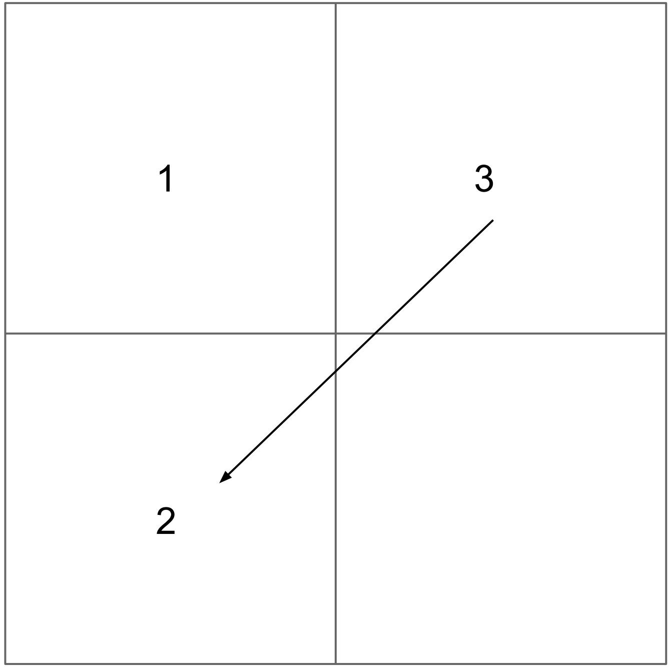
Artificial Intelligence
15 / n^2 - 1
Puzzle Solver

Ryan Toner

Introduction - Project Background

- 15 Puzzle is 4x4 matrix with tiles labeled 1 through 16 (last is blank)
- 16!/2 possible states ~1.0461395e+13 (over 10 trillion)
  - “Inversion” problem halves total number of states
- Even playing optimally on hardest configurations takes up to 80 moves
- Solving the 15 puzzle is an NP-Complete Problem
  - This means there is no polynomial time algorithm that can solve this problem
  - However, solutions can be verified in polynomial time
  - Using the boolean satisfiability problem (SAT) as black-box, it can be shown that the 15-puzzle is NP-Complete
- Solving this problem is programmatically difficult
  - <u>This project is fundamentally a research problem</u>

Example of inversion counting



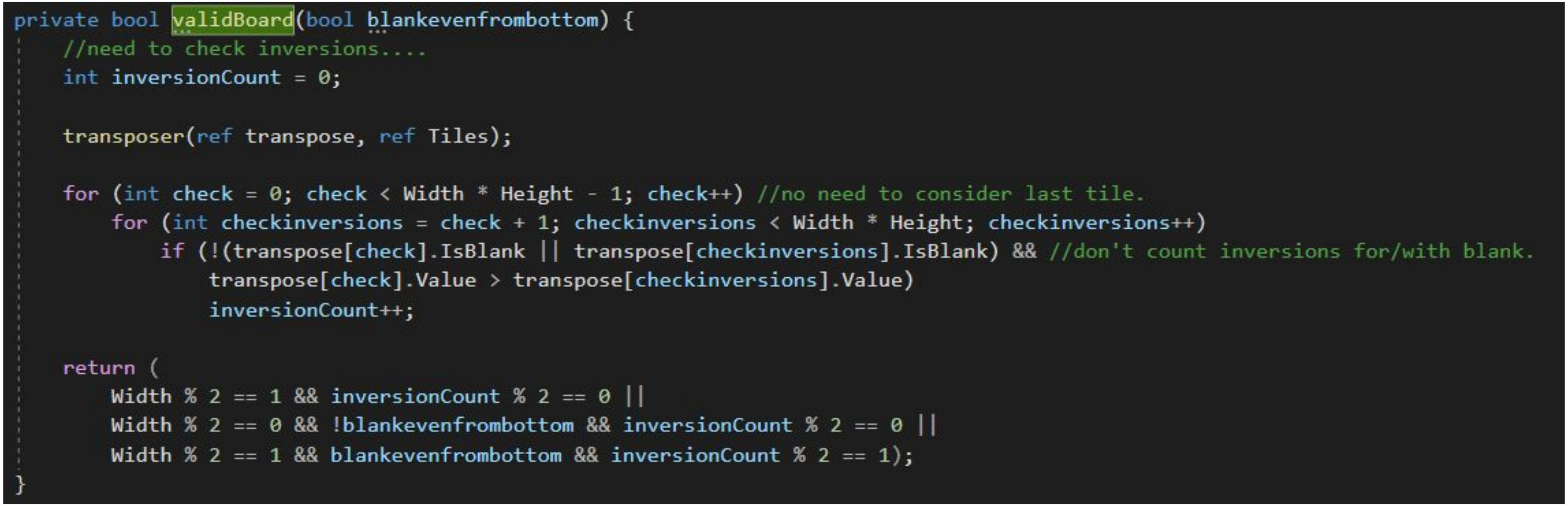
- 3 has a higher value than 2, but 3 comes before 2 in the matrix (when scanning top to bottom and left to right)
- The same notion is generalized to a 4x4 Grid
- This board has exactly 1 inversion

- Since *grid width* is *even*, and *blank tile* is *on the bottom row* (0 moves away), and the *inversion number is odd*, this board is unsolvable

Validity of Board States – “Inversion” Problem

- Inversion Count determines if board is solvable.
- **THEOREM 1**: the puzzle on any n x m board with n, m > 1 has at most (n x m)! /2 legal configurations.
- **THEOREM 2**: Any solvable configuration will remain solvable given valid moves.
- Formula for solvability:
- A board is solvable **if and only if**
  - ((grid width odd) && (# inversions even)) ||
  - ((grid width even) && ((blank on odd row from bottom) == (#inversions even)))

Formula to count inversions



```
private bool validBoard(bool blankevenfrombottom) {
    //need to check inversions....
    int inversionCount = 0;

    transposer(ref transpose, ref Tiles);

    for (int check = 0; check < Width * Height - 1; check++) //no need to consider last tile.
        for (int checkinversions = check + 1; checkinversions < Width * Height; checkinversions++)
            if (!(transpose[check].IsBlank || transpose[checkinversions].IsBlank) && //don't count inversions for/with blank.
                transpose[check].Value > transpose[checkinversions].Value)
                inversionCount++;

    return (
        Width % 2 == 1 && inversionCount % 2 == 0 ||
        Width % 2 == 0 && !blankevenfrombottom && inversionCount % 2 == 0 ||
        Width % 2 == 1 && blankevenfrombottom && inversionCount % 2 == 1);
}
```

15 Puzzle Web App (Technologies Used)

- Using a client-server model
- Client runs in browser using HTML, CSS, Bootstrap, Javascript, and AJAX to communicate with server
- Server is IIS Express using ASP.NET with C# and Razor backend
  - Code written using Model-View-Controller architecture
  - Model represents board state
  - Controller handles requests
  - View generates the initial client view data
  - Server calls local python code, including loading the neural network and pathfinding algorithms

