CS230

Solving Sudoku with Neural Networks

Charles Akin-David, Richard Mantey
{aakindav, rmantey}@stanford.edu

ABSTRACT

Sudoku is one of the most popular logic-based games of all time, exploiting individuals' abilities for relational reasoning and pattern development. We explore the use of convolutional neural networks (CNN), and long short term memory (LSTM) networks in solving these puzzles. While the results show that both approaches to solving this problem are feasible, the CNN model is more computationally efficient, allowing for better training, consequently yielding higher training and test accuracies.

1. INTRODUCTION

Logic based games like Sudoku have been shown to help delay neurological disorders like Alzheimer's and dementia. In solving Sudoku puzzles, players rely on both the temporal and positional dependencies between the numbers on the board. Motivated by these semantics, we decided to attempt to solve Sudoku puzzles using CNNs and RNNs, - Bidirectional LSTMs - presenting us with the opportunity to explore relational and visual interaction networks. These methods use relational reasoning, which requires that the networks draw "logical" conclusions about the relationships between the numbers in the puzzle. The inputs to the networks were 81 length arrays representing the numbers on an unsolved 9x9 sudoku board. The outputs were the predicted 81 length array representations of the solved boards.

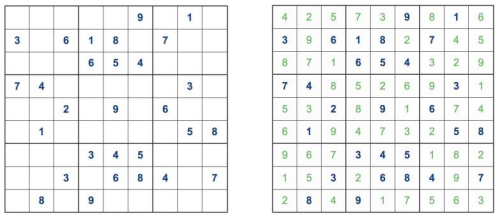

Figure 1. *Sample sudoku board showing quiz (input) and its solution (desired output)*

2. RELATED WORK

Search Algorithms: The most popular approaches to solving sudoku utilize tree search algorithms and heuristic search algorithms such as best-first search, A*, and backtracking. These methods allow for the incorporation of the rules of sudoku into the problem solving process as heuristics, increasing the likelihood of convergence on a solution. Convergence though comes at the heavy cost of computational speed , i.e. depending on the complexity of the quiz, some of these may take a very long time to converge on a solution. [1] [2]. Given this cost, there have been attempts to solve these puzzles using Machine learning approaches especially as testing times for ML models are usually less than the time needed to run an entire search algorithm.

Neural Networks: Currently there's an application on the iOS app store named *Magic Sudoku.* [3] This app solves sudoku using a convolutional neural network. This allows the application to be blazing fast while solving sudoku boards with up to 99% accuracy.

OptNet solves sudoku using a network architecture that integrates optimization in the form of quadratic problems. This allows the network to learn sudoku problems purely from data with high accuracy. The main drawback of this approach is that it has a cubic complexity to the number of variables and/or constraints which means that the number of hidden layers must be small in order to guarantee real-time predictions which compromises accuracy. [4] Recurrent Relational Networks obtain state-of-the-art results by solving 96.6% of the hardest sudoku puzzles. This approach is seeming to be the best approach for solving Sudoku puzzles, however the authors mention that Residual Networks like those used for AlphaGo may result in even higher accuracies. [5]

3. DATASET AND FEATURES

The dataset was formed using a sudoku generator, created by Ariel Cordero[6]. The generator first creates full sudoku (9x9) boards using the constraints that the numbers, 1-9, be unique in each row, column, and (3x3) subgrid. These fully generated boards were deemed as solutions and become the labels. To build the puzzles, the generator logically "plucks" numbers off the full solution boards ensuring that the removed number can still be deduced from the remaining numbers on the board. The plucked number is replaced with a 0 to represent a blank in the board. The plucking mechanism usually leaves 46-49 blanks in the board. The resulting quiz puzzle was used as the input, and, paired with the correlating solution, each (quiz, solution) pair was used as a training example. 1,000,000 examples were generated, 999,000 examples were kept for the training set and a 100 examples were set aside for the train-dev set. The test set contained 100 examples sourced from actual sudoku boards with unique solutions, evenly distributed in difficulty. Code was added to flatten out the (9x9) quiz, solution boards to strings of length 81 and to save these string pairs in a csv file. This file was read into the CNN model, which transforms them into 9x9 matrices. The LSTM converts the same 81 length strings into 81x9 One-hot label representations. The main features used were the labels representing the number for each cell on a sudoku board. The table below shows a representation of the quiz/solution training example from Figure 1.

