

Final Year Project Report

Full Unit - Final Report

Playing Games and Solving Puzzles Using Artificial Intelligence

Terique Carnegie

A report submitted in part fulfilment of the degree of

BSc (Hons) in Computer Science

Supervisor: Michail Fasoulakis



Department of Computer Science
Royal Holloway, University of London

April 9, 2025

Declaration

This report has been prepared on the basis of my own work. Where other published and unpublished source materials have been used, these have been acknowledged.

Word Count:

Student Name: Terique Carnegie

Date of Submission: 11/04/2025

Signature: Terique Carnegie

Table of Contents

Abstract	3
1 Introduction	4
1.1 Artificial Intelligence Solving a Sudoku	4
1.2 Purpose of my Application	4
1.3 Project Milestones	5
1.4 Professional Issues Faced with the Project	5
2 Application Development	6
2.1 Development of Terminal Interface	6
2.2 Development of Backtracking Algorithm	7
2.3 Development of Graphical User Interface	9
2.4 Development of the convolutional Neural Network	11
3 Final Product	12
3.1 Application Architecture	12
3.2 Running the Application	12
4 Project Evaluation	15
4.1 How well the Application Achieves its Aims	15
4.2 Potential Improvements for the Application	15
5 Conclusion	16
Bibliography	17
Appendix	18

Abstract

The general purpose of artificial intelligence is to give computers the ability to replicate human intelligence through the use of problem-solving algorithms. AI has the potential to outperform human intelligence in solving many different and difficult problems due to its ability to “think” by running problem solving computations at much greater speeds and accuracy than a human and it does this through leveraging the superior processing capabilities of a computer [1].

Due to the nature of problem-solving algorithms, Artificial Intelligence is able to provide assistance and insight into many different fields of expertise making it popular and very much in demand as an asset to humanity. In order to increase the value of AI, it is necessary to deeply research ways to improve problem solving algorithms and to discover which of the algorithms works best for a specific problem, and it is also important to develop methods for humans and AI to safely interact. One way this can be achieved is through games and puzzles. Using games and puzzles as a means to research and further develop AI can be very beneficial as it allows for problem solving algorithms to be tested against fun solvable challenges which can provide insight on how well the algorithms are able to solve problems, using games can also provide a way for humans to interact with artificial intelligence through either testing the speed and efficiency of both the player and the algorithms or through the use of 2 player games like chess and checkers. This type of approach to using artificial intelligence has been seen in the past from the first working checkers program to appear in 1952 [3], and chess playing programs being developed shortly thereafter [2] which had a role in furthering research and development of modern algorithms and uses of Artificial Intelligence.

Chapter 1: Introduction

1.1 Artificial Intelligence Solving a Sudoku

Artificial Intelligence has been applied to many different fields of expertise and has had many advancements in their problem solving algorithms. Some of the more modern and complex problem-solving AI algorithms that can be engineered to play and solve challenges in games are backtracking algorithms, constraint satisfaction algorithms, search algorithms, minimax algorithms, Monte Carlo Tree Search, reinforcement learning and neural networks. All of these algorithms allow for AI to be able to play games and potentially interact with human players, however for the sudoku solver, the project will focus on and utilise only a few of these AI algorithms. The selected algorithms for solving this problem will be back tracking and neural network algorithms. The other problem-solving algorithms are capable of providing a solution but my reasonings for my selection will be explained in the following paragraphs.

There are several different types of backtracking algorithms, such as consistency pruning, looking ahead, efficient backtracking and variable and value heuristics, but the basic principle behind backtracking algorithms is to explore all possible solutions through attempting different paths recursively and regressing back to try a different path when the current is no longer deemed viable. A sudoku can be solved by assigning numbers to an empty square but validating the number before to check if it is safe to assign, however this kind of approach has a worst-case time complexity of $O(9^{n \cdot n})$ [4].

Neural networks like convolutional neural networks and recurrent neural networks can be used to solve problems through pattern recognition. It does this by imitating how the human brain works through layers of interconnected nodes that takes in data, performs computations on the data and provides a prediction based on the data analysed. This can be applied to solving a sudoku as if trained on the rules and sequences of numbers of sudoku training data, the algorithm could be able to predict the numbers that will fit sudoku sequence with up to 99% accuracy [5].

1.2 Purpose of my Application

Through this project I plan further the understanding of how Artificial Intelligence can be applied to solve games and puzzles with logical thinking and pattern recognition by developing a platform using the Python programming language for developing my AI algorithms, with the pygame library for the development of the Graphical User interface and the Visual Studio code IDE to effectively manage my work files, to aid in my projects goal of demonstrate the capability of Artificial intelligence and how it can quickly assess and solve puzzles and problems in a gaming environment, by creating multiple problem-solving algorithms which will tackle and solve sudoku puzzles of increasing sizes and complexity, and evaluating which of the following algorithms between both backtracking and neural networks is better for solving a sudoku puzzle efficiently and potentially faster than humans in most cases. I believe that this project will help me to display my skills as a software engineer and can help me to demonstrate a greater understanding in the application of Artificial intelligence and the type of fields and situations that it can be applied to in the technology sector, which will make me very versatile in this industry.