Server

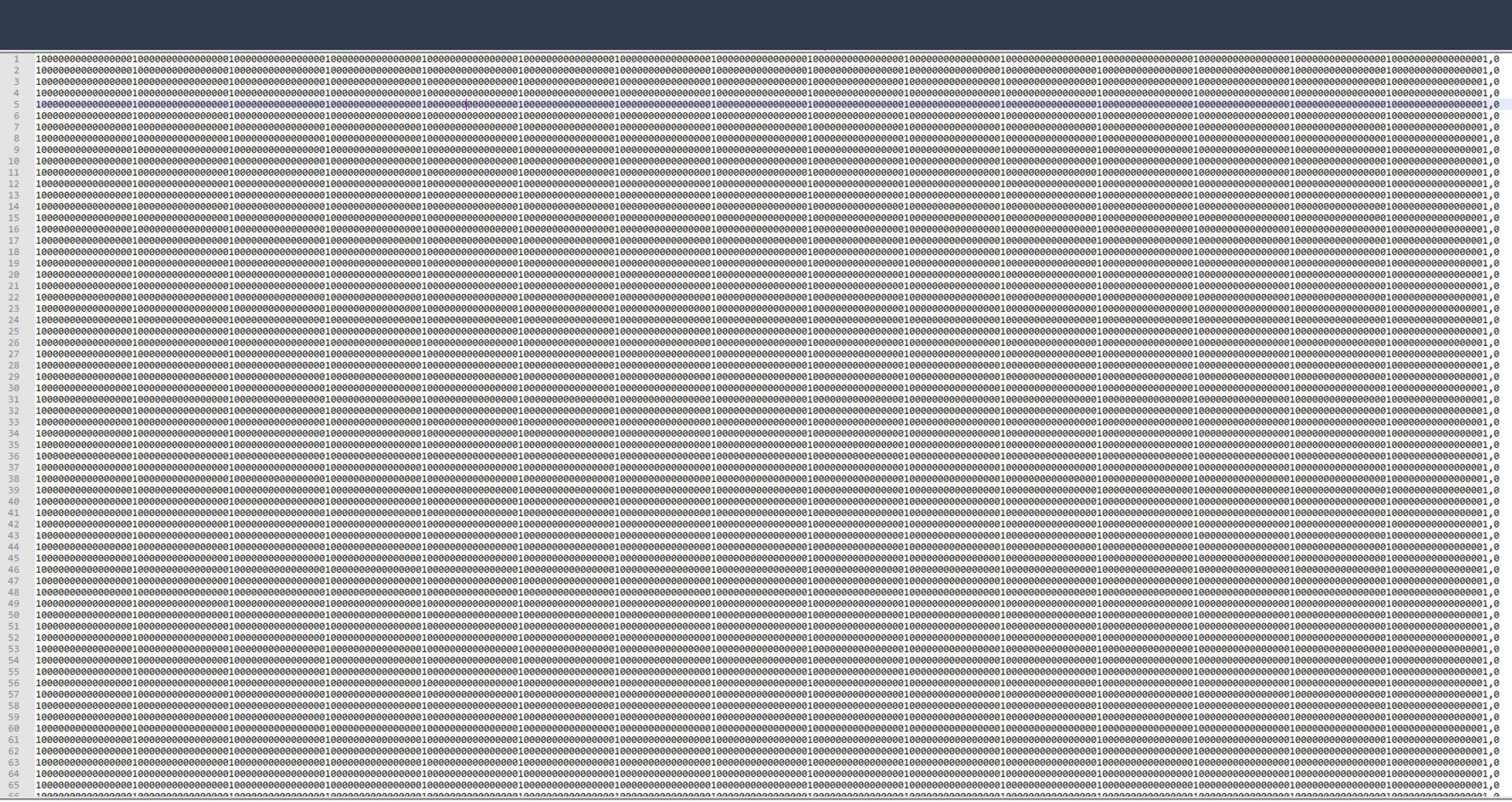
- Running C# and Object-Oriented Components that generate the board and run the core of the game
    - Serves the HTML to the client
    - Creates instance of “board” in backend
    - Takes AJAX requests from server e.g. “Click,” “ArrowKey,” and “Solve” that return the valid updated board state
- Runs python code in anaconda environment using command line
    - Reads board state
    - Exports moves to solve the puzzle
    - Using Tensorflow 2.0 GPU with Nvidia CUDA and CuDNN

Training Algorithm

- Created independently of the application
- Start with 4x4 solved board and use deterministic graph “walking” with BFS to generate valid board states
    - Uses Queue to perform BFS
    - Does not have to worry about inversions (since the initial state is solved)
- Moves up to 50 moves away with a limit of 50,000 moves per distance value
- Generates over 2 million boards in ~520mb training.csv file
- Uses data augmentation technique:
    - Some distances (e.g. 0, 1, 2, 3, 4, 5) have less than 50,000 possible boards
    - while(# moves for the distance value < ⅖ (50,000))
        - Cyclically print existing data in modular cycle
    - This technique prevents skewing and over-fitting to the training dataset using boards that are many moves away
- The algorithm ensures that it will not “refind” an existing board by using a hashset

Training Algorithm Encoding

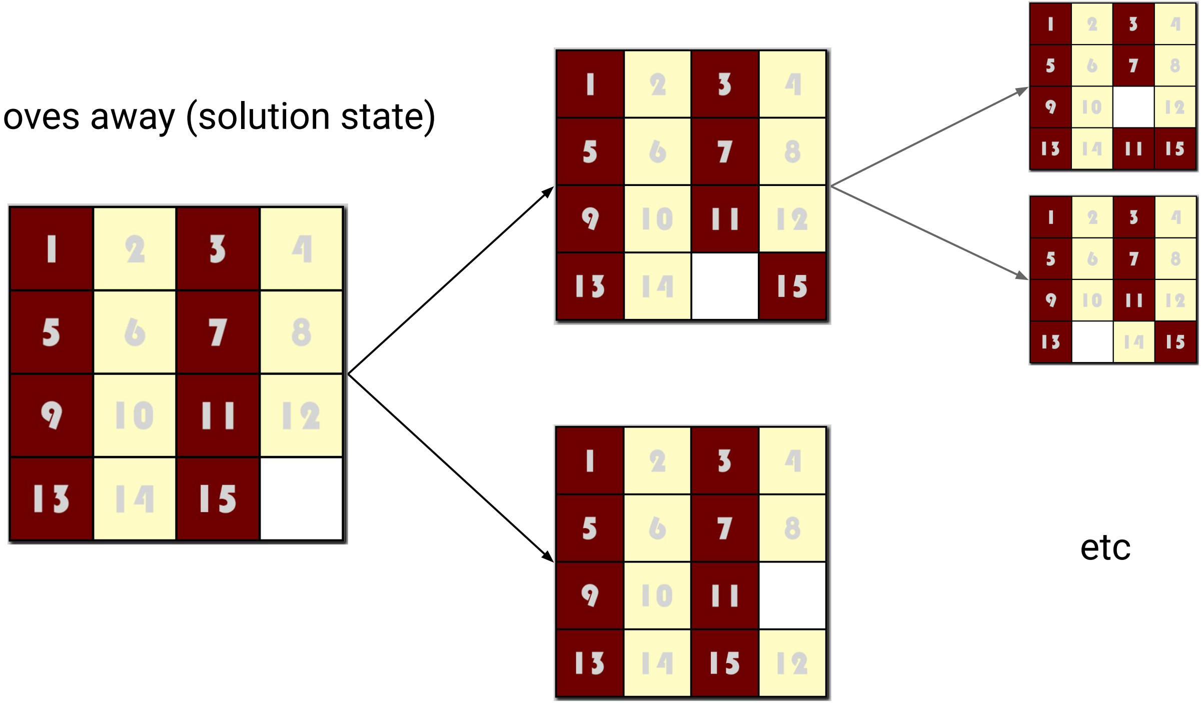
- The board is in 4x4 matrix configuration
- Each tile can occupy 1/16 spots
- Use 1-hot encoding to represent the the tile's location
    - Each tile needs 16 numbers
    - The board has 16 tiles
    - The total board encoding is 256 = (16^2)
- The board encoding is our "features," or input
- The number of moves from the solution is the "target," or output
- [0010000000000000…… (256), 15]
- ^ features ^ target



```
1000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001,0
1000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001,0
1000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001,0
1000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001,0
1000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001,0
1000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001,0
1000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001,0
1000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001,0
1000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001,0
1000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001,0
1000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001,0
1000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001,0
1000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001,0
1000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001,0
1000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001,0
1000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001,0
1000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001,0
1000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001,0
1000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001,0
1000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001,0
1000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001,0
1000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001,0
1000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001,0
1000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001,0
1000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001,0
1000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001,0
1000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001,0
1000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001,0
1000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001,0
1000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001,0
1000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001,0
1000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001,0
1000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001,0
1000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001,0
1000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001,0
1000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001,0
1000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001,0
1000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001,0
1000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001,0
1000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001,0
1000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001,0
1000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001,0
1000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001,0
1000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001,0
1000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001,0
1000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001,0
1000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001,0
1000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001,0
1000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001,0
1000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001,0
1000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001,0
1000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001,0
1000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001,0
1000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001,0
1000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001,0
1000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001,0
1000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001,0
1000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001,0
1000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001,0
1000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001,0
1000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001,0
1000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001,0
1000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001,0
1000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001,0
1000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001,0
1000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001,0
```

