Reinforcement Learning – Sudoku Puzzle

Ravindra Rao
Data Scientist, Insights & Data, Capgemini
Ravi_rao26@hotmail.com

Abstract
This paper outlines the approach of using Reinforcement Learning to train a machine learning model to solve a traditional sudoku puzzle.

Keywords
Sudoku, Reinforcement Learning, DynaQ, Machine Learning, Q-Learning

1. Introduction

The write-up illustrates the application of Reinforcement Learning in solving Sudoku puzzle. The Sudoku puzzle, as we are aware is a 9x9 grid of numbers so arranged that each row and each column of the grid consists of digits 1-9 and the 9 blocks of 3x3 grid also consists of digits 1-9.

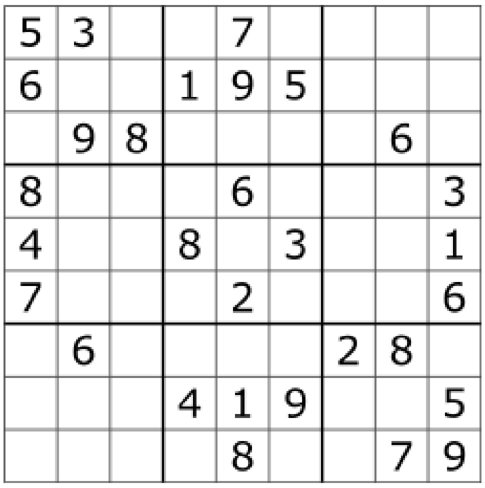
2. Sudoku Solve Approach

There are two approaches when it comes to employ a computer to solve a sudoku puzzle – Coding Approach, Machine Learning Approach. The coding approach follows a constrained programming method which uses a backtracking mechanism to solve the puzzle. For reference, I have written a code in python which processes a given sudoku puzzle and produces the solved sudoku puzzle. The code has been packaged into the application .exe file and uploaded over here [1].
The objective of writing a code to solve the sudoku puzzle was to gain a deeper understanding on how machine learning could be applied to solve a sudoku puzzle. There are deep learning solutions implemented where in a CNN (Convolutional Neural Network) is used to solve sudoku puzzle. Supervised learning [2] is used where in the CNN network is trained on millions of unsolved (input) and solved (output) sudoku images. This strategy does come across as a brilliant method and instances of 98%-99% accuracy has also been observed.
But if we look at a sudoku puzzle then there are some rules to be followed and solving a sudoku puzzle also involves application of a strategy to solve the sudoku in minimum number of steps possible. The supervised learning approach deprives the model of learning this strategy. Even though the objective is achieved through supervised learning but this approach doesn’t seem right for solving puzzles. To give an analogy over here, it’s like teaching a child addition of numbers without teaching the child to count (strategy).
Given the scope of application of strategy in solving the sudoku puzzle, Reinforcement Learning could be implemented which would enable the model to learn the strategy to solve the puzzle rather than memorizing it as seen in case of supervised learning. Here below, I have outlined an approach which could help us in implementing Reinforcement Learning solution/model for sudoku.
The Reinforcement Learning consist of an environment which consists of State, Actions and Rewards and a RL agent which is subjected to the environment. The agent takes an action at each time step which results in the next state and subsequent reward. The objective of the agent in any given state of an environment is to select the next action which maximises the reward it receives thus propelling it to next state until it reaches the terminal state. In the learning phase the agent would take random actions leading to random states. The learning algorithm does keep a tab on the rewards incurred as a result of a particular action in a particular state. In subsequent iterations the agent is able to map the states and

actions thus developing a strategy which will aid in achieving the terminal state in minimum number of steps.
In a traditional sudoku puzzle out of 81 numbers, 30 numbers are already provided. These numbers are ‘valid’ numbers. So here, we would be having 30 valid numbers, 51 invalid numbers. Now for defining the states of a sudoku puzzle, we can define states as 51 invalids, 50 invalids, … 0 invalids. 0 invalids state representing the solved status or the terminal state.

2.1 Reinforcement Learning Approach - State Action Rewards

Below, considers an example of a sudoku puzzle



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

Figure 1. Sample Sudoku Puzzle

The above sample problem (Figure 1) can be represented as

```
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
```

The default solved sudoku puzzle is

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

Now if we are to map the above default solved puzzle against the given input puzzle, the resultant sudoku puzzle would take the below shape. Refer this state of the puzzle to be ‘Initial State’ (Figure 2).