1.3 Project Milestones

There were a couple of milestones achieved in the development of the Artificial Intelligence solver. The following milestones played a significant role in putting together the application so that it can run and achieve the goals that the project was set to achieve within the first term of development, these milestones were:

- **The development of the Console Interface:** this milestone had significance in the project as this milestone allowed for displaying the puzzle on the screen both solved and unsolved, this was a needed milestone as if the grid could not be displayed to the screen, it would be difficult to develop any of the AI algorithms as determining if the solving algorithms are outputting the correct results as reading from the array is not very efficient or user friendly due to the lack of a grid layout and sub-grid division.
- **The development of the Backtracking algorithm:** the first solving algorithm is key and very a very important milestone for this project as it displays that AI can be adapted to playing games and solving puzzles, it shows that machines are capable of making an attempt at solving the problem, and if the potential solution is wrong, it can go back and make a different attempt at a viable solution. As well as demonstrating how it is able to do so in a much shorter amount of time when compared to humans.
- **The development of the Graphical User Interface:** having a graphical user interface to display the steps and speed of the algorithm to a user is beneficial to proving that algorithm can learn from its mistakes, mimicking human intelligence, as it shows the algorithm choosing the incorrect solution and when realising its mistake, it displays the algorithm changing its solution to a valid one. This helps to demonstrate that AI can be used to solve problems quickly through the Sudoku grid solving animations that are run in the pygame window. It also provides an interface for the users to be able to interact with the application and switch between different solution algorithms, and sudoku puzzles.
- **The development of the Convolutional Neural Network:** Another integral milestone in the development of this project is the implementation of the secondary artificial intelligence algorithm, which is a CNN (a convolutional neural network), this milestone is important as it shows that a machine is capable of learning distinct patterns and features of a problem and its solution, similarly to a human, and use this knowledge to accurately predict a viable solution to a given problem. With the developed model, my application will be able to demonstrate how artificial intelligence can be trained to attempt to solve a mathematical problem and give a prediction with a suitable degree of accuracy.

1.4 Professional Issues Faced with the Project

Chapter 2: Application Development

2.1 Development of Terminal Interface

The main focus for the start of the sudoku solver needs to be the interface for displaying the sudoku. This is a necessary starting point as in order to demonstrate Artificial Intelligence solving the sudoku, the product needs to be able to display both the unsolved and solved solution in a way that is easily digested by the target audience. The borders, regions, numbers and void values need to be distinct and observable to those trying to understand what the AI algorithms is trying to achieve, and if the algorithms are doing so correctly. That makes this element of the product integral to the development of my project.

The best approach to initially work on this feature would be to work on a console based interface. This would be effective as the first iteration of the sudoku interface as it only needs to display an easily readable representation of the sudoku puzzle. This was achievable in the console as the borders can be represented through the '-' character for row division and '|' character for column division, the numbers as just numbers and the void values as an '.' character. this gave a simple and easy to read sudoku display on the console.

Division of the grid into sub-grids in the terminal would work through calculating the size of the sub-grids which can be based upon the size of the inputted grid as in most cases the length of the sub-grid is the square root of the length of the grid, for example, a 9x9 grid would have 3x3 sub-grids or a 16x16 would have a 4x4 sub-grids. However, in some cases a sudoku can have sub-grids with unequal length rows and columns, such as a 6x6 grid which has 2x3 sub-grids or a 12x12 which has 3x4 sub-grids. For the solver to be able to adapt so that it can print and solve sudokus of different sizes the application would need to be able to work out the correct row and column length based of the grid size.

A way to approach the interface algorithm is to calculate the square root of the grid length, and storing this value as a row and column divisor, however in the case where the square root is not a whole number, the algorithm rounds the value down for the row divisor and rounds it down for the column divisor but adds 1, which in most cases is enough to find the correct sub-grid lengths, as in the case of a 6x6 grid, the algorithm would calculate the root to 2.449... and round it down to 2 for the row divisor and set the value to 3 for the column divisor, creating the 2x3 sub-grids necessary for displaying the puzzle.

The divisors would be used by the algorithm to print a series of '-' characters for every row which is fully divisible by the row divisor, which separates the rows in the sub-grids. And for the columns, the algorithm would print the '|' character in the column index which is fully divisible by the column divisor. With this algorithm, the application is capable of taking in sudokus in the form of a 2 dimensional array and displaying a user friendly way of observing both an unsolved and solved sudoku problem.

A look at the final sudoku printing algorithm:

```
def print_grid(self, grid):
    if(grid == None):
        print("No found solution!")
        return
    divisor = grid.__len__() ** 0.5
    if(divisor%1!=0):
        row_divisor = round(divisor)
```

```

        column_divisor = round(divisor)+1
    else: row_divisor, column_divisor = divisor, divisor
    # logic for dividing sudoku grid into regions depending on sudoku length
    for i in range(grid.__len__()):
        if i % row_divisor == 0:
            print(" - " * grid.__len__())
            # uses dividing value to place row borders
            row=""
            for j in range(grid[i].__len__()):
                if(j % column_divisor ==0):
                    row+= " |"
                    # uses dividing value to place column borders
                    row+=" "+str(grid[i][j]) if grid[i][j]!=0 else " *"
                    # adds number or * to be printed for final sudoku output
            print(row+" |")
    print(" - " * grid.__len__())
    # printing the numbers of grid list or an * where there is a 0
    # and the borders to display the sudoku in the console to the user

```

2.2 Development of Backtracking Algorithm

2.2.1 What is Backtracking?

Backtracking is a main method of creating an AI algorithm to solve a constraint satisfaction problem. Constraint satisfaction problems are problems in which there are a finite set of variables that contain a finite domain of potential values which can be assigned to form a solution. These values will need to satisfy all the constraints of the problem in order for it to be a correct answer. Examples of CSPs are scheduling, puzzle solving such as sudokus and even resource allocation. Backtracking is an example of a complete algorithm, in which it can guarantee the if a solution does or does not exist for the constraint satisfactory problem [6].

Implementing a backtracking algorithm to solve a CSP is done through recursion and may even use a decision tree in order to represent the different potential values for the variable set of the CSP [7] The algorithm would generally work through choosing a value for a variable in the set, validating its choice, and if at any point a valid value for a variable can not be found, it will backtrack to a point where alternative decisions are available and have not all be exhausted and a different path to a potential solution is taken. This is done repeatedly until a solution is found, or there are no longer any unexplored values (all decisions have been exhausted completely) meaning that there is no solution to the constraint satisfaction problem [8].