Training Algorithm Diagram

0 Moves away (solution state)



1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	

1	2	3	4
5	6	7	8
9	10	11	12
13	14		15

1	2	3	4
5	6	7	8
9	10	11	
13	14	15	12

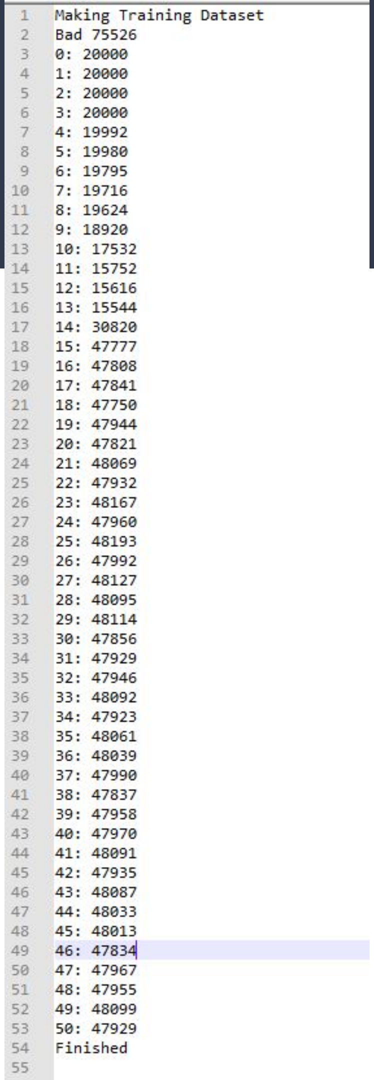
1	2	3	4
5	6	7	8
9	10		12
13	14	11	15

1	2	3	4
5	6	7	8
9	10	11	12
13		14	15

etc

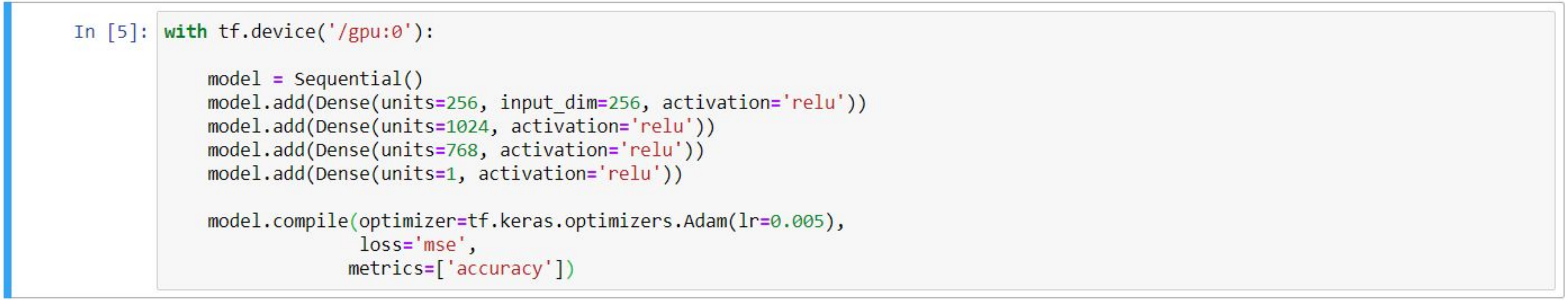
Training Algorithm Code

- https://github.com/RyanTonerCode/15-Presentation



```
Making Training Dataset
Bad 75526
0: 20000
1: 20000
2: 20000
3: 20000
4: 19992
5: 19980
6: 19795
7: 19716
8: 19624
9: 18920
10: 17532
11: 15752
12: 15616
13: 15544
14: 30820
15: 47777
16: 47808
17: 47841
18: 47750
19: 47944
20: 47821
21: 48069
22: 47932
23: 48167
24: 47960
25: 48193
26: 47992
27: 48127
28: 48095
29: 48114
30: 47856
31: 47929
32: 47946
33: 48092
34: 47923
35: 48061
36: 48039
37: 47990
38: 47837
39: 47958
40: 47970
41: 48091
42: 47935
43: 48087
44: 48033
45: 48013
46: 47834
47: 47967
48: 47955
49: 48099
50: 47929
Finished
```

