Generation of Sokoban Stages using Recurrent Neural Networks

Muhammad Suleman, Farrukh Hasan Syed, Tahir Q. Syed, Saqib Arfeen, Sadaf I. Behlim, Behroz Mirza
Department of Computer Science
National University of Computer and Emerging Sciences, Karachi, Pakistan

***Abstract*—Puzzles and board games represent several important classes of AI problems, but also represent difficult complexity classes. In this paper, we propose a deep learning based alternative to train a neural network model to find solution states of the popular puzzle game Sokoban. The network trains against a classical solver that uses theorem proving as the oracle of valid and invalid games states, in a setup that is similar to the popular adversarial training framework. Using our approach, we have been able to verify the validity of a Sokoban puzzle up to an accuracy of 99% on the test set. We have also been able to train our network to generate the next possible state of the puzzle board up to an accuracy of 99% on the validation set. We hope that through this approach, a trained neural network will be able to replace human experts and classical rule-based AI in generating new instances and solutions for such games.**

***Keywords*—stepwise cooperative training; generative networks; recurrent neural networks; Sokoban; puzzles; deep learning**

I. Introduction

Solving board-games and puzzles is often NP-hard and therefore arriving at a solution is not computationally feasible. Many strategies for solving board games have been developed using AI techniques. One important approach is to change board logic into a constraint satisfaction problem (CSP) and then solve the CSP using a theorem prover. Although most theorem provers perform optimizations for specialized cases, the generation of solution by theorem provers still remain computationally infeasible for higher order problems. In this paper we investigate whether we can speed up this process for solving board games using neural networks. The basic idea is to generate solved stages using the theorem prover and use this stage data to make training and testing data set for neural network. In other words, instead of solving games using a game tree we are interested to discover whether a trainable AI can replace the game tree. We therefore investigate if we could suitably train a neural network which can give us *exact* next stages after learning from training set provided by theorem prover. This is different from all the other generative neural network processes e.g. a generative adversarial network (GAN) in which an approximate result is acceptable such as a generated image that may be similar to an image in the dataset, in our case approximate result is not acceptable since it can result in deformed stages which lead to dead ends. The other crucial difference is in the way training is performed. In GANs, two networks compete against each other, where one generates an example that may or may not be similar to the examples in the training set's distribution, something which is decided by a discriminative network. Our discriminative network is replaced by the puzzle-solving theorem prover which is guaranteed to deterministically generate the next stage. We conclude that with proper sampling we can get an exact next stage of board games with high accuracy.

In our approach, we train a Recurrent Neural Network (RNN) to classify Sokoban board states as valid or invalid, and use one valid state to propose the next valid state, such that each state brings us closer to the solution. The problem of declaring a particular state of the Sokoban board may be easy enough if all valid and invalid states could be enumerated, but to decide a move onward from a particular board state enumerating all following states may not be a combinatorially appealing solution. There are several motivations for the work being presented:

1) the trained network could be used for training new players as well as provide in game help. The network can be used to identify any state as either valid or invalid. This in turn may be used to identify whenever a player reaches a blocking state (from which there is no solution) and guide the player to back-track some steps. We can also use the model to design new levels for the game. If our proposed solution works well for Sokoban, it may be extended for more complicated puzzles and problems.
2) methods proposed in the literature takes exponential time to reach a solution in worst case. For example, the theorem prover we used to generate the solution takes around 5 minutes to solve many 8×8 puzzle. Larger puzzles will take increasingly more time. If we could train neural networks to solve the puzzles, solutions could be reached a lot quicker. If the learning approach works well, the networks may be trained to learn puzzles of any size just by watching human players play. All other methods for solving board games with single player and step by step solution require extensive domain specific knowledge to implement.

Concretely, our objectives are as follows:

1) to create some invalid Sokoban mazes. Invalid maze means any maze which is unsolvable. We needed to do this as although valid mazes are available on the internet [1], invalid mazes are not.
2) to design and train neural network architecture that could detect whether any given maze was valid or invalid
3) to generate step by step solutions for valid mazes using the theorem prover[2]

4) to use the solutions (from objective 3) to train a neural network so it could produce solution steps for similar mazes.