2.2.2 Implementation of Backtracking Algorithm

A backtracking algorithm works similar to how a human would use backtracking as once a human discovers a mistake, an approach to fixing it would be to go back on their working out and find the cause of fault which gives them the undesirable output. The algorithms would work by making its own attempt to solve a problem. so in the case of a sudoku it would attempt to fill in blank squares with a potential valid value, and if a problem occurs in its

solution later on, such as the next blank square not being able to find a single valid value, then the algorithm would need to progress back to the cause of the problem, which would be the incorrect value in the previous square, or the ones before until it finds a different valid value to progress. The backtracking algorithm will then continue its attempt with a different solution for the sudoku, and this is done repeatedly till a viable solution is formed.

When developing the solving algorithm, first came the crucial element of finding a blank square in the grid, this would be represented by a 0 in the sudoku grid's 2 dimensional array. the coordinates on the grid would be returned to the main algorithm when found, then using these, the main loop will attempt to fill the square with numbers ranging from 1 to the max value, which is also the length of the grid (usually 9), until a viable value for the square is found. In order to check the validity of a value, the algorithm would exhaustively search through the row, column and sub-grid (checking every value in that area) for the value that is equal to the value about to be inputted into the blank square if the value is not found in any of the specified areas then it is considered valid, if the same number is found, then the current value is invalid and the algorithm attempts to check the next value for validity. If a viable value cannot be found, the algorithm clears the current square and backtracks to the previous square, and checks the next values for viability, if there are no viable values, the algorithm will clear the current square and backtrack once again to the square before and once again try different values until a different viable value is found and the algorithm searches for the next blank value and the algorithm repeats itself, until there is no more blank squares, or no more values to try on the grid that gives a solution to the problem.

A look at the main Backtracking algorithm:

```
def backtracking(grid, sudoku):
    position = empty_space(grid)
    # locates an empty square on the grid and stores its location
    if position == None:
        return grid
    # if there are no empty squares, the solution has been
    # found and the grid is returned
    for i in range(1, grid.__len__() + 1):
        sudoku.update(i, position[0], position[1])
        # updates the GUI to display the current attempt for a viable value
        if safe_value(grid, position, i):
            grid[position[0]][position[1]] = i
            # test a viable number in the open position
            backtracking(grid, sudoku)
            # checks the next space after the previous viable solution
            if(empty_space(grid) == None):
                return grid
            # if there are no empty positions, this solution is correct
        grid[position[0]][position[1]] = 0
        # if the current value is not viable, it is set to 0, but the i remains
        # as the none viable solution to be incremented
        sudoku.update(0, position[0], position[1])
        # updates the GUI to clear the current square
    return None
# if there is no number that goes into the empty position,
# an empty grid is returned as there is no solution
```

2.3 Development of Graphical User Interface

For the development of the graphical user interface to display the sudoku and the solving algorithm, pygame was a powerful python library to leverage in the programming of the GUI as it provides an adaptable environment in which the sudoku grid can be drawn on the window and updated with little difficulty while still being able to display a way of visualising the backtracking algorithm. When creating the grid on the window it was best to give it the length 340 which was big enough for the user to see the sudoku and the values in each square clearly. When dividing the grid into sub-grids, the same root and rounded divisor logic used in the console print function, however in the case of the GUI the grid was divided visually using thicker lines to create the division between the whole and sub grids. Then after the grid has been drawn in the window, the algorithm starts adding values to square by placing a new square over the empty square with the intended value.

It is also necessary to be able to clear the squares within the grid, this is so that the sudoku solver will be able to show its iterations of the solution. In the updating squares method, the squares cleared through overwriting the square by filling the surface with a white square then updating the value to a new square to cover the cleared square. this function would need to be called every time the value changed in the algorithm in order to give the visual backtracking effect.

By introducing a GUI to display the solving algorithm, the solver is able to show the step by step solution being formed through mistake adaptation approach my backtracking algorithm takes as well as showing how quickly the computer can calculate this as the GUI updates the values very rapidly, demonstrating how much faster a computer is compared to a human, despite the GUI technically wasting computer processing speed, so it is faster to print the output directly then updating a GUI, but then the algorithm will not be demonstrated in an easily comprehensible manner when being displayed in that manner.

The final iteration of the Graphical User Interface of the application now also incorporates additional sections which are placed around the main grid, which has been adjusted to better center the application as it is the main and primary focus of the project, which displays a guide on the inputs the application uses to allow for the user to interact with the program, giving access to features such as switching between the different solution algorithms and the unseen sudoku problems to watch the application solve with. However, due to the slow nature of the development of the Convolutional Neural Network, it has only been optimised to solve 9x9 sudoku grids, which means the adaptive nature of the grid's rendering algorithm is not fully utilised in the final production application, although this will not compromise the applications ability to provide an insight into how well and efficiently an algorithm can solve the mathematical sudoku problem.

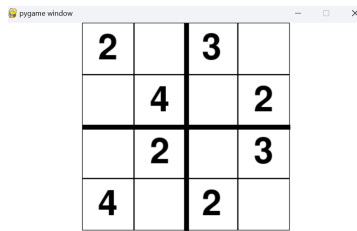
The main loop for the displaying the grid in the GUI

```
for row in range(grid.__len__()):
    self.board.append([])
    # adds a array inside another array, representing rows on the board
    if(row%row_divisor==0 and row != 0):
        pygame.draw.line(self.screen, (0, 0, 0),
            (col_offset, row*square_size),
            (col_offset+(grid.__len__()*square_size), row*square_size), 8)
        # where the board is divided based on divisor,
        # thicker lines are used to show the divisions that
        # make up the sub-grids
    for col in range(grid.__len__()):
```

```

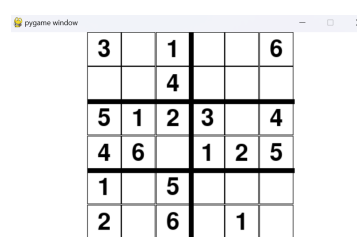
square = pygame.Rect(col_offset+(col*square_size),
row*square_size, square_size, square_size)
value = self.font.render("" if grid[row][col] == 0
else str(grid[row][col]), True, (0, 0, 0))
val_rect = value.get_rect()
val_rect.center = square.center
self.screen.blit(value, val_rect)
self.board[row].append(square)
pygame.draw.rect(self.screen, (0, 0, 0), square, 1)
# the squares are created within the boundaries of the grid,
# and smaller rectangles with the preset
# puzzle values are displayed within the squares where their
# coordinates are found from the input array
if(col%column_divisor==0 and col != 0):
    pygame.draw.line(self.screen, (0, 0, 0),
        (col_offset+(col*square_size), row*square_size),
        (col_offset+(col*square_size), grid.__len__()*square_size), 8)
# where the board is divided based on divisor, thicker lines are used
# to show the divisions that make up the sub-grids