Neural Network

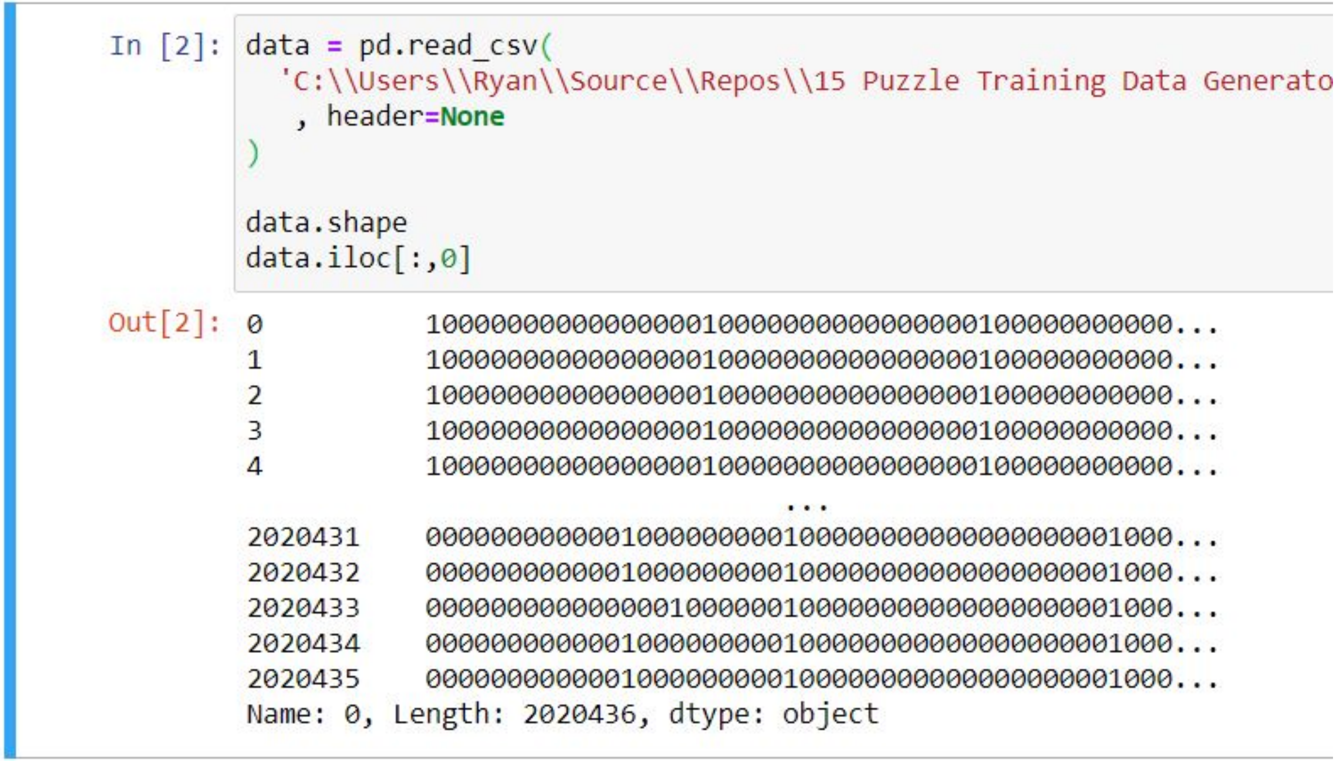


```
with tf.device('/gpu:0'):

    model = Sequential()
    model.add(Dense(units=256, input_dim=256, activation='relu'))
    model.add(Dense(units=1024, activation='relu'))
    model.add(Dense(units=768, activation='relu'))
    model.add(Dense(units=1, activation='relu'))

    model.compile(optimizer=tf.keras.optimizers.Adam(lr=0.005),
                  loss='mse',
                 metrics=['accuracy'])
```

Neural Network Architecture

- Layers:
  - Input Layer (256 dense units), relu activation
  - Hidden Layer 1 (1024 dense units), relu activation
  - Hidden Layer 2 (768 dense units), relu activation
  - Output Layer (1 dense output unit), relu activation (better performance over linear here)
- Parameters:
  - Adam optimizer with LR = 0.005, using MSE loss function
  - Max epochs set to 50 (only ran ~20)
  - Batch size of 45 to handle large dataset
- Tools:
  - Pandas to read .csv
  - Numpy to handle array manipulation
  - Sklearn for train_test_split
  - Tensorflow 2.0 with Keras backend

Input Data



In [2]:
```
data = pd.read_csv(
  'C:\\Users\\Ryan\\Source\\Repos\\15 Puzzle Training Data Generato
   , header=None
)

data.shape
data.iloc[:,0]
```

Out[2]:
```
0          1000000000000000010000000000000000100000000000...
1          1000000000000000010000000000000000100000000000...
2          1000000000000000010000000000000000100000000000...
3          1000000000000000010000000000000000100000000000...
4          1000000000000000010000000000000000100000000000...
                                  ...
2020431    0000000000001000000000100000000000000000001000...
2020432    0000000000001000000000100000000000000000001000...
2020433    0000000000000001000000100000000000000000001000...
2020434    0000000000001000000000100000000000000000001000...
2020435    0000000000001000000000100000000000000000001000...
Name: 0, Length: 2020436, dtype: object
```

Preparing the training data

```
x_data = data.iloc[:, 0]

x_train = []

for row in x_data:
    a = []
    for s in str(row):
        a.append(int(s))
    x_train.append(a)

dt = np.dtype('i4')

x_train = np.array(x_train, dtype=dt)

y_train = data.iloc[:,1].to_numpy()
```

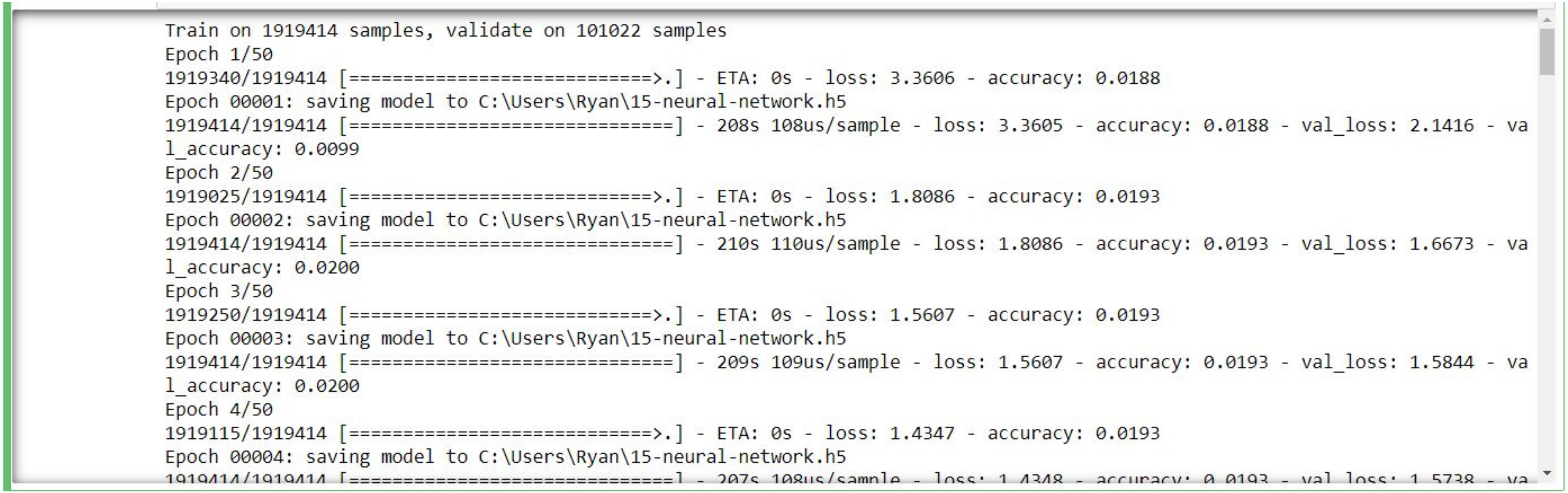
Preparing the training data

```
x_train1, x_valid, y_train1, y_valid =

    train_test_split(x_train, y_train, test_size=0.05, shuffle= True)
```

