Sudoku Solver with Convolutional Networks

Junghwan Kim
junghwk11@gmail.com

Introduction

With the rise of AI and machine learning, logic or mind games such as Go and chess have caught huge attention in implementing AI models to master games. In 2016, AlphaGo, created by Google Deepmind, battled Korean Go professional Lee Sedol, demonstrating how dominating computer programs can be in these logic based games. Moreover, world chess champion and grandmaster Magnus Carlsen has shared his thoughts on chess models, saying he cannot beat chess engines even on his smartphone.

Similar to these logic based games, Sudoku is a combinatorial number placement puzzle where you need to strategically place digits from 1 through 9 once without overlapping in both rows and columns.

Motivated by dominating performance of AI models in other logic based games and Sudoku's nature in requiring logic based thinking, I was naturally drawn to using machine learning models, more specifically neural networks to solve Sudokus.

Background

While researching about this topic, I stumbled upon a paper from Stanford University's C230 class, where the authors used neural networks to solve Sudoku. In their paper *Solving Sudoku with Neural Networks*, Charles Akin-David, Richard Mantey explores using two deep learning architectures of Convolutional Neural Networks(CNN) and Bidirectional Long Short Term Memory(Bi LSTM) models to solve Sudoku. The paper also includes existing algorithms for solving Sudoku and also current implementation of neural networks in solving Sudoku by app called Magic Sudoku. Since this paper by Akin-David and Mantey already does extensive background research on Sudoku and implementing neural networks, this paper will focus on: reviewing their framework, recreating CNN, and discussing how the model can be improved.

Literature Review

As mentioned above, the paper ‘Solving Sudoku with Neural Networks’ by Charles Akin-David and Richard Mantey explores the use of deep learning techniques such as Convolutional Neural Networks (CNNS) and Bidirectional Long Short Term Memory (Bi LSTM) networks, to solve Sudoku puzzles.

Data

For their project, Akin-David and Mantey utilized Ariel Cordero’s Sudoku generator to create a 1 million Sudoku puzzle dataset. Each puzzle was paired with its solution, creating input-output pairs for supervised learning. Out of the million examples that were generated, 999,000 examples were used for training-dev sets. For their test, the authors sourced 100 examples sourced from the actual Sudoku book with unique solutions and evenly distributed in difficulty.

Methods

Akin-David and Mantey used two different deep learning architectures: CNN and Bi LSTM. For CNN, they built 16 layer architecture with 512 filters per layer and the final layer being a 1 by 1 convolution with 10 filters for classification. They also mentioned the model's architecture using inference to solve the whole sudoku board at once, with the softmax step adding soft probabilities to all cells across all labels.

For the Bi LSTM model, they used a set of 3 bidirectional LSTM, so in total of 6 LSTMs. Each LSTM has 200 hidden and memory units that were aimed to replicate the human approach to solving Sudoku puzzles. Outputs from these LSTMs were combined and passed through a dense layer to predict cell values. Similar to the CNN, the authors used softmax activation for classification.

Results

In short, the paper’s CNN outperformed the Bi LSTM in both training and test accuracy. For 3 layer LSTM, it had training accuracy of 0.814, test accuracy of 0.467, and average test F1 score of 0.465. In contrast, 15 convolutional layers (plus final layer) CNN had training accuracy of 0.997, test accuracy of 0.86, and average test F1 score of 0.898. The disparity in the accuracy suggests that CNNs are well-suited for solving Sudoku puzzles due to their ability to model spatial patterns efficiently. Moreover, the authors highlighted that LSTM showed no misclassification bias across labels, suggesting potential improvement with larger datasets or architectural adjustments.

Inference and Implementation

From reviewing Akin-David and Mantey's paper, I've decided to recreate the CNN model and possibly improve the model. Although the authors did share their repository, I've decided to attempt to build the model myself. Further information of my methodology and plan is explained below.

Experiment 1: Initial Model

Data

For this paper, I was able to find the Sudoku dataset from Kaggle. The dataset has two columns, 'puzzle' and 'solution' and 9 million rows. Both columns were stored in a string of integers, with the length of 81. For the 'puzzle' column, the solutions were replaced with 0s, while the 'solution' column was filled with 1-9 digits in place of 0s.