```



2		3	
	4		2
	2		3
4		2	

Figure 2.1: 2x2 grid

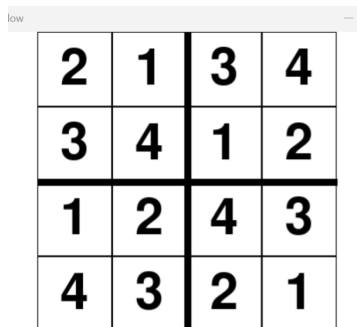


3		1			6
		4			
5	1	2	3		4
4	6		1	2	5
1		5			
2		6		1	

Figure 2.2: 6x6 grid

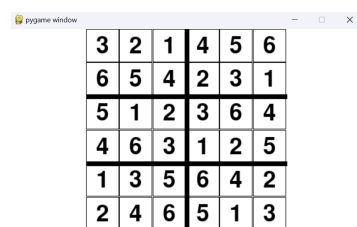
				2				
1		3	4					5
2				5		4		1
3	4				5		9	
8		7				3		4
	9		3				1	7
6		5		3				9
4					8	7		2
			1					

Figure 2.3: 9x9 grid



2	1	3	4
3	4	1	2
1	2	4	3
4	3	2	1

Figure 2.4: 2x2 grid



3	2	1	4	5	6
6	5	4	2	3	1
5	1	2	3	6	4
4	6	3	1	2	5
1	3	5	6	4	2
2	4	6	5	1	3

Figure 2.5: 6x6 grid

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

Figure 2.6: 9x9 grid

2.4 Development of the convolutional Neural Network

2.4.1 What is a Neural Network?

A neural network is based off of how the humans are capable of thinking through the use of a network of interconnected neurons. A neural network mimics the human brain's neuron connections through modelling an interconnected network of nodes which can receive information, process the data and output that data either to another layer in order to attempt a greater accuracy in its prediction through additional processing, or to the user as a final prediction output [9]. the model is capable of learning from its mistakes through the use of backpropagation and can also process information in parallel all of which can help the model to do things like pattern recognition, which can be applied to many different fields and problems which could potentially be the best means of finding a solution [10].

This Artificial Intelligence model is implemented by separating the interconnected network into layers. These layers are the input layer, the hidden layer (or layers) and the output layer [11]. The input layer is the primary layer that is used to take in all the inputs to be processed, hence its name. The hidden layers consist of the nodes that begin to process the inputted data, this is done through weights and biases associated with the connection between 2 nodes. These nodes are trained through passing in test data to help set and adjust the weights and biases of the connected nodes, by using a technique called backpropagation the algorithm can analyse predicted outputs and based on the output, make changes to weights and biases to update model and reduce its rate of error, allowing for growth and the ability to learn from its own mistakes [12]. The output layer is responsible for compiling the final prediction to be used as a solution to the problem it was provided to solve.

2.4.2 Convolutional Neural Networks

2.4.3 Implementation of the Convolutional Neural Network

The next focus of the project moving forwards will be on developing a Neural Network that will be trained on taking in sudoku solutions, this is so that it will learn what a solution to a sudoku should look like and it should recognise the patterns/rules of a sudoku so that when it is faced against an incomplete sudoku, it will be able to provide an accurate prediction of a solution, that will hopefully be the correct answer to the inputted problem.

With the use of a neural network, there is a possibility of overfitting, which is when the network produces great results when working within the training data, but when introduced to a new puzzle outside of the training set, the algorithm holds poor results. This is a risk to consider when developing the AI sudoku solver as poor results will impact the validity of the project results which will need to be able to show the effectiveness of artificial intelligence when implementing it in an environment to play games and solve puzzles. To mitigate this risk, it would be important need to dedicate time in the next term to researching solutions to overfitting, such as pruning, cross validation or data augmentation.

2.4.4 Model 1

Chapter 3: Final Product

3.1 Application Architecture

The project was divided in a way so that each file would have a specific purpose, a specific feature that would be needed to be used by the main file, in which the entire application is ran from. This organisation of the code base is beneficial as it allows for the program to be well documented, as the comments written in a file is relevant and specific to the feature being programmed. Also this architecture is good for the development of classes and features as they can be built upon and tested independently of one another which is necessary for keeping and maintaining good quality code for the AI sudoku solver, this also has the additional benefit of making feature updates and improvements easier to implement as making changes will only affect the current feature code and potential bugs will not be able to interfere with the progress being made on the other features of the main application.

An agile type of methodology style was implemented in the development process of the AI sudoku solver as the primary focus of the term time development cycle was putting together rapidly developed features that is capable of doing just enough to get the application running and connecting them together to get a working prototype which is able to print and solve the puzzle in a way which is easily interpreted by the applications intended users. This style of development will be carried out in the next term of development also as it would be more important to focus on developing the key features of the application so that the prototype is able to meet the set aims and objectives of the project and only once these are achieved, improvements and updates can be made to the code base to improve the application incrementally until the final product has been formed.