Saving the network

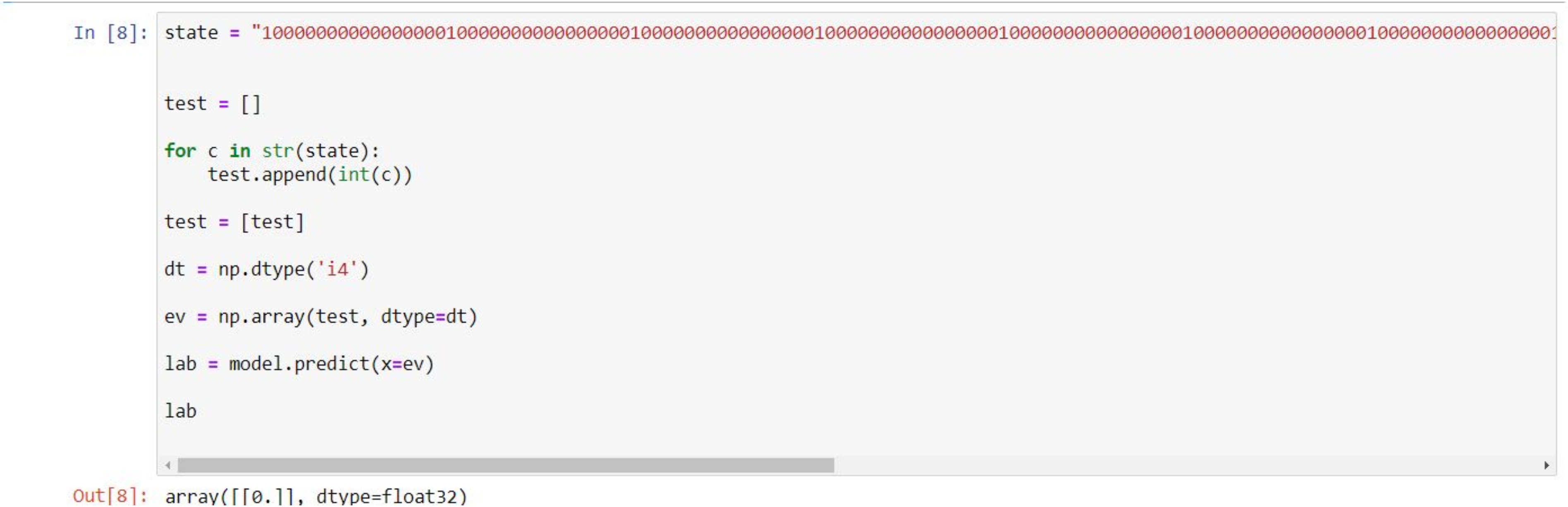
```
Cp_callback =
tf.keras.callbacks.ModelCheckpoint(filepath=best_model_filename,
save_weights_only=True, verbose=1)


val_data = (x_valid, y_valid)

model.fit( x=x_train1, y=y_train1, verbose=1,
           epochs=50, shuffle=False, validation_data = val_data,
           callbacks=[cp_callback], batch_size=45
)
```

Network Results



```
Train on 1919414 samples, validate on 101022 samples
Epoch 1/50
1919340/1919414 [============================>.] - ETA: 0s - loss: 3.3606 - accuracy: 0.0188
Epoch 00001: saving model to C:\Users\Ryan\15-neural-network.h5
1919414/1919414 [==============================] - 208s 108us/sample - loss: 3.3605 - accuracy: 0.0188 - val_loss: 2.1416 - va
l_accuracy: 0.0099
Epoch 2/50
1919025/1919414 [============================>.] - ETA: 0s - loss: 1.8086 - accuracy: 0.0193
Epoch 00002: saving model to C:\Users\Ryan\15-neural-network.h5
1919414/1919414 [==============================] - 210s 110us/sample - loss: 1.8086 - accuracy: 0.0193 - val_loss: 1.6673 - va
l_accuracy: 0.0200
Epoch 3/50
1919250/1919414 [============================>.] - ETA: 0s - loss: 1.5607 - accuracy: 0.0193
Epoch 00003: saving model to C:\Users\Ryan\15-neural-network.h5
1919414/1919414 [==============================] - 209s 109us/sample - loss: 1.5607 - accuracy: 0.0193 - val_loss: 1.5844 - va
l_accuracy: 0.0200
Epoch 4/50
1919115/1919414 [============================>.] - ETA: 0s - loss: 1.4347 - accuracy: 0.0193
Epoch 00004: saving model to C:\Users\Ryan\15-neural-network.h5
1919414/1919414 [==============================] - 207s 108us/sample - loss: 1.4348 - accuracy: 0.0193 - val_loss: 1.5738 - va
```

Network Results



```
In [8]: state = "100000000000000001000000000000000010000000000000000100000000000000001000000000000000010000000000000000100000000000000001

        test = []

        for c in str(state):
            test.append(int(c))

        test = [test]

        dt = np.dtype('i4')

        ev = np.array(test, dtype=dt)

        lab = model.predict(x=ev)

        lab

Out[8]: array([[0.]], dtype=float32)
```

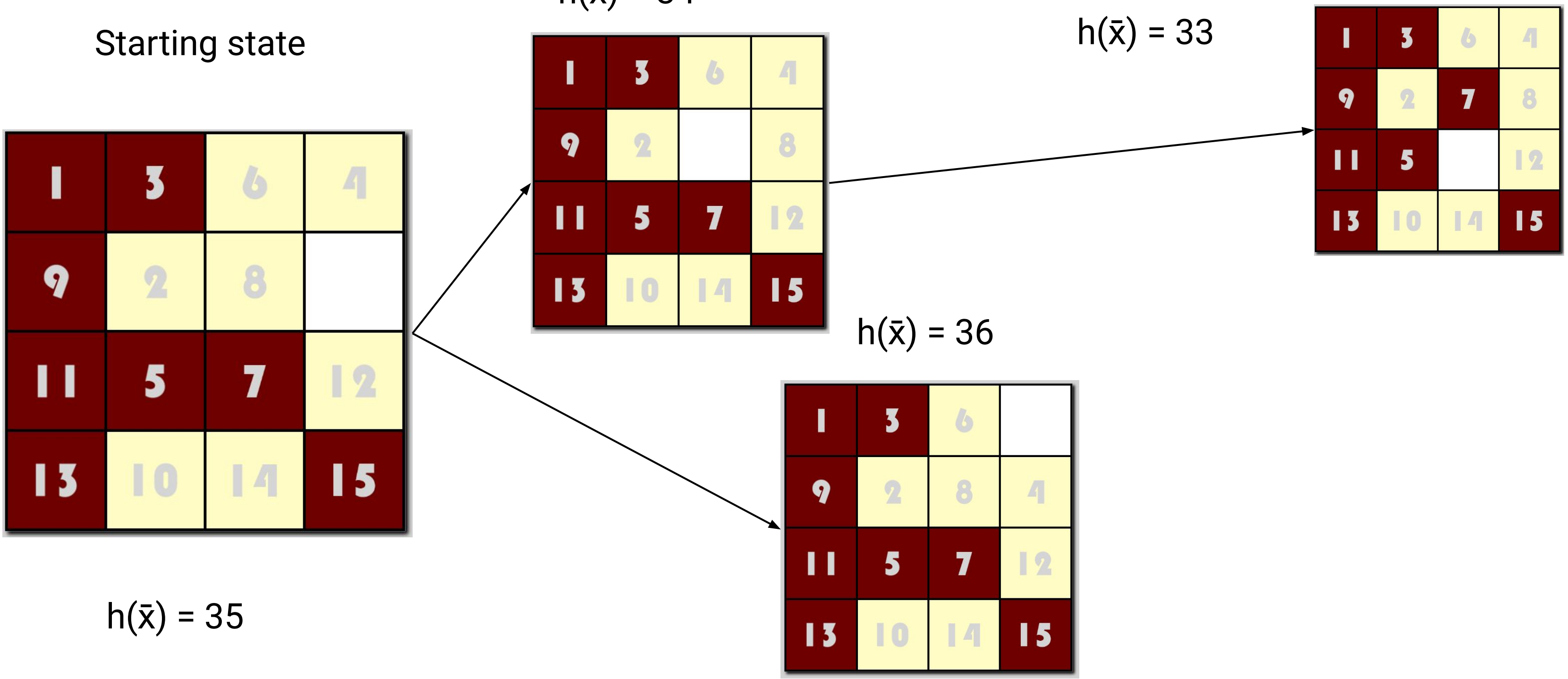
What does this network get us?

