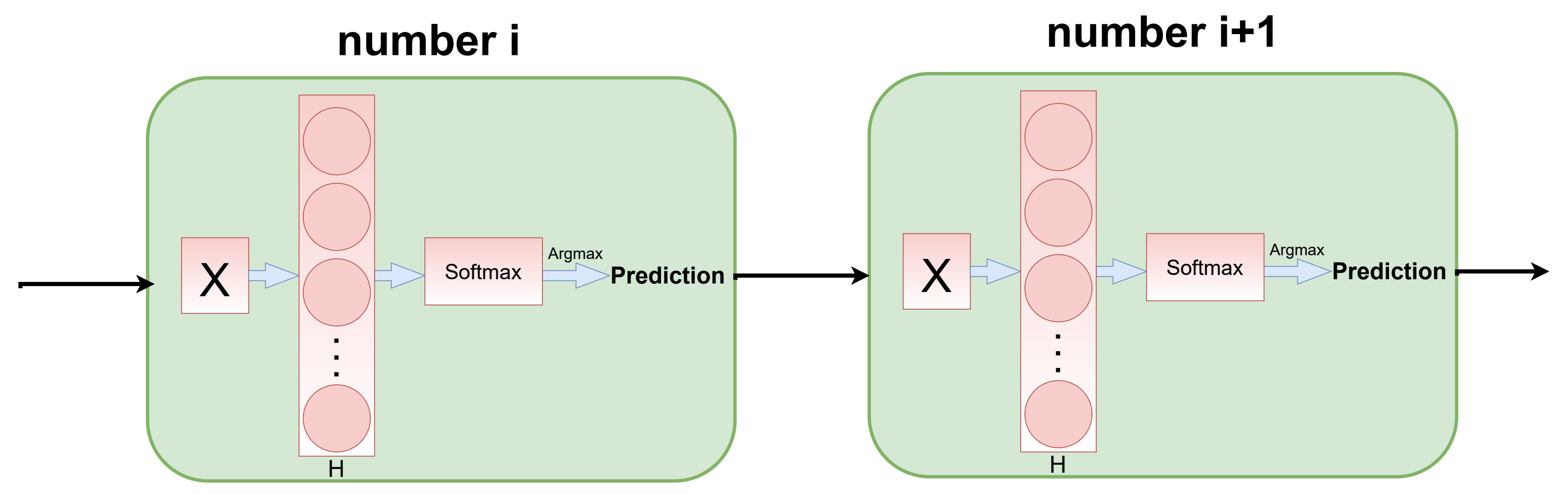
Constraints are represented with binary tensor of dimension $(81, 3, 81)$, where the first index enumerates the 81 cells of the puzzle, the second one enumerates the constraints (row column and subgrid). The last index enumerates the cells that constrain the cell in question.

```
...
ConstraintMask= $torch.zeros((81,3,81), dtype = torch.float)$
...
#subgrid constraints
for a in range(81):
    $r = a\ //\ 9$
    $c = a\ \%\ 9$
    $br = 3 \cdot 9 \cdot (r\ //\ 3)$
    $bc = 3 \cdot (c\ //\ 3)$
    for b in range(9):
        $r = b\ \%\ 3$
        $c = 9 \cdot (b\ //\ 3)$
    ConstraintMask[ $a,\ 2,\ br\ +\ bc\ +\ r\ +\ c$] = 1
```

The above loop will fill in a vector of length 81 with ones for cells constraining the cell in question by the subgrid rule, and leaves zeros everywhere else.

Network architecture

At each step we fill in the most probable number predicted by neural network based on already predicted sequence. To accomplish this we build is a basic MLP with one hidden layer. The input vector for scoring each cell is constructed from the constraint masks. This way the network knows what cells are filled and what numbers are possible to enter in the cell in question, without violating the constraint. The input vector for each empty cell is then run through the net.