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

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

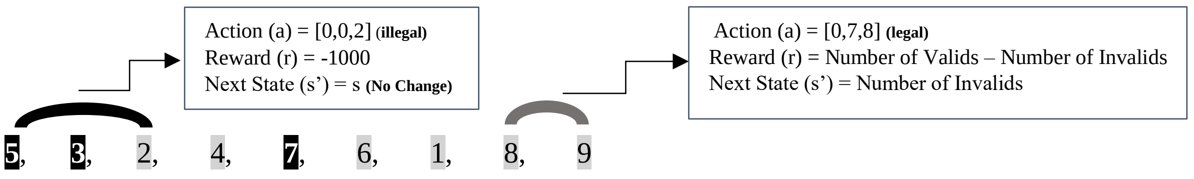
Figure 2. Initial State

The numbers in bold are provided numbers. These would be 'valid' by default. The numbers highlighted in black are invalid.

So initially there are 30 valid numbers (provided) and first iteration through the code would result in 21-invalids state. This would be the initial state of the environment: 21-invalids. The states of the environment would be defined as: 51-invalids, 50-invalids, 49-invalids…0-invalids.
The action set would be swapping of the numbers across the rows which would result in change of number of invalids i.e. change of state. For example, 1st row, swap 2 & 9. If we consider the action set, it would be set of 36 actions for each row. It is equal to 9C2 = 36 (number of combinations of 2 objects out of 9 objects). For 9 rows, 36*9= 324 possible actions for each state.
The goal of the agent is to reach the terminal state from any given initial state. In order to motivate the agent to execute an appropriate action in a given state, reward is associated to the actions which is more in magnitude if the action results in state closer to the terminal state.
For the sudoku puzzle, the provided numbers shouldn't be touched/swapped. The action set could be divided into two sets, legal action & illegal action. For example, if we consider the first row then the action which swaps 5 with any other number, for instance 5 with 3, 5 with 2, 5 with 4 is an illegal action as 5 is provided in the first row, first column whereas action which swaps 2 with 4, 2 with 6 are legal actions.
Each and every action would be associated with a reward which will be accumulated by the agent when that action is taken. The illegal actions would be carrying a reward of -1000, the negative value would deter the agent in considering this action. The legal actions would be carrying a reward which is equal to the difference between the valid state and invalid states.

2.2 State, Action & Rewards - s (current state), a (action), s' (next state), r (reward)

Environment Current State (s) = 21-invalids


Action (a) = [0,0,2] (illegal)
Reward (r) = -1000
Next State (s') = s (No Change)
Action (a) = [0,7,8] (legal)
Reward (r) = Number of Valids – Number of Invalids
Next State (s') = Number of Invalids
5, 3, 2, 4, 7, 6, 1, 8, 9

Row 1 Illustration (Initial State – Refer Figure 2)

Referring the above illustration, given the current state: 21-invalids, the next action would be swapping of any two numbers across the rows. For illustration purpose, first row is considered. The action is defined in the format [Row Index, Index of 1st Number, Index of 2nd Number].
For instance, action [0,0,2] represents swapping of first element and third element of the first row. The action set is the list which is defined as follows as dictionary object.
0: [0,0,1], 1: [0,0,2],2: [0,0,3] ….8: [0,0,8] ….35: [0,7,8] ….73: [1,7,8] ……323: [8,7,8]
Thus, at each state there is a possibility of executing any one of the 324 actions.

2.3. State Diagram

Below (Figure 3) is the state diagram representing the transition between the states depending on the actions taken in that state.
As mentioned above, out of the possible 324 actions, anyone can be executed. The transition between the states is non sequential. Illegal actions would not result in any state change but the agent would be rewarded with -1000.
The legal actions would fetch the reward which would be equal to the difference between instances of valid numbers and invalid numbers.