- From a given board input, we can use this semi-linear regression model to predict the number of moves it should take to solve the board.
- *Okay… that’s great, but how does this help you solve the game?*
- We can use a pathfinding algorithm similar to the way we trained the data to solve the puzzle
- Use a modified A* Algorithm with our “neural” heuristic function

Heuristic Function Explained

- A heuristic function is a function that measures how good something is
- In this case, smaller values mean you are closer to solving the puzzle
- Suppose $\bar{x}$ is a board and h is our heuristic function
- Then $h(\bar{x})$ is the measurement of how close the board $\bar{x}$ is from being solved
- The domain of this function is boards
- The codomain of this function is a subset of the real numbers from around [0,250]
    - Recall 80 move maximum for optimal playing

Using pathfinding and heuristic (example)



Starting state

1	3	6	4
9	2	8	
11	5	7	12
13	10	14	15

$h(\bar{x}) = 35$

$h(\bar{x}) = 34$

1	3	6	4
9	2		8
11	5	7	12
13	10	14	15

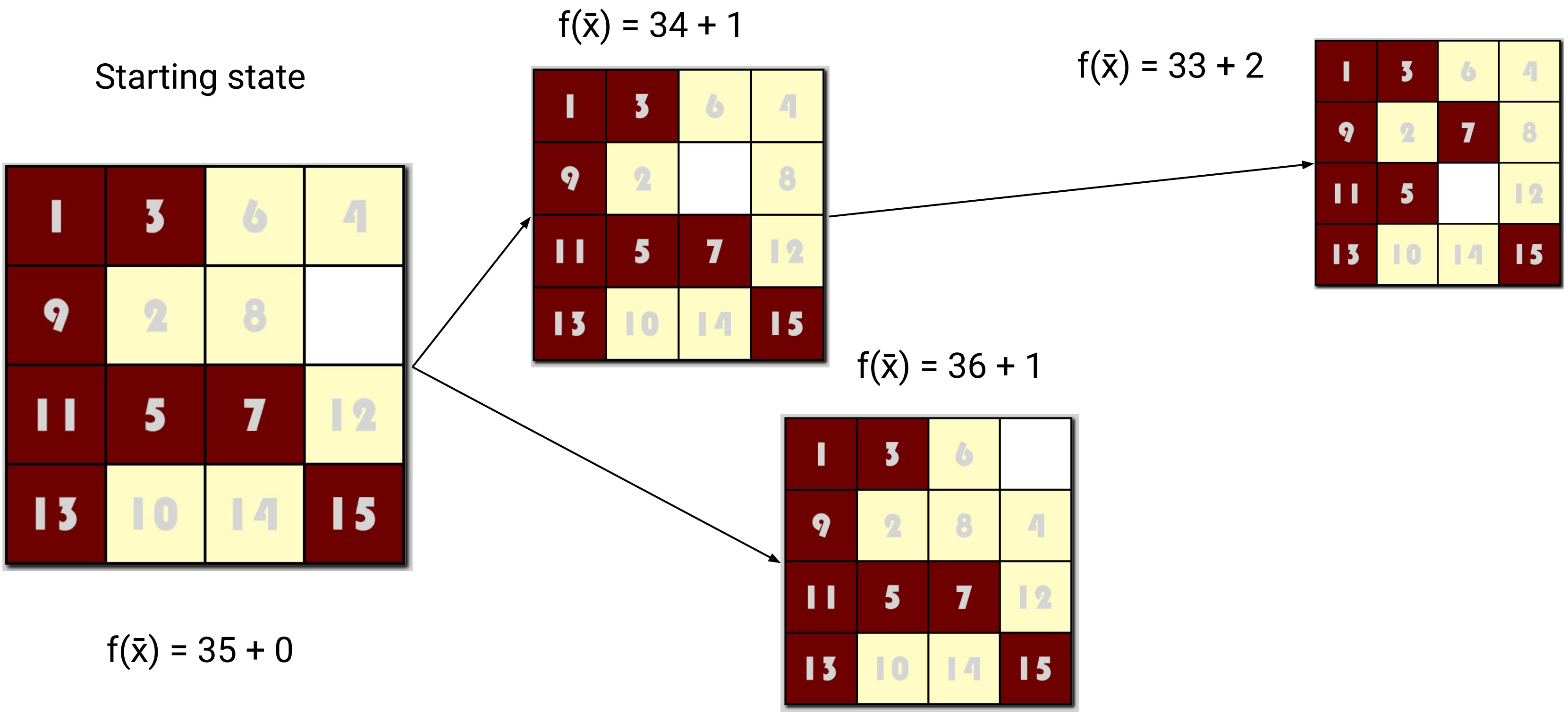
$h(\bar{x}) = 36$

1	3	6	
9	2	8	4
11	5	7	12
13	10	14	15

$h(\bar{x}) = 33$

1	3	6	4
9	2	7	8
11	5		12
13	10	14	15

Using pathfinding and heuristic (example)