To the best of our knowledge, ours is the first work that:

1) applies a deep learning method to general puzzle-solving, and
2) uses a hybrid generator-generator training where the generator network is trained against a game-tree generator.

II. Literature Review

Sokoban is a popular puzzle game. It was developed in the 1980s in Japan. The player is given a maze. The maze consists of walls, boxes, floor, holes and one player. The player has to push the boxes onto the holes. Any maze is valid if it satisfies certain conditions. For example, if there are an equal number of holes and boxes and some way for the player to push the boxes so they cover all the holes, the maze is valid. A number of approaches have been used to find optimal solutions to Sokoban puzzles. Some solutions have used graph algorithms such as breadth first and depth first search algorithms.

[3] has proposed solutions using Best-FS (Best-First Search),Iterative-Deepening A* search, and Genetic algorithm. Sokoban solutions can be viewed as expansion of trees of possible actions based on a certain state of the puzzle. Both Breadth first and Depth first graph algorithms will blindly traverse the trees hoping to find the solution. The Best-FS algorithm uses heuristics to decide which tree branch to take based on which path will move a box closer to a hole.

In [4], a hierarchical decomposition approach has been proposed where the problem is divided into a sequence of higher actions and elementary actions. Secondly, a database is maintained which keeps track of the mistake made by the algorithm giving it the ability to learn.

[5] propose a method for finding optimal solutions using Instance Dependent Pattern Databases. Using this method, the puzzle is decomposed into a goal zone, entrance and maze zone. A distance database is created from each box to a space called an entrance which is any square from which the goal may be reached.

In [6], an algorithm to generate Sokoban levels automatically has been described where they create an empty room, place goals in the room and find states farthest from the goal state, i.e. go from the end state back to a start state.

In all the papers discussed, knowledge about Sokoban and its rules is necessary to implement the solver or maze generator. We propose to circumvent the need to know the rules of a game for actual game-playing by the use of a generative neural network.

III. Methodology

We constructed a dataset of 700 valid Sokoban puzzles from [1]. We divided the dataset into halves so that one half $(i.e.350)$ represent the valid Sokoban puzzles and the other 350 are modified in a way that it leads to an invalid maze stage. Each puzzle has a dimension of 8×8. For puzzles which are smaller in any dimension, we pad with walls.

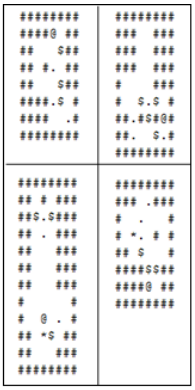

Fig. 1. Some valid maze configurations[8]

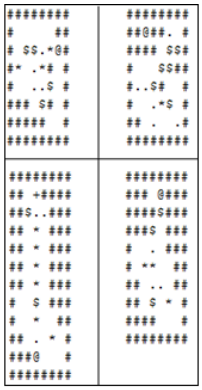

Fig. 2. Some invalid maze configurations

Each level of Sokoban contains 7 elements shown in Fig. 3. This results in a state space of 7^{64} stages. We then assign each maze element an integer value as shown in Fig. 4. We then convert all the elements to $1D$ as shown in Fig. 5.

Level element	Character
Wall	#
Player	@
Player on goal square	+
Box	$
Box on goal square	*
Goal square	.
Floor	(Space)

Fig. 3. Maze elements

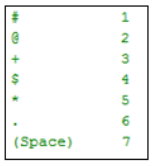

Fig. 4. Convert to integer values

There need to be a certain number of holes and a certain number of boxes in the maze, and the elements of the maze, i.e. the Walls, Boxes, Holes, and the Player need to be in a particular pattern for the maze to be valid. Some of the valid and invalid configurations of maze are shown in Fig. 1 and Fig. 2 respectively. Furthermore, since the player can only move in four directions (up, down, left, right), each state is the result