One of the best and direct ways to improve your model performance is to have a large dataset to train on. With 9 million entries in the dataset, in theory, this would lead to better performance from the model when trained. Unfortunately, I'm working with my laptop and datahub provided by the school which is not computationally rigorous enough to even preprocess the dataset.

With my attempt to one hot encode the solutions failing due to memory issues, I had to sample the dataset to 2 million. However, 2 million, or even 1 million was still too big for the kernel I was working with to fit the model, so I had to settle with 100,000 dataset initially.

After sampling the dataset, I converted both columns to np.array and reshaped them into the shape of (100000, 9, 9) to normalize the 'puzzle' column and normalize the one-hot encode the 'solution' column.

For data preprocessing after reshaping, I utilized sklearn's pipeline to preprocess the data (normalization and one-hot encoding) for encapsulation of the process.

Model

Similar to the referenced paper, for my CNN, the architecture consists of an initial convolutional layer that processes the input grid, followed by a sequence of convolutional layers that extract high-level spatial features. Each of the middle layers applies a 3 by 3 convolution with 512 filters, followed by batch normalization and a ReLu activation. Again for more concise implementation, I pipelined the model as well.

Results

After training and evaluating the model, it was returned with 73% test accuracy. Comparing it to Akin-David and Mantey's CNN's testing accuracy of 0.86, my model is outperformed. This result can be explained by the difference in the training size and possibly better CNN architecture by them.

Although this was expected, it is still an underwhelming performance by my model. To further develop and enhance my model, below is my attempt to do so.

Experiment 2: Enhanced Model

To enhance the initial model, below are some steps I've taken to enhance the model.

Data

For the data set, preprocessing, normalization, and one-hot encoding stayed the same. Instead, I increased the sample size to 200,000 instead of 100,000. With the increased dataset, I hope to achieve better model performance.

Model

To the model, I've made two big changes: increased model depth, and introduced residual connections.

First, I've increased the number of convolutional layers from 16 to 20 in the hope that deeper networks will extract more complex patterns from the puzzle. In general, higher depth can improve accuracy but also introduces the risk of vanishing gradients. Moreover, due to the change in the number of convolutional layers, the pipeline is also changed accordingly.

In order to avoid the vanishing gradient problem which can stop the training process of the model, I decided to add rectified linear unit activation into the model. For each ReLu Layer, the output is added to the residual connection from the previous layer.

Results

The enhanced model had a test accuracy of 79.55%. This is a 6% increase from our baseline model. Considering that the model was only trained and tested on 200,000 entries of dataset, and with only 4 more convolutional layers than the baseline model, it is promising to see the increase in the test accuracy.

Limitations

Limitations in this research include: lack of computing power, small dataset size and lack of comprehensive model design. The lack of computing power that I had directly led to having to sample from the big 9 million dataset to only 200,000. Moreover, running the model was extremely time consuming and led to inefficiency and decreased working performance in conducting more thorough investigation, fine tuning, and more added features of the layer.

For better approach in the near future, implementation of regularization approaches such as drop out or pooling can increase the model performance. As mentioned in the Akin-David and Mantey paper, incorporating reinforcement learning can lead to higher performance as well.

Conclusion

Though overall my idea might be similar to the previous work of Akin-David and Mantey, I've tried to differentiate my research with taking a deeper approach into creating CNN in my own way and with strategies I believe will lead to better results of the model. Comparing the result of different dataset size, utilizing model pipeline, and implementation of ReLu activation compared to traditional CNN was my effort to approach the problem of solving Sudoku with neural networks.

Even with the effort of trying to perform better than the referenced paper with my approach to the model, the desired result was not achieved. However, the enhanced model showed improvements compared to the initial model, providing us with valuable insights in importance of dataset size and better model design.

Reference

- Akin-David, C., & Mantey, R. (2018). Solving Sudoku with Neural Networks. Stanford; Standford University.
- Kaggle sudoku dataset: https://www.kaggle.com/datasets/rohanrao/sudoku?resource=download
- Though not copied, I referenced COGS 181: Deep Learning HW4 network definition to help build my modelpy