3.2 Running the Application

When running the main application there 2 different sets of outputs are provided for the user to examine, the console displayed unsolved and solved sudoku puzzle and the pygame window in which the puzzle is displayed initially unsolved, and then shows the step by step procedures the solving algorithm takes to present the user with an actual solution.

The console based output was beneficial to the first sprint development cycle as it got the job done in terms of displaying what the algorithm needed to solve and the result the AI algorithm produced, this method of output done best at showing just how quickly the AI algorithm could solve the problem as it would almost instantly display both the unsolved and solved solution to the sudoku. However, this method of output was inefficient of displaying the steps and the decisions that the algorithm makes as I would need to print multiple instances of semi-complete solutions, which is not a console friendly design as many iterations and decisions are made, so the mistakes and inaccuracies made before the solution is found is abstracted from the users.

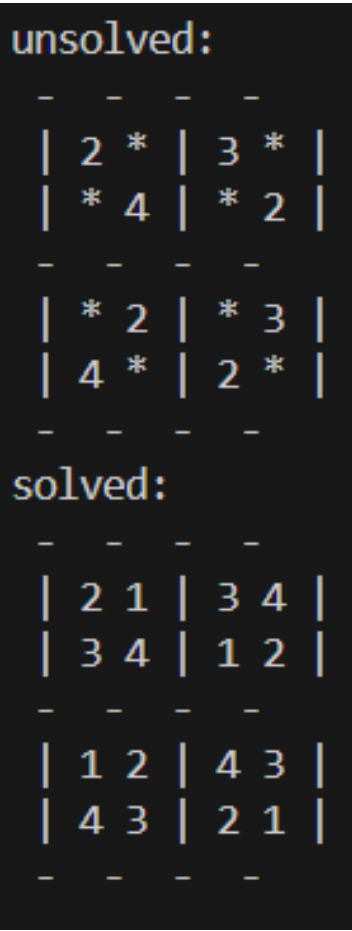


Figure 3.1: 2x2 grid

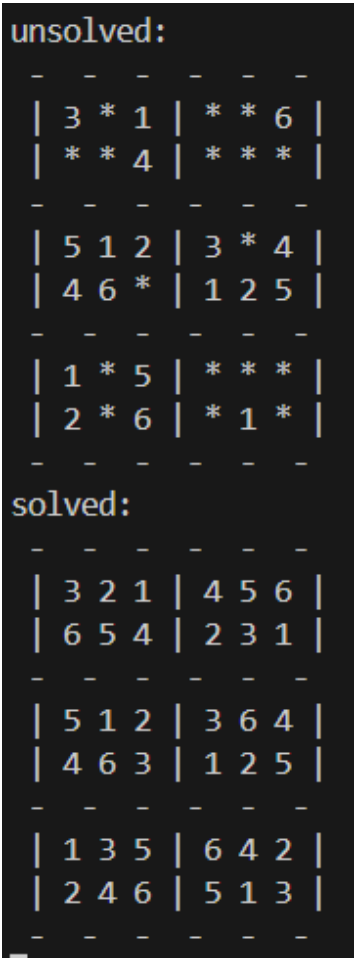


Figure 3.2: 6x6 grid

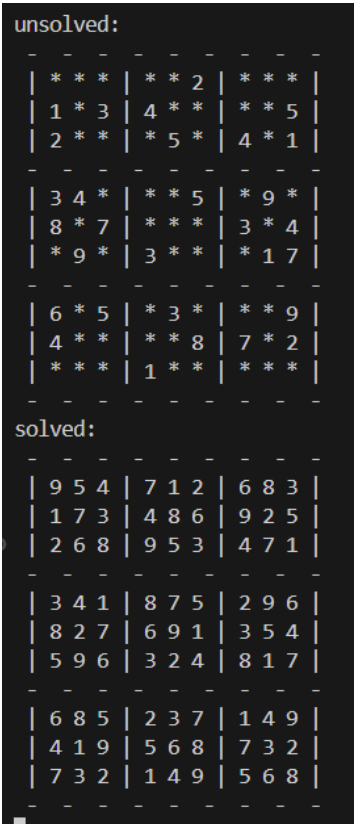


Figure 3.3: 9x9 grid

The GUI based output was a beneficial for its ability to display the steps and decisions that the algorithm takes to find the solution. It shows the mistakes the algorithm finds and shows the how it works backwards, similarly to how a human would potentially try and find the solution to the sudoku problem. However, this form of output, despite solving the solution fairly quickly, potentially much quicker than most humans could, due to the algorithm requiring extra steps of clearing the squares and updating its value on the window's grid means that this output is generally slower as more computational power is needed from the machine, although this is not that serious of an issue as the speed of the solving algorithm is still capable of proving and meeting the aims and objectives the application was set out to achieve.

After additional changes the layout of the final application was altered to better center the grid on screen as it is the primary focus of the application as this is where the processing power of Artificial Intelligence will be used on the problem and its solution displayed. in addition to this, 2 other sections have been included in the interface to firstly explain to the user how to use the application, which inputs the software takes to allow the user to switch between the backtracking algorithm and the Convolutional Neural Network prediction with the Enter and Space keys, and different preset sudoku problems which the algorithms have not seen before with the Number keys 1-4, as well as the Backspace key to clear the current solution and leaving behind the unsolved sudoku. The second section is where the time taken for the solution algorithm to give its final answer, and the number of mistakes the backtracking algorithm had made to reach the final solution or the number of mistakes the final prediction of the Convolutional Neural Network has made.