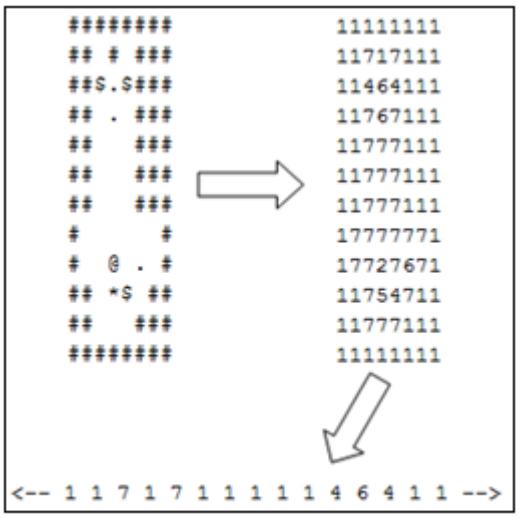

Fig. 5. Conversion of data to 1-D

TABLE I. MODEL LAYERS EXPERIMENT 1

Layer no.	Layer type	No. of Neurons
1	LSTM	128
2	Fully connected feed forward	2

of simple changes to the previous state, and is dependent on the layout of the maze elements shown in Fig. 3.

Our experiments are done on Keras with a Tensorflow backend. We used a special type of RNN called Long-Short Term Memory (LSTM)[7]. The LSTM provides a mapping betweennsequences of length 1. The input to the network at a given time stamp is a board state, and the predicted output is the next board state. For adversarial training, we limit the use case to Sokoban and the theorem proven in [2] which outputs a unique board state given the previous one.

IV. EXPERIMENTS

A. Experiment 1 - Prediction of Valid / Invalid state

There are many constraints which need to be fulfilled for a Sokoban puzzle to be valid. E.g. The number of boxes should be equal to number of holes. In other solvers these constraints are explicitly checked using if else statements. In deep learning approach, the network learns all of these just by viewing examples, i.e. it frees programmer from writing explicit constraints as if/else checks. Before coming to final experiment for prediction of valid / invalid state, we did a series of experiments to verify that our network can actually learn all the constraints for finding valid / invalid state. Input and Output of this experiment are shown in Fig. 6. Following are brief descriptions of our prior experiments.

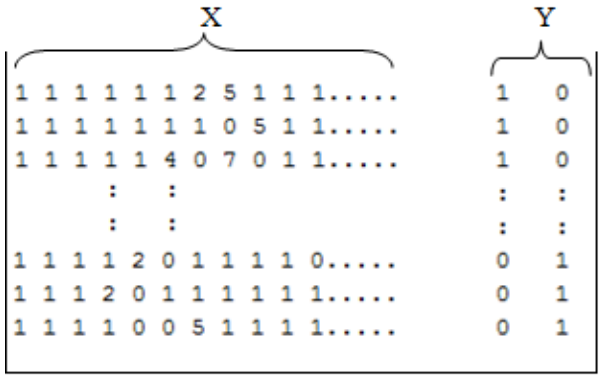

Fig. 6. Input and Output to the model

In our first experiment we made a Sokoban board with only three symbols, Box = '$', Goal square '.', Floor '(space)', and 16 squares. Each square can contain any of the three symbols. We marked a board as invalid if it contained unequal number of '$' and '.'. Our state space contained $3^{16} = 43046721$ examples. Out of these examples we randomly selected 2000 examples for our training data. After training our network up to 99.99% accuracy we tested our model on 50 unseen examples. In almost all experiments, (49 out of 50) unseen examples were correctly classified.

We subsequently experimented on an increasing number of squares and training examples:

TABLE II. RESULTS: B=BOX, G=GOAL, SQ=SQUARE, SP=SPACE, F=FLOOR, W=WALL

No.	Size	Symbols	Exp.	State space	Acc.	Cor. classified
2	64	B, G, Sq, Sp	8000	$3exp(16)$	99.99	$49/50$
3	128	B, G, Sq, Sp	8000	$3exp(128)$	99.99	$46-48/50$
4	16	B, G, Sq, F, W	2000	$4exp(16)$	99.99	$49/50$
5	64	B, G, Sq, F, W	4000	$4exp(64)$	99.99	$49/50$
6	128	B, G, Sq, F, W	8000	$4exp(128)$	99	$46-48/50$

TABLE III. MODEL LAYERS EXPERIMENT 2

Layer no.	Layer type	No. of Neurons
1	LSTM	128
2	Repeat Vector	
3	LSTM	128
4	LSTM	8

We repeated above prior experiments for 3, 4 and 5 relations and results are same. This showed that our network can learn up to 5 relations of length 128. More results can be produced by performing further experiments.