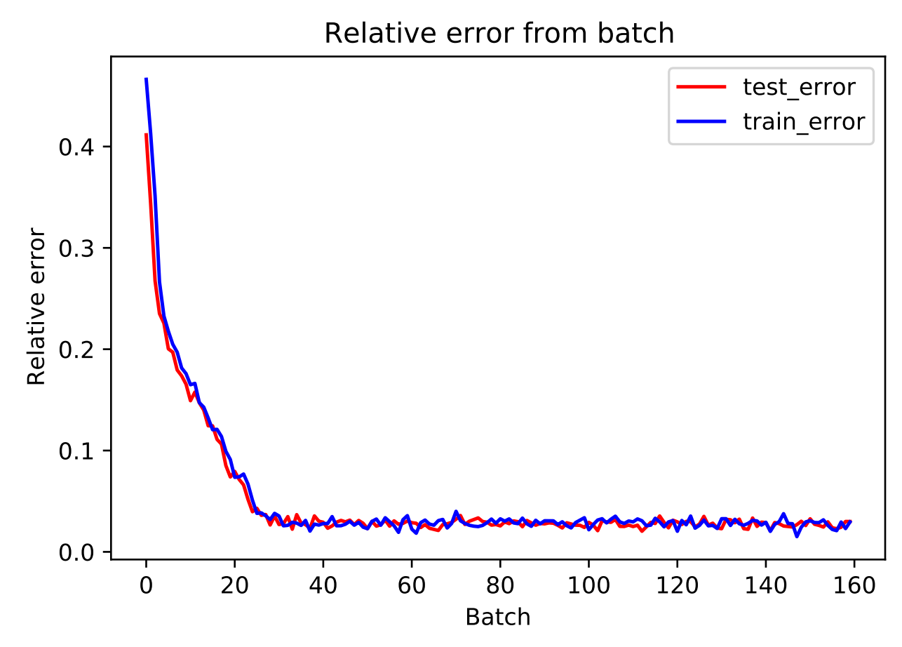
Results

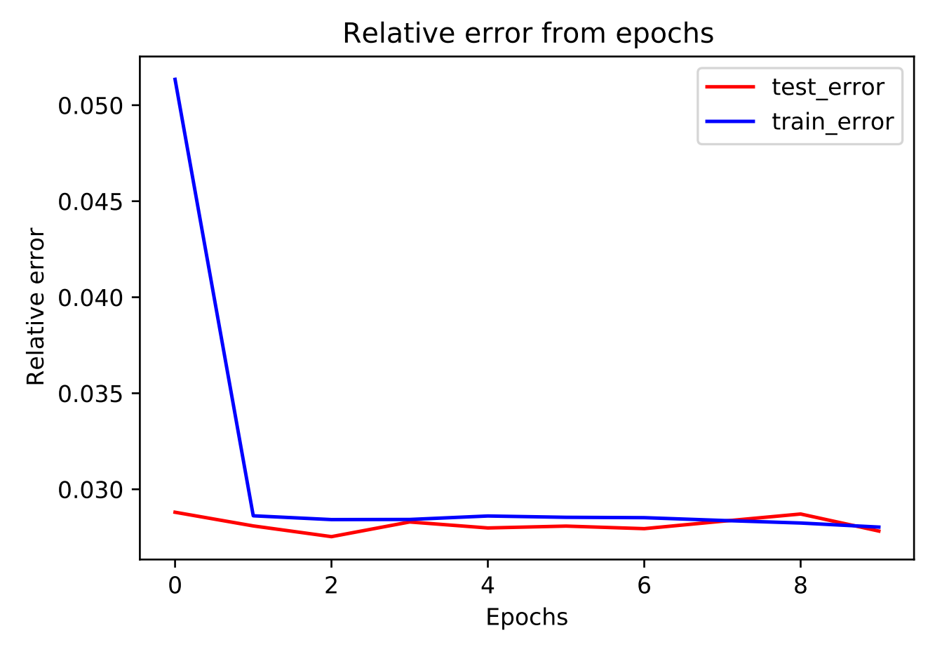
Quality of the algorithm is evaluated using two indicators:
The first is the accuracy in filling the cells (the number of correct cells to the number of all cells)
The second is the accuracy of Sudoku solution (the number of correctly filled Sudokus to the number of all Sudokus)

$$Accuracy_1 = \frac{\sum_{i=1}^{N}\sum_{j=1}^{M} \mathbb{I}\{y_{ij} == \hat{y}_{ij}\}}{NM} \cdot 100\%$$

$$Accuracy_2 = \frac{\sum_{i=1}^{N} \mathbb{I}\left(\sum_{j=1}^{M} \mathbb{I}\{y_{ij} == \hat{y}_{ij}\} == M\right)}{N} \cdot 100\%$$

where N is the dataset size, $M = 81$ is the number of cells in sudoku.
Neural network training:





Results (20,000 sudoku has been solved):

Algorithm	Percentage of correctly filled cells	Percentage of fully correctly filled sudokas	Speed of work
Backtracking	100%	100%	5min 7s
Naked Twin approach	100%	100%	1min 8s
Neural Network:	97.2%	87.2%	27.5 s

Conclusion

From the above experiments on solving Sudoku, it was found out that using backtracking Sudoku can be solved, but it will be very slow, as it uses multiple combinations as it proceeds. The neural network is faster than the usual brute force, but it does not work very accurately, so it is not an ideal candidate for this task. The best approach, according to this experiment, is the naked twins approach.

References

1. Program code.
https://github.com/ML-MountainLover/Sudoku_Studying
2. Deep Sudoku Solver (Multiple Approaches)
https://www.kaggle.com/datasets/rohanrao/sudoku
3. An Introduction to Convolutional Neural Networks, ArXiv e-prints, O'Shea, Keiron and Nash, Ryan, 2015
4. Sudoku NN in PyTorch
https://github.com/modulai/pytorch_sudoku