Chapter 4: Project Evaluation

4.1 How well the Application Achieves its Aims

The final product provided uses 2 different algorithms to solve the classical mathematical problem, a sudoku. The backtracking algorithm and the Convolutional Neural network both work within the application to demonstrate just how well machines can learn through using 2 different techniques and still provide a suitable solution to a range of sudoku problems of varying difficulty which is more computationally efficient in comparison to a human. The application shows this through displaying the grid and rendering the solution in real time, along with the time it took the algorithm to calculate its solution and the number of mistakes it has made to get to its conclusion, in the case of the backtracking algorithm, and the mistakes in the final prediction for the CNN. Despite the errors in development that needed to be overcome to develop the project to this state, the product does its job well in meeting its objective of developing AI algorithms that is capable of playing games and solving puzzles, and doing this task more effectively and efficiently than mankind, proving the superior processing capabilities of a computer.

4.2 Potential Improvements for the Application

In the end, the Project done well in meeting the set out expectations, however, there is always room for improvement and there are a range of features that could have made for a better application and would have provided a better user experience, which would have aided the project in superseding its objectives and better demonstrating the power of Artificial intelligence as a means to solve problems. One of the areas that needed improvement was the final convolutional model. Despite the model having a decently high accuracy and validation accuracy, around 97%, these scores were not yet perfect as there was still a degree of uncertainty in the predictions which lead to a few mistakes being generated in some of the 81 cells, which with additional time to develop could have been ironed out to find a truly optimal Convolutional Neural Network. In addition to the final model, the application could have benefitted from a feature which would have allowed users to input their own sudoku problems, rather than the preset selection to switch between, as this would have shown the effectiveness of the 2 algorithms with a greater variety of problems. Training additional optimal CNN models on variety of sudoku sizes would give the application the ability to show an ability to solve size variants of the classic sudoku problem, which would have allowed for the utilisation of preexisting features such as the adaptive grid rendering and the backtracking algorithms versatility.

Chapter 5: Conclusion

In conclusion, the development of the AI sudoku solver has gone in achieving its goals as significant progress has been made for each milestone such as the development of the console interface, the graphical user interface, the Backtracking solving algorithm and the CNN which helps the application to demonstrate how Artificial intelligence is superior to human intelligence due to its computational capacities and speed and how it can be applied and adapted to a gaming environment, in order to attempt to play games and solve puzzles efficiently through learning from its mistakes and trying to discover an accurate and viable solution for a problem, which in the case of this project, is a solution to different sudokus of varying difficulty.

Bibliography

- [1] AI for Social Good (n.d.). Artificial Intelligence vs. Human Intelligence: Exploring the Debate and Key Points. [online] Available at: <https://aiforsocialgood.ca/blog/artificial-intelligence-vs-human-intelligence-exploring-the-debate-and-key-points> [Accessed 8 Oct. 2024].
- [2] C. Strachey, Logical or non-mathematical programmes, in: Proc. Association for Computing Machinery Meeting, Toronto, ON, 1952, pp. 46–49.
- [3] J. Kister, P. Stein, S. Ulam, W. Walden, M. Wells, Experiments in chess, J. ACM 4 (1957) 174–177.
- [4] GeeksforGeeks (n.d.). Sudoku — Backtracking-7. [online] Available at: <https://www.geeksforgeeks.org/sudoku-backtracking-7/> [Accessed 9 Oct. 2024].
- [5] Charles Akin-David, Richard Mantey, n.d. Solving Sudoku with Neural Networks. [pdf] Available at: https://cs230.stanford.edu/files_winter_2018/projects/6939771.pdf [Accessed 10 Oct. 2024].
- [6] S. C. Brailsford, C. N. Potts, and B. M. Smith, *Constraint satisfaction problems: Algorithms and applications*, European Journal of Operational Research, vol. 119, no. 3, pp. 557–581, 1999. Available at: <https://www.sciencedirect.com/science/article/abs/pii/S0377221798003646>
- [7] P. van Beek, *Backtracking Search Algorithms*, Foundations of Artificial Intelligence, vol. 2, pp. 85–134, 2006. Available at: <https://www.sciencedirect.com/science/article/abs/pii/S1574652606800088>
- [8] Melanie, *Backtracking: What is it? How do I use it?*, DataScientest, 29 Mar 2024. Available at: <https://datascientest.com/en/backtracking-what-is-it-how-do-i-use-it>
- [9] A. D. Dongare, R. R. Kharde, and A. D. Kachare, *Introduction to Artificial Neural Network*, International Journal of Engineering and Innovative Technology (IJEIT), vol. 2, no. 1, pp. 189–194, 2012. Available at: <https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=04d0b6952a4f0c7203577afc9476c2fcab2cba06>
- [10] C. Woodford, *How neural networks work - A simple introduction*, Explain that Stuff, 12 May 2023. Available at: <https://www.explainthatstuff.com/introduction-to-neural-networks.html>
- [11] IBM, *What is a Neural Network?*, IBM, 2024. Available at: <https://www.ibm.com/topics/neural-networks>
- [12] J. Schmidhuber, *Deep Learning in Neural Networks: An Overview*, Neural Networks, vol. 61, pp. 85–117, 2015. Available at: <https://arxiv.org/abs/1404.7828>

Appendix

Sudoku Interface

Importance of Interface Implementation

The main focus for the start of my project needs to be the interface for displaying the sudoku. this is a necessary starting point as in order to demonstrate Artificial Intelligence solving the sudoku, my product needs to be able to display both the unsolved and solved solution in a way that is easily digested by the target audience. the borders, regions, numbers and void values need to be distinct and observable to those trying to understand what the AI algorithms is trying to achieve, and if the algorithms are doing so correctly. that makes this element of the product integral to the development of my project.

Implementation of Sudoku Interface

In order to implement this feature, I have decided to initially work on a console based interface. I felt this was necessary as my first iteration of the sudoku interface only needs to display an easily readable representation of the sudoku puzzle. this was achievable in the console as the borders can be represented through the '|' character for row division and '—' character for column division, the numbers as just numbers and the void values as an '*' character. this gave a simple and easy to read sudoku display on the console.