In our final experiment we have 7 symbols Wall '#', Player '', Player on goal square '+', Box '$', Box on goal square '*', Goal square '.', Floor '(Space)'. We downloaded 696.txt from [1]. This file contains 696 valid sokoban puzzles. We randomly selected nearly 350 sokoban puzzles out of the 696 puzzles. To make invalid puzzles we made little changes in each of 350 valid puzzles. Finally, we gave these 700 examples to our network along with their respective labels. Around 27-29 out of 30 unseen examples are correctly classified after training our network up to 99 percent.

B. Experiment 2 - Next-state predictor

In this experiment we downloaded 54 puzzles of 8×8 from $696.txt$ [1]. We gave every puzzle to the theorem prover[2]. The input and output of theorem prover is shown in Figs. 7 and 8 respectively. For valid mazes, the theorem prover generated step by step solutions. Fig. 8 demonstrates how the theorem prover generated step by step solutions.

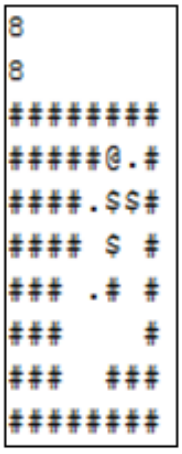

Fig. 7. Theorem Prover input

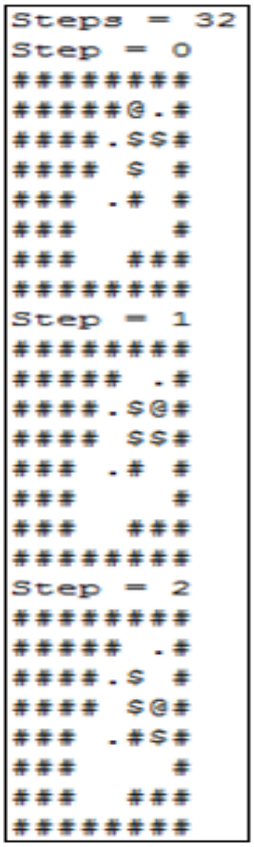

Fig. 8. Theorem Prover output

Our basic idea is to get board configuration in step 2 as the output of board configuration 1. For example, the output in step 2 becomes input for step 1. If our puzzle is solved in 32 steps than we will get 31 training rows for our network {1-2, 2-3, 3-4, ... 31-32}. Both X and Y contain 64 characters as shown in Fig. 6. The model of our network is shown in Table III. Each puzzle out of 54 puzzles gets solved in 16 to 64 steps. As a result we get nearly 3000 rows for our initial experiment. After achieving 99 percent accuracy on training data, we tried to solve whole puzzles using our network. We randomly chose a single puzzle from the 54 puzzles that we used for training set. This was a puzzle that was solved by theorem prover in 64 steps. 57 out of these 64 steps are correctly guessed by our network. For the other7 steps it gives an error of maximum 2 elements out of 64 elements in the maze. Next we fed an unseen puzzle to the network. The theorem prover had solved this puzzle in 64 steps as well. Although our trained network was unable to guess any step up to complete 64 places correctly, it guessed a maximum of 4 elements incorrectly out of the total 64 elements. Most of the time, the error was only at 2 places out of 64 places. We think this shows that 54 puzzles were not enough for solving Sokoban 8x8 puzzles. We will discuss how to increase problem instances for our problem in sections 5 and 6.

V. Results and Discussion

Experiment 1 shows that we can find valid/invalid configuration for many board game with very few examples. Many Sokoban solvers which use A*s search algorithm get stuck when given an invalid sokoban puzzle with just 1 unmoveable box, they keep on searching the whole state space for solution. Our proposed solution is very simple. We train a network which just guesses whether a given problem is valid or invalid. If it is invalid, we won't waste time in trying to find a solution.

