The deadline for submitting this assignment is October 23.

Consider the game of tic-tac-toe. The game board is a 3x3 grid, all nine cells of which are empty in the beginning. Two players, 1 and 2, take turns to mark the cells of the board one by one. The game is won by the player who manages to mark all three cells in a row or column or across any of the diagonals.

Consider the flattening of the game board grid into a vector, the first three elements of which denotes the top-row, the next three elements denote the middle row, and the last three elements denote the last row. Like this: $[(1,1),(1,2),(1,3),(2,1),(2,2),(2,3),(3,1),(3,2),(3,3)]$.

A configuration in the game board is a vector of nine values (taken from 0, 1, and 2 – to denote that the corresponding cell is empty, marked by player 1, and marked by player 2, respectively). For example, a configuration looks like this: $[1,0,2,0,1,2,0,0,0]$.

Let us consider only those configurations where the number of 1's and 2's are equal, both either two or three, and there's at least one unmarked cell (i.e., a cell with a 0). Let us call these as *valid* configurations. We want to train a network to predict a *good* move for player 1, given a valid configuration. A *move* here would simply mean that one of the 0's in the vector is replaced by a 1. We call such a move *bad* if:

- there's a way for player 1 to win at this stage and it does not make a winning move, *or*
- there's no way for player 1 to win at this stage; however, player 1's move allows player 2 to win from the resulting configuration in one move.

A move which is not *bad* is classified as *good*. Your task is to do the following.

1. Create data (training and testing) for this using z3. Essentially, you need to generate a tuple (config, index-for-player-1-move), and its classification as good or bad. For example:

   $([1,0,2,0,1,2,0,0,0],8) \rightarrow$ good
   $([1,0,2,0,1,2,0,0,0],7) \rightarrow$ bad
   $([1,0,2,0,1,2,0,0,0],1) \rightarrow$ bad
   $([1,1,2,0,0,2,0,0,0],8) \rightarrow$ good
   $([1,1,2,0,0,2,0,0,0],4) \rightarrow$ bad
   $([1,1,2,0,0,2,0,0,0],6) \rightarrow$ bad

   Notice, however, that the marking of 1 and 2 is symbolic. So, it might be better to use the following encoding for symbols:

   $0 \rightarrow 100, \quad 1 \rightarrow 010, \quad 2 \rightarrow 001$

   and, similarly, 9 bits to encode the index corresponding to the cell being marked by player 1. Like this:

   $8 \rightarrow 000000001, \quad 7 \rightarrow 000000010, \quad 6 \rightarrow 000000100$
   $5 \rightarrow 000001000, \quad 4 \rightarrow 000010000, \quad 3 \rightarrow 000100000$
   $2 \rightarrow 001000000, \quad 1 \rightarrow 010000000, \quad 0 \rightarrow 100000000$

   So, instead of using the input vector of 10 integers (first 9 for the board, and the last one for the index), we will have an input vector of 36 bits (3 bits each for 9 numbers for the board, and 9 bits for the index). The output (1 for *good* move, 2 for *bad* move) can similarly be encoded using two bits.

The classification – *good* or *bad*, for a valid configuration – must be found by checking (using z3) the two conditions, stated earlier, in which a move is classified as bad.

2. Create a network and train it using this data. Select (randomly) 50–60% of the generated data for training, and use the remaining data for testing. You may manipulate the dimensions of this network to get an accuracy of at least 85% on the training data, and at least 75% on the testing data. Please keep at least 2 hidden layers, and a total of 20 hidden neurons or more. Use ReLU as the activation function, except in the last layer where you should use softmax. While encoding this for property checking, you should not encode softmax. Instead, you should simply check which output value is bigger (like we did in the primality example demonstrated in the class).

3. In the trained network, you should check the following correctness property – *Does there exist an input (a valid configuration, and an index for player 1's move) which is classified as* good *by the network, but from the configuration resulting from it, it is possible for player 2 to make one move and win?* Use z3 to produce such an input, if one exists, or to show lack of one.

4. In the trained network, also check the following property – *Does there exist an input, where the configuration has the center cell and at least one of the corner cells marked by player 1, which is classified as* good *by the network, but from the configuration resulting from it, it is possible for player 2 to make one move and win?* Once again, you should use z3 to produce such an input, if one exists, or to show lack of one.

Note that you can verify other, more involved, properties as well. For instance, you may want to know it is possible to start from a valid board where both the players have marked two cells each, pick a *good* move for player 1, and then assuming that player 2 can play anywhere you once again pick a *good* move for player 1, but it is still possible for player 2 to make one move and win. This is not a part of the assignment, but we encourage you to try such properties.

Submission You are required to submit the scripts that you have written for all the tasks mentioned above – script to generate data (it must use calls to z3 solver to decide if a move is good or bad), script that can create and train a network, and output a trained network, another script that takes this network as input, creates z3 constraints for the property and verifies the desired property. The entire thing may be in one file, or split over multiple files, whichever way you like. You must include a `README` file that tells us how to run the code and very briefly explain which modules do what part of the task.