Planned Improvements to the Interface

Once the other features of my project have become slightly more developed, I plan to return to improve the sudoku interface through various means, such as randomised sudoku generation of varying difficulty, as well as a better user interface to display the sudoku rather than primarily using the console, which works well and is readable, but is not the best means of displaying the puzzle in a visually appealing and user friendly fashion.

Solving Algorithm

How a Backtracking Algorithm could Solve a Sudoku

A backtracking works similar to how a human would back track. The algorithms makes an attempt to solve a problem. so in the case of a sudoku it would attempt to fill in blank squares with a potential solution, and when a problem occurs in its solution, it goes back to the cause of the problem, which would be the invalid potential solution in that square or a previous one, and continues its attempt with a different solution for the sudoku, and this is done repeatedly till a viable solution is formed.

Implementation of Backtracking Algorithm

When developing the solving algorithm, first came the crucial element of finding a blank square in the grid, this would be represented by a 0 in the sudoku grid array. the coordinates on the grid would be returned to the main algorithm when found, then using these, the main loop will attempt to fill the square with numbers ranging from 1 to the max value, which is

also the length of the grid (usually 9), until a viable value for the square is found (a value that is not repeated in the row, column or sub-grid). if a viable value cannot be found, the algorithm clears the current square and backtracks to the previous square, and checks the next values for viability, if there are no viable values, the algorithm will clear the current square and backtrack once again to the square before and once again try different values until a different viable value is found and the algorithm searches for the next blank value and the algorithm repeats itself, until there is no more blank squares, or no more values to try on the grid that gives a solution to the problem.

Planned Improvements for the Backtracking Algorithm

In order to improve the speeds of my backtracking algorithm, I would need to look into my uses of exhaustive searches which is used a lot in my algorithm for different functionalities such as, finding an empty square by looping through every value in the grid array, checking validity of a value by searching through every value in the row, column and sub-grid for a repeated value. this is very inefficient, and I believe this can be rectified by maybe implementing faster search algorithms or making use of other methods and techniques to make the searches slightly quicker and efficient, which is necessary to prove AI's superior problem solving capabilities.

Improvement of the User Interface

Creating a Graphical User Interface

When creating my graphical user interface to display the sudoku and the solving algorithm, I decided to pygame as it provides an adaptable environment in which the sudoku grid can be drawn on the window and updated with little difficulty while still being able to display a way of visualising the backtracking algorithm. when creating the grid, I decided to give it the length 340 which was big enough for the user to see the sudoku and the values in each square clearly. When dividing the grid into sub-grids I used the same divisor logic used in the console print function, however I divided the grid visually using thicker lines to create the division between the whole and sub grids. I then started adding values to square by placing a new square over the empty square with the intended value, this could be cleared through overwriting the square by filling the surface with a white square then updating the value to a new square to cover the cleared square. this function would need to be called every time the value changed in the algorithm in order to give the visual backtracking effect.

The Benefits of Having a GUI to Display the Problem Solving Algorithm

By introducing a GUI to display the solving algorithm, I am able to show the step by step solution being formed through mistake adaptation approach my backtracking algorithm takes as well as showing how quickly the computer can calculate this as the GUI updates the values very rapidly, demonstrating how much faster a computer is compared to a human, despite the GUI technically wasting computer processing speed, so it is faster to print the output directly then updating a GUI, but then the algorithm will not be demonstrated in an easily comprehensible manner.

Planned Improvements to the GUI

In order to improve the Graphical User Interface I plan to make use of radio buttons or checkboxes to select which algorithm to use when solving the puzzle, I also feel a reset button to clear the grid and start button to start the algorithm would also be a nice touch, and will make the application experience more convenient. Additional text would also be needed, and useful for displaying the time it takes for solving the problem as well as potentially a mistakes counter for how efficiently/accurately it initially solves the problem.

Next Solving Algorithm

How could a Neural Network Solve a Sudoku

My focus moving forwards will be on developing a Neural Network that will be trained on taking in sudoku solutions, this is so that it will learn what a solution to a sudoku should look like and it should recognise the patterns/rules of a sudoku so that when it is faced against an incomplete sudoku, it will be able to provide an accurate prediction of a solution, that will hopefully be the correct answer to the inputted problem.

Layers of the Network

Conv2D is used to extract features, makes use of filters called kernels to produce feature maps, which aids in pattern recognition in the network. the use of filters allows for parameter sharing which means less parameters are needed for the network to generalise better as filters cover multiple parts of the input. currently uses varied filter sizes and kernel sizes 3x3.

BatchNormalisation aims to standardise the inputs so that the process of training becomes more stable and efficient. this allows for a higher rate of learning and quicker training processes, it also helps to prevent overfitting by adding on a bit of "noise" to the input of each layer. does this through working out the mean and variations of the inputs, then normalises based on the results.

MaxPooling2D works by reducing spacial dimensions of the input data in order to reduce complexity and memory consumption in computations. it does this by keeping the maximum value from the pooling window, which helps to highlight prominent features, which improves generalisation and reduce overfitting.

Dropout hopes to achieve a reduction in overfitting by randomly dropping out neurons (resetting to 0) so that the network is forced to learn redundant representations of the data, also allowing for better generalisation by preventing network from being too specialised to the training data. dropout can aid in reducing co-adaptation which is where neurons depend on other specific neurons, this means single neurons can learn more robust features independently.

Flatten layer converts the multidimensional data into a singular dimension which is what is needed as input for the dense layers of the CNN so that they can process and learn from the test data.

Dense layers are fully connected layers as each neuron in this layer is connected to every neuron in the previous layer, so every input now has an impact of every output, which gives the model the ability to learn the patterns and complex relations of the training data.

Types of Activations

Activation Functions this is the function in which the neurons in the network uses to learn and make decisions as it determines which neurons should activate and pass data to other layers to form the final prediction.