Quiz	000009010306180700000654000740000 030002090600010000058000345000003 068407080900000
Solution	425739816396182745871654329748526 9 315328916746194732589673451821532 68 497284917563

Table 1. *Dataset Example*

4. METHODS

Two Deep Learning architectures were employed: Convolutional Neural Networks and Bidirectional LSTMs. Both methods used a final softmax layer to generate probability values over the K expected labels - K = (1 , 2, ... 9) - using:

$$P(C_k \mid x) = y_k(x) = \frac{e^{a_k}}{\sum_{j=1}^{K} e^{a_j}} \quad (1)$$

Where x represents the outputs from the previous layer, and $a_k = w_k^T x + b_k$, with w_k and b_k representing the softmax layer weights and bias for the class k. The argmax over the soft probabilities were then taken to choose the most likely class label for a given cell in the board. The loss for both models was calculated using the Cross Entropy Loss.

$$-\sum_{k=1}^{K} I(k,\ t_{class})\ log\ P(t = k|x) \quad (2)$$

For class labels t ∈ {1, ..., K}, we use an indicator function to denote whether or not the chosen label t is equal to class, k, and if so we sum the log probability of k equalling t given x. Finally, for learning, both architectures utilized the adaptive momentum estimation (Adam) optimizer with standard hyperparameters: α=0.001, β_1=0.9, β_2=0.9, ε =10^{-8}.

Require: α: Stepsize
Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates
Require: $f(\theta)$: Stochastic objective function with parameters θ
Require: θ_0: Initial parameter vector
$m_0 \leftarrow 0$ (Initialize 1st moment vector)
$v_0 \leftarrow 0$ (Initialize 2nd moment vector)
$t \leftarrow 0$ (Initialize timestep)
while θ_t not converged **do**
$t \leftarrow t + 1$
$g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep t)
$m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)
$v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)
$\widehat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)
$\widehat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)
$\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \widehat{m}_t / (\sqrt{\widehat{v}_t} + \epsilon)$ (Update parameters)
end while
return θ_t (Resulting parameters)

Figure 2. *Adam Algorithm*

Adam optimization involves using the first (m_t) and second (v_t) moments of the gradients. The first moment involves the exponentially decaying average of the previous gradients and the second moment involves exponentially decaying average of the previous squared gradients. The Adam optimizer then uses the combined averages of previous gradients at different moments to better adaptively update the parameters. [7]

4.1 CONVOLUTIONAL NEURAL NETWORKS

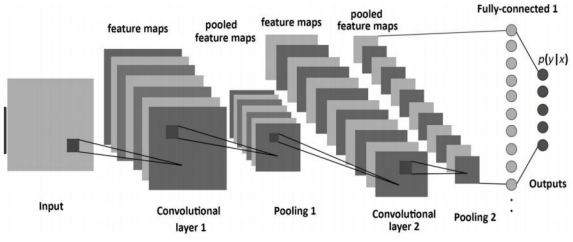

Figure 3: *CNN*

A 16 layer CNN was built with each of the first 15 convolutional layers having 512 filters, with the final layer being a 1x1 convolution with 10 filters. Before running the batched inputs through the CNN, the dimensions were expanded by one, i.e. (N, 9, 9, -1) to retain the (N, 9, 9) shape throughout the network. The extra dimension is changed depending on the number of filters for each convolutional layer. The 15 convolutional layers each used a filter size of 3x3, 'same' with 0 padding, pad size of 1, and a stride of 1. The 3x3 filter size allowed the model to focus on solving each 3x3 subgrid at a time. The 1x1 convolutional layer used 10 filters, changing the shape of the final dimension such that softmax could output 10 probabilities over labels 0-9. The argmax over the soft probabilities were then taken and the most probable label for the corresponding cell chosen. The model's architecture uses inference to solve the whole sudoku board at once, i.e. by virtue of the softmax step adding soft probabilities to all cells across all labels.