Starting state

1	3	6	4
9	2	8	
11	5	7	12
13	10	14	15

$f(\bar{x}) = 35 + 0$

$f(\bar{x}) = 34 + 1$

1	3	6	4
9	2		8
11	5	7	12
13	10	14	15

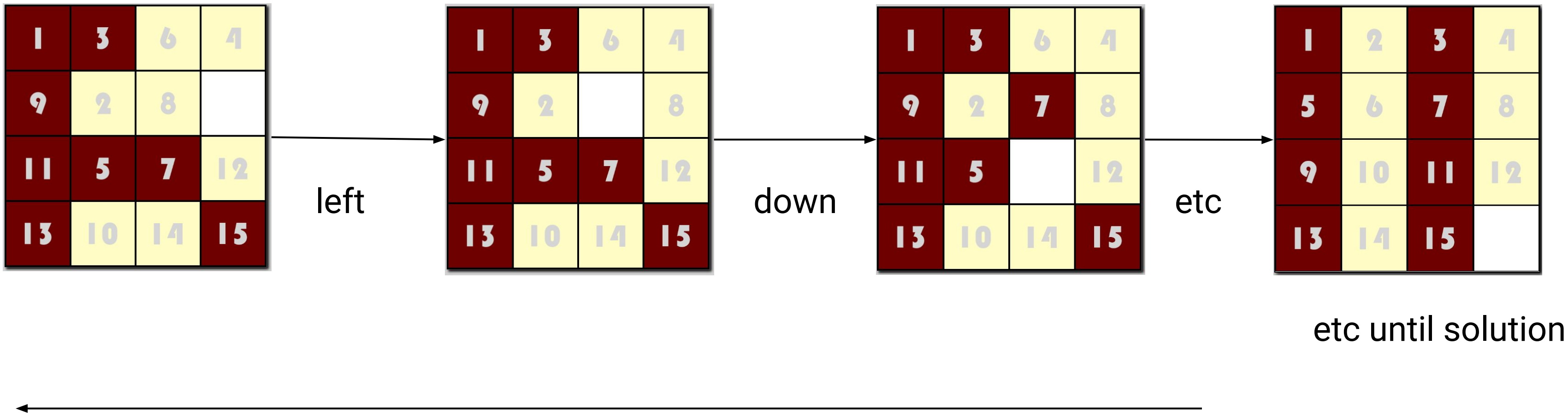
$f(\bar{x}) = 33 + 2$

1	3	6	4
9	2	7	8
11	5		12
13	10	14	15

$f(\bar{x}) = 36 + 1$

1	3	6	
9	2	8	4
11	5	7	12
13	10	14	15

Backtracking for solution



1	3	6	4
9	2	8	
11	5	7	12
13	10	14	15

left

1	3	6	4
9	2		8
11	5	7	12
13	10	14	15

down

1	3	6	4
9	2	7	8
11	5		12
13	10	14	15

etc

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	

etc until solution

To get the solution for the puzzle, work using backtracking from the nodes of the solution path (a subset of the A* Priority Queue Graph) and collect the moves from board to board. Then, reverse this list and output the moves.

Heuristic evaluated

- Generally finds solutions in only a few hundred boards using A*
  - This is very good and much better than manhattan distance (which may take hundreds of thousands)
- Greedy pathfinding algorithm can also be used, but it finds sub-optimal solutions
- If the move count is not predicted correctly for the initial state, there are two cases:
  - 1. The network underestimates the total number of moves. In this case, A* will have to work harder to catch up and compute greater path lengths until it reaches the prediction
  - 2. The network overestimates the total number of moves. In this case, it may affect the ability for A* to find optimal solutions.
- Case 1 is performance related
- Case 2 is accuracy/ efficiency related
- We can use this to train different networks that change the heuristic goals!

What Changed?

- Using function $f(\bar{x}) = h(\bar{x}) + g(\hat{y})$
  - $h(\bar{x})$ is still the heuristic
  - $g(\hat{y})$ is new function that represents the pathlength for a path $\hat{y}$
- Why?
  - This $f(\bar{x})$is better at finding optimal solutions over $h(\bar{x})$ alone
  - $g(\hat{y})$ metric “punishes” taking long paths and forces algorithm to explore other states
  - $g(\hat{y})$ prevents bias to perform depth-first search in a priority-queue
  - As $h(\bar{x})$ decreases, $g(\hat{y})$ will increase
- Adding 1 for the pathlength?
  - $g(\hat{y})$ adds exactly one for each new board in the path, which makes physical sense

A* Algorithm

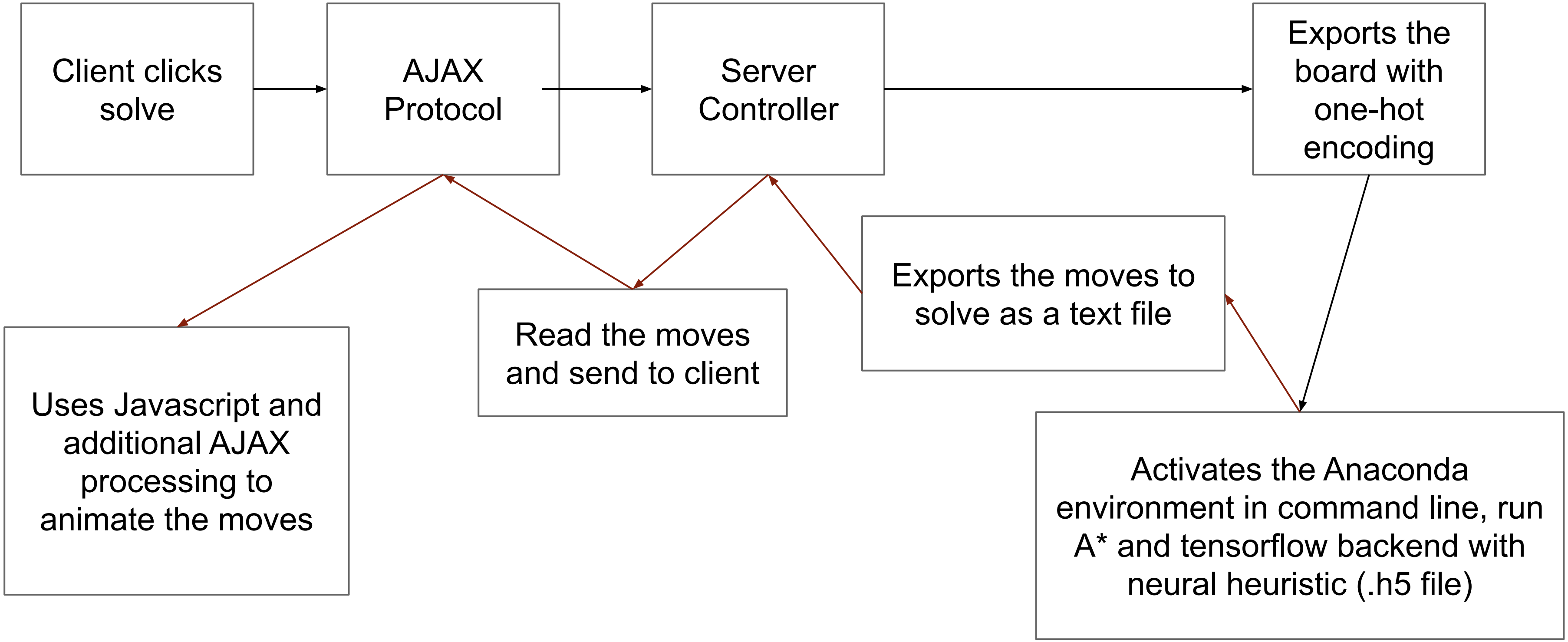
- https://github.com/RyanTonerCode/15-Presentation/blob/master/Python/intelligence.py
- Uses priority queue and graph data structure
- Path (aka the solution to the puzzle) is found using backtracking from the solution board back to the initial unsolved state, and reversing the list
- Prevents re-exploring already found states