ReLU this is an activation function which works by taking in the input x and setting the output to x if positive (output is same as input) and if x is negative ($x \leq 0$) the output of the output is 0, this is a simplistic activation which is great for computational efficiency however, can lead to dying ReLU problem were neurons set to 0 are inactive consistently and therefore not learning.

Leaky ReLU this is a variant of ReLU which attempts to address the dying ReLU problem by allowing for a very small non zero values which is calculated by multiplying the input with a very small constant (for example 0.01) if the input is $x \leq 0$ and just outputting x if the input is positive. this prevents neurons from being repeatedly deactivated in the network as they avoid the output being set to 0.

ELU this activation also aims to prevent the dying ReLU problem by using an exponential function on inputs $x \leq 0$, this allows for the output to avoid being set to zero, as it is set closer to 0 but not actually 0, so the dying neuron problem is avoided. otherwise, for positive inputs, the output remains the same as the input.

Softmax softmax is a final layer activation which this ensures that all output probability is normalised so that all probabilities add up to 1, so that it is perfect for classification as the probabilities are appropriately distributed over classes (perfect for sudokus as each cell has classes 1-9 as a potential solution).

Model Compilation

Network Optimizer a network optimizer is used to help adjust the weights of neurons and the biases in the neural network while in its training phase. it works to reduce the loss (errors) of the network while improving the accuracy of the predictions. it uses backpropagation to update network parameters based on gradient calculations, which in turn helps the neural network learn the dataset.

Learning Rate when using the Adam optimiser, the learning rate can effect the accuracy of the model during its learning phase, it is important as this dictates how much the models weights adapt when learning. the size in which this value is set can effect the speeds of convergence and as a result the training time, as well as how precise the updates in the weights which can improve the accuracy as well as keeping the rate of error (loss) to a minimum.

Loss Function this function is important for the training of the model, as this function is how the accuracy of the model prediction is calculated. this is needed to provide a measure on how well the model is performing on the training dataset. this function also works with the loss function to aid in the adjustment of weights and biases, to try and limit the loss which helps the model to learn more efficiently and predict more accurately. with classification models it is common to use a Cross-Entropy loss function, as this helps with the models learning process.

Model Training

Epochs each time a model trains through an entire training set, this is called an epoch. the number of epochs refers to the number of times the model will cycle through the same entire training set. having multiple epochs are important as this allows for the model to better understand the data as one train through is not typically enough for the model to learn to effectively and accurately predict. However, if there are too many epochs, there is a chance for the model to learn the specific training set and as a result will struggle to predict new examples which means it is overfitting.

Batch size this is the amount of samples the network takes from the training set and trains with in one forward and backward pass. smaller batch sizes can lead to more frequent updates to the model which can help to make better predictions as well as consuming less computational resources, but larger batches despite the greater computational demand, can train the model much faster as well as having more stable updates which give smoother convergence.

Callbacks callbacks can be helpful in the training phase of the model as it can provide a range of functionalities such as improving performance through early stoppage if validation loss stops decreasing (to avoid overfitting) and saving checkpoints for the model while training on the set so that at the end it can restore the point with the best learnt parameters.

Problems Encountered

Severe Underfitting A serious issue that came with my first few drafts at developing my model is extreme underfitting. this is where the model was failing to properly learn from the training set, this was a serious issue as if it wasn't accurately learning and predicting with the training set, when I test it on new data, the prediction I get from the model is very much inaccurate and could not even pass as an almost solved sudoku. I believe the reasons for this the low numbers of conv2D layers, as well as very small filters and kernel sizes, in addition to extremely high batch sizes. these will need to be more finely adjusted to try and discover the appropriate model for solving a sudoku puzzle.

Extreme Training Durations after adjusting the sizes of filters and conv2D layers, as well as the types of activation those layers required, I found when having too many layers, around 5 or more, and filters above 162 and greater, with kernels (5,5) and higher each and in combination gave extremely high training times per epoch, some parameters coming up to 5 hours on each epoch, which is very computationally exhaustive and in some cases not even beneficial as the models accuracy would be barely increasing in these hours, meaning the model is training past its peak which could be an indication of overfitting.

Slight Overfitting with some of my larger model attempts, I would yield better predictions to the sudokus I would pass in, however due to the long times spent training, the model would provide very high accuracy but slightly lower validation accuracy and when testing on new data would provide sudoku solutions with quite a few mistakes. I think moving forwards the model will need some parameters reduced and some of the data and layers more normalised or even dropped out in order to try and avoid neuron dependancy and overfitting in the model, which should smooth out and reduce mistakes made when solving the problem.

Potential Solutions to the Problems Encountered

Number of Conv2D Layers increasing the number of convolutional layers can increase the models ability to learn by better extracting features from the training set being passed in so that it can use those features to classify and predict far more accurate predictions. however too many layers brings overfitting and can increase computational costs significantly.

Increased Number of Filters by using more filters the convolutional layers are able to extract a deeper representation of the data, detecting patterns and features more precisely and as a result returning predictions with far less errors. although with greater increases in filters, the greater the demand on training durations, which may not give much in return.

Increased Kernel Sizes larger kernels allows for the network to capture more extensive spatial features of the training data, this allows for smoother feature extractions as well as reduction the amount of layers the network would need to capture these features. similarly to the layers and filters, too many can still increase the computational costs in training the model and this could also lead to overfitting.

Final Improvements to the User Interface

Improved Layout adjustments to the User Interface has been made to better center the sudoku on screen and also providing an instructions section to the application describing how the user can use the app, as well as providing a section in which the time the algorithm takes to provide the solution along with the number of mistakes it has made to come up with its final solution.

Use of Additional Inputs allows user to use keys on the keyboard to navigate through different features of the app, such as number keys to switch between the sudoku problems, the enter key to solve with backtracking, the space key for the Neural Network solution and the backspace to clear the current solution leaving the user back with the unsolved sudoku puzzle.