4.2 BIDIRECTIONAL LSTM

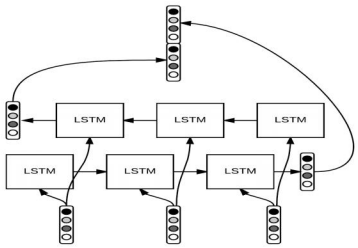

Figure 4. *Bi-LSTM*

The sudoku problem was also modelled using a set of 3 bidirectional LSTMs. Each LSTM had 200 hidden and memory units. the desire was to model the human approach to solving sudoku puzzles. Intuitively, to solve a cell, humans examine the row, column, and subgrid that the cell exists in. Consequently, quizzes were represented as sets of 3, 81x1x9 one-hot representations over the 9 possible labels. Each input was a concatenation across the rows,

columns, or 3x3 subgrids for a single quiz. The inputs were fed into their corresponding LSTMs i.e. the first LSTM took in a row representation, the second, a column representation, and the last, a subgrid representation all for the same quiz. The outputs from all 3 models were pooled together and fed into a feed-forward layer with 9 neurons to convert them back into a 81x1x9 vector. This output was then passed through a softmax layer to output probabilities values over labels 1-9, with the highest probability (argmax) for each cell chosen.

5. EXPERIMENTAL RESULTS/ DISCUSSIONS

The primary metrics used in evaluating the accuracy of the models were accuracy and F_1 score. Accuracy for both the CNN and LSTM were calculated as the percentage of correctly labelled cells, across a set of quizzes. F_1 score encodes the precision and recall. Precision is the proportion of all cells in a set of quizzes that the model labeled as 'k', which truly have the label 'k'. Recall is the proportion of cells, in a set of quizzes, that actually have the label 'k' and were accurately classified by the model as 'k'. Below is a summary of the final results:

Architecture	Training Accuracy	Test Accuracy	Average Test F_1 Score
1 layer LSTM	0.791	0.467	0.465
2 Layer LSTM	0.814	0.467	0.465
3 Layer LSTM	0.814	0.467	0.465
10 Conv CNN	0.97	0.78	0.856
15 Conv CNN	0.997	0.86	0.898

Table 2. *Summary of Results*

5.1 CONVOLUTIONAL NEURAL NETWORK

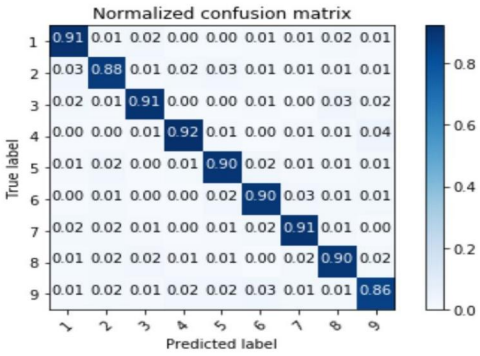

Figure 5. *Normalized Confusion matrix for 3 layered bidirectional LSTM(train acc. = 0.814, test acc. = 0.467)*

	Val	Test Acc	Val	Test Acc	Val	Test Acc
num epochs	3	0.74	5	0.76	10	0.78
conv layers	10	0.72	12	0.74	15	**0.81**
dropout	0.2	0.66	0.1	0.72	0.05	0.77
L2 lambda	1e-3	0.77	2e-3	0.73	1e-4	0.74

Table 3. *CNN Hyperparameter Tuning*

Table 3 shows hyperparameter tuning for the CNN with a base of 3 epochs and 10 conv layers. The test accuracy generally increases with more epochs, with the best model having been run using 15 epochs [Table 2]. Test Accuracy generally increased with the number of conv layers, however, after 15, the accuracy began to decrease which was due to overfitting. Dropout and L2 regularization did not lead to an optimal test accuracy so neither were included in the final model. This is because, the variance issue discovered from running the CNN model stemmed mainly from the different data distributions, as the training set contained mostly easy to medium puzzles while the test set contained an even distribution of puzzles of difficulties ranging from easy to expert. The confusion matrix [Figure 5], illustrates the absence of

misclassification bias in the CNN model. This suggests that this model can be leveraged for future improvements.