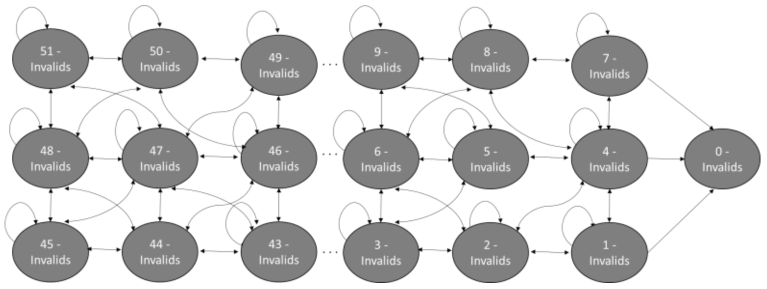


Figure 3. State Diagram

2.3 State-Action Matrix

At each and every state the agent could possibly execute any of the 324 actions. For each and every state, the learning algorithm would maintain a lookup table which could be represented as 18*18 matrix consisting of probability value of executing the action when in that state.

<u>State: 51-invalid</u>
[a0, a1, a2 …. a17]
[a18, a19 …… a35]
.
.
[a306, a307, ...a323]

<u>State: 50-invalid</u>
[a0, a1, a2 …. a17]
[a18, a19 …… a35]
.
.
[a306, a307, ...a323]

For 51 such states, the number of these values would be 324*324*… (51 times) i.e. 324^51. The Q-Learning Algorithm during its learning phase will be updating the lookup table, this operation is internal to the algorithm. The actions which fetches maximum reward or which results in greater rewards over a period of time will be updated with higher probability values during the learning phase.
The agent during the learning phase would learn a strategy to plan the sequence of actions which will result in reduction of invalid states, thus leading to the terminal state of 0 invalids being present which is the solved status.

2.5. Code Convergence

Now, considering the above sudoku puzzle example (refer section 2.1), a code [3] has been written to check for convergence (i.e. reaching the solved state) where in the next action is chosen on basis of

reward at each step. The code illustrates states, action rewards definition of the sample sudoku puzzle, demonstrating convergence i.e. arriving at the solved state incrementally by determining next action based on maximum rewards.
It’s been observed that the execution does get into an infinite loop and is not able to converge. But on resetting the environment to the initial state post execution of certain number of iterations, convergence has been observed. Here the number of iterations within an episode is set to 2000. In the 6th execution cycle, convergence was observed in the 939th step.
The code was run on other sudoku puzzles, the convergence was observed in 27th execution cycle, 1428th step in one of the puzzles whereas in case of other puzzle, the convergence was observed in the 1st execution cycle, 338th step.
Since the convergence is achieved through code, no matter how many times the code is executed on the sudoku puzzles, it would take the same time to converge as there is no learning incurred in the code.

3. Reinforcement Learning Algorithm

As explained in the above section, the convergence achieved through code does not incur any learning. However, a machine learning approach could be applied in solving the sudoku puzzle where in it would learn an optimal strategy to reach the terminal state (i.e. convergence) through number of iterations in learning phase.
Using the approach stated in this post, an environment simulating sudoku puzzle could be defined. The Action set, reward set and state set can be defined as illustrated above.
The Sudoku puzzle environment is a dynamically changing environment and hence it would be beneficial if planning is incorporated. Dyna Q algorithms are suited for factoring changes in environments during learning phase by incorporating planning step. Dyna Q+ algorithm could be employed to train the reinforcement agent in solving the sudoku puzzle.
Here, each and every sudoku problem is an environment in itself. A reinforcement learning trained model would be able to solve the sudoku puzzle optimally it was subjected to in the learning phase. One of the challenges that is foreseen over here is to come up with one model that is able to solve any sudoku puzzle. This would involve training the model on many sudoku problems. This could be seen as the next step where in mechanism could be devised to train the model on several sudoku puzzles so that it may come up with an optimal strategy that could be applied on any sudoku puzzle, this is subjected to further exploration beyond the scope of this paper.

4. Conclusion

The Sudoku puzzle solution through application of a code uses a backtracking approach. Alternatively, machine learning approaches are being sought to solve the sudoku puzzle. If the sudoku puzzle is defined in terms of state, actions and rewards then the solving of puzzle can be viewed as an optimization problem where in the number of invalid states have to be reduced through sequence of actions in minimal number of steps possible. This does involve application of strategy which can be learnt through Reinforcement Learning.

5. Reference

1. https://github.com/Ravi-Rao26/sudoku-solver-execute - Project Repository. Sudoku solver
2. https://www.theregister.co.uk/2017/12/01/ai_machine_leaning_sudoku/ - Supervised learning.
3. https://github.com/Ravi-Rao26/sudoku-solver-machine-learning/tree/master/Code-Convergence – Project Repository. Demonstrating convergence through code for sample puzzle