Python backend

- Loads the saved neural-network model
- Loads the board state saved by the server
- Like the training algorithm, capable of generating valid moves
- Processes these moves using A* to find solution
- Prints the solution moves as a text file

Demo

Solve Button Mechanics



Client clicks solve

AJAX Protocol

Server Controller

Exports the board with one-hot encoding

Activates the Anaconda environment in command line, run A* and tensorflow backend with neural heuristic (.h5 file)

Exports the moves to solve as a text file

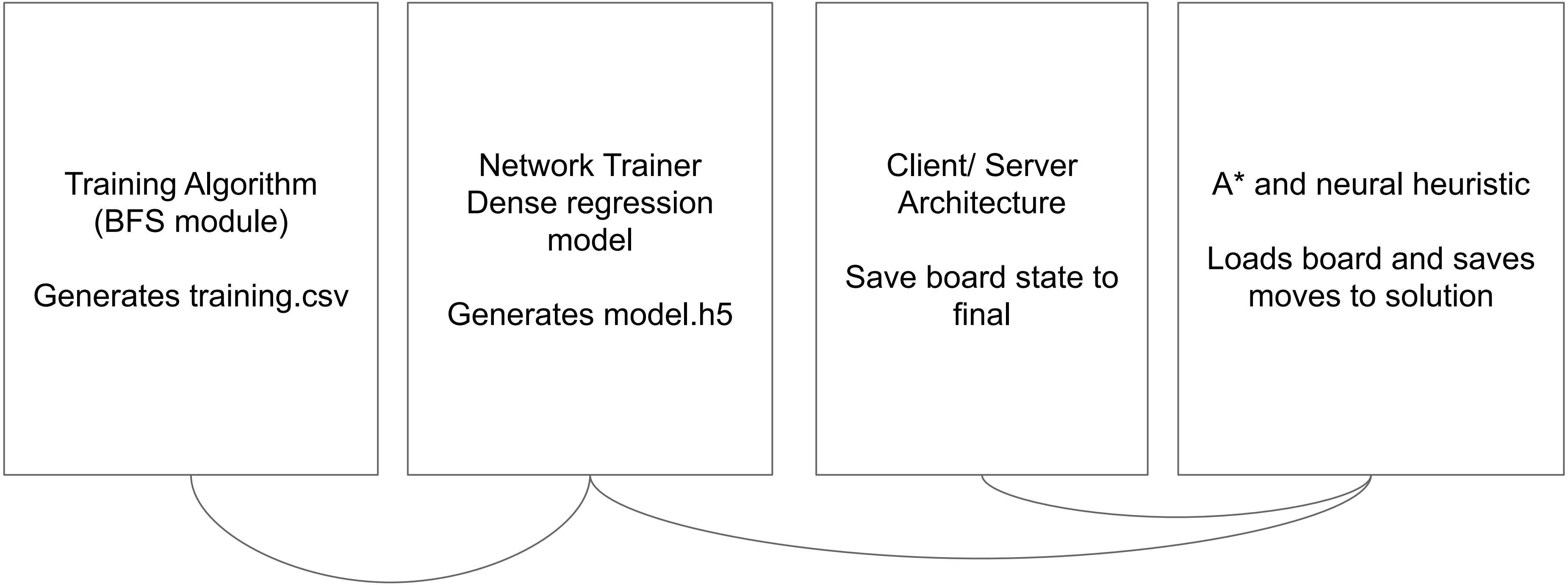
Read the moves and send to client

Uses Javascript and additional AJAX processing to animate the moves

Technical Challenges

- Requires many components to get this to work
- Not using inter-process communication, so instead I am using text files as the memory communication between the various stages of the pipeline
- Dataset takes time to generate
- Network takes long time to train

Recap: High-Level Overview



Training Algorithm
(BFS module)

Generates training.csv

Network Trainer
Dense regression model

Generates model.h5

Client/ Server Architecture

Save board state to final

A* and neural heuristic

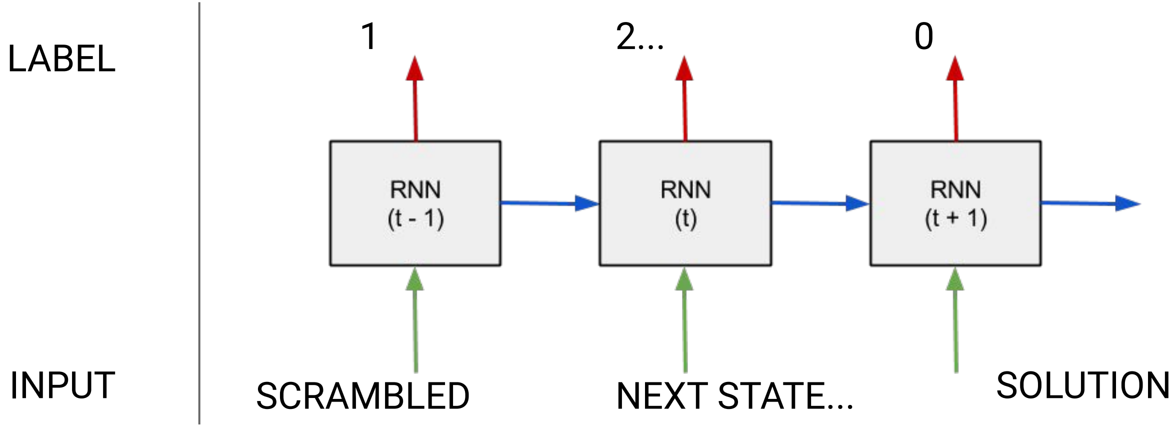
Loads board and saves moves to solution

Conclusions and Lessons Learned

- As a proof-of-concept, this project exceeded my expectations
- The network runs extremely quickly and efficiently given random board states
- The neural-network completely outperforms other heuristic functions such as Manhattan-Distance
- Solutions found by the network exhibit “creativity”
    - Uses the higher-dimensional space of the neural network to find extremely clever solutions
    - Does not solve boards like a human (it clearly bested me)
- Proves that although solving the 15-puzzle is NP-complete, we can use AI, graph-theory, and machine-learning to solve difficult configurations
- A basic neural network architecture is sufficient as a proof-of-concept
    - We could easily add additional training data up to 80 moves and improve the performance

Future Research (RNN)

- We could try using time series with a recurrent neural networks with dynamic, variable-length output sequences for a research project
- In essence, this technique could map moves (up, down, left, right) with boards like a “sentence”
- We can use our existing work to generate more sophisticated data sets



LABEL

1

2...

0

RNN (t - 1)

RNN (t)

RNN (t + 1)

INPUT

SCRAMBLED

NEXT STATE...

SOLUTION

Cited Sources

1. https://www.cs.bham.ac.uk/~mdr/teaching/modules04/java2/TilesSolvability.html
2. http://kevingong.com/Math/SixteenPuzzle.html
3. https://medium.com/breathe-publication/solving-the-15-puzzle-e7e60a3d9782
4. https://github.com/prestoj/15-puzzle