5.2 BIDIRECTIONAL LSTM

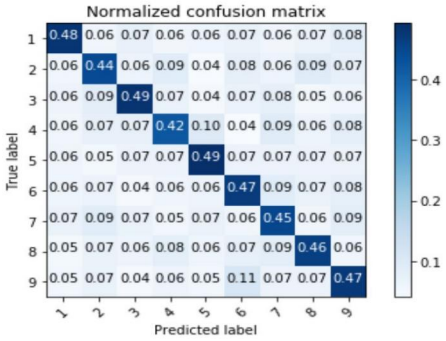

Figure 6. *Normalized Confusion matrix for 3 layered bidirectional LSTM*

1 to 3 layered LSTMs were used. [Table 2] It was discovered during training that it would take an impractical amount of time to train any one model over the entire 999,000 training examples. In order to deal with the computational costs, the models were run on 1000 training examples with a batch size of 1, over 100 epochs. While batching allowed for decreased computation time, for the same number of epochs, smaller batches yielded much higher accuracies. From the experiments, a marginal increase in training accuracy was observed as the number of layers were increased [Table. 2], illustrating the minute benefits, both in accuracy and computational speed, to using more than 2 layers. This suggests that given our current model configuration, an increase in the number of training epochs, or hidden dimensions, would be needed in order to improve train accuracy. The model also generalized equally across our test examples regardless of layer number i.e. 0.47 test accuracy. It is likely that, owing to the high bias problem, the model has barely learnt enough about the sudoku puzzles in our dataset to even attempt to generalize across real-world examples. Training on more data would likely resolve this. The normalized confusion matrix [Figure 6], shows that the model misclassified all classes about equally, suggesting that with better training, higher accuracies may be attained. Together, these results lead us to think that in order to truly perform well on this task we would need to incur the computational expense of training on a larger dataset, and that the human approach to solving sudoku puzzles may be inherently inefficient.

6. EXPERIMENTAL RESULTS/ DISCUSSIONS

In this study we have shown that it is possible to solve sudoku using Deep learning approaches. We employed the use of CNN and LSTM architectures and found that, the CNN approach was structured more efficiently and consequently performed better at solving sudoku quizzes with a training accuracy of 99.7% and test accuracy of 86%. Owing to the structuring of the LSTM problem, it yielded a lower training accuracy of 81% and test accuracy of 47%. The LSTM however showed promise in that it it did not possess misclassification bias. In order to improve on this work, the training set of the CNN needs more difficult puzzles. In the case of the LSTM, an increase in training set size and a reconsideration of the structure of the LSTM approach may yield better accuracies without compromising computational speed. Finally, an extension of both methods to either incorporate reinforcement learning with backtracking or recurrent relational networks will allow for higher accuracies.

References:

[1]Ace.ucv.ro, 2018. [Online]. Available: http://ace.ucv.ro/anale/2012_vol2/05_Nicolae_Ileana.pdf. [Accessed: 12- Feb- 2018]

[2]"Solving Sudoku Puzzles using Depth First Search", Dan's Website, 2018. [Online]. Available: http://logicalgenetics.com/solving-sudoku-puzzles-using-depth-first-search/. [Accessed: 20- Feb- 2018]

[3]"Behind the Magic: How we built the ARKit Sudoku Solver", Prototypr, 2018. [Online]. Available: https://blog.prototypr.io/behind-the-magic-how-we-built-the-arkit-sudoku-solver-e586e5b685b0. [Accessed: 20- Feb- 2018]

[4]Proceedings.mlr.press, 2018. [Online]. Available: http://proceedings.mlr.press/v70/amos17a/amos17a.pdf. [Accessed: 20- Feb- 2018]

[5]Arxiv.org, 2018. [Online]. Available: https://arxiv.org/pdf/1711.08028.pdf. [Accessed: 20- Mar- 2018]

[6]"Arel's Sudoku Generator", Ocf.berkeley.edu, 2018. [Online]. Available: https://www.ocf.berkeley.edu/~arel/sudoku/main.html. [Accessed: 23- Jan- 2018]

[7]Arxiv.org, 2018. [Online]. Available: https://arxiv.org/pdf/1412.6980.pdf. [Accessed: 20- Feb- 2018] - Adam

Contribution

Charles - CNN
Richard - Bi-LSTM
We both worked on data generating/cleaning, training model, analysis, background research, hyperparameter tuning, and analysis.

Github Repo: https://github.com/charlesakin/sudoku