We conduct our Experiment 2 to provide an efficient solver for board games like Sokoban, Colour Bridge and other similar games. The basic motivation for Experiment 2 is that once trained properly, neural network can guess solution to any Sokoban puzzle within seconds. In comparison, when we give random examples of 8x8 sokoban puzzle to any current solver which uses A* algorithm or theorem proving technique, it normally takes atleast 60 seconds to solve a puzzle, and for many 8x8 puzzles current solvers get stuck altogether. We have not experimented on 9x9 or bigger instances of the puzzle because most current solvers get stuck on these problems. As a consequence, we are unable to generate the required amount of train data.

There is a direct analogy between all the sequence to sequence learning tasks. For example puzzle solving by our method has direct analogy with question-answering task from supervised learning. Both are sequence to sequence learning, both have vocabulary. The difference is that the vocabulary is very small in case of a Sokoban puzzle, i.e. just 7 words, while in question-answering context, it is atleast 32 (as in the case of babi_rnn (facebook data)) for which an RNN gives 99% accuracy for 1 word answers. Here we are trying to extract a 49 or 64 word answer, so the error is expected. Bigger dataset which expect answers of 10 or more length are not extracted exactly i.e. giving 81% or less accuracy even by state of the art results [squad references]. what we are trying to discover is that if vocabulary size is small, is it possible to extract exact answers of size 49 or greater. We conclude that for the sample size we used in our experiments, we are able to extract answers with an error of 2 characters most of the time. For question- answering task, an answer with an error of only two characters can be a most ideal answer, but for puzzle solving we will need exact answers. We keep proper method of sampling from puzzle state space as future work since solution to the puzzle solving problem lies in proper sampling according to our analysis. We keep implementation and result enhancement using proper sampling method as future work.

VI. Conclusion and Future Work

In this work we propose a variant of adversarial training for the application of puzzle solving, where one generative model is used to train another in a way similar to a human oracle providing labels at each time step to a recurrent neural netwrok. We obtain 99% accuracy on validation set which implies network has learned whatever structure has been given to it in training and validation set. Now if test data is similar to training and testing set, the network will be able to extract correct answer. Since this is not happening, it implies that sample size is small. Several reasons mainly related to the size of the possible state space of the puzzle problem suggest that the problem lies in small size of training set. In other words the sample size is not large or varied enough to train the network properly.

There are many more reasons for comparing our work with other sequence to sequence learning tasks. There are non-trivial differences between Question-Answering (QA) and Puzzle Solving. For example, QA can be based on comprehension where answer spans searching through multiple interrelated sentences in a monotonic order, while in puzzles a change of just 1 position can result in new state. Variable-length answers (which make the QA problem difficult) and a fixed-length answer is also another key difference between QA and puzzle solving task. Due to these differences we can expect that in modeling sequence to sequence learning, the easiest task can be modeling of state to state for puzzle solving (fixed length answers, not dependent on many positions/sentences, with a very small vocabulary size). This argument motivates us to expect long exact answers for puzzle solving using sequence to sequence modeling.

REFERENCES

[1] http://www.sourcecode.se/sokoban/levels/

[2] http://bach.istc.kobe-u.ac.jp/copris/puzzles/sokoban/

[3] M. Dorst, M. Gerontini et al, "Solving the Sokoban Problem", 2011

[4] Jean-Noel Demaret, Francois Van Lishout et al, "Hierarchical Planning and Learningfor Automatic Solving of Sokoban Problems"

[5] Andre Grahl Pereira et al, "Finding Optimal Solutions to Sokoban using Instance Dependent Pattern Databases", Institute of Informations Federal Univerity of Rio Grande do Sul, Brazil, Proceedings of the Sixth International Symposium on Combinatorial Search, 2013

[6] J. Taylor, I. Parberry, "Procedural Generation of Sokoban Levels", Dept. of Computer Science and Engineering, University of North Texas

[7] Hochreiter, Sepp and Schmidhuber, Jurgen, "Long Short-Term Memory", Neural Computation, 9(8):1735-1780,1997