Best policy for notakto

Best policy for the first player on 3X3 board: 6 basic mapping pairs. The number of pieces is always even since we only consider the first player.

Pair #1

Current state			Winning state		
				W	

Pair #2

Current state			Winning state		
X					W
	X				W
			W	W	

Pair #3

Current state			Winning state		
	X				
	X				
			W		W

Pair #4

Current state			Winning state		
X		X			
X	X				
				W	

Pair #5

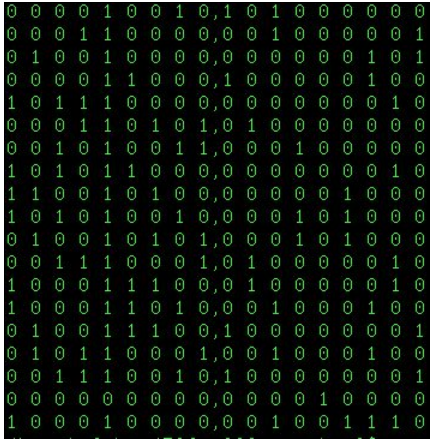
Current state			Winning state		
X		X			
	X		W		W
	X				

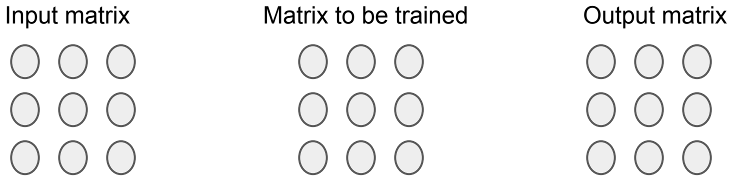
Pair #6

Current state			Winning state		
X					W
	X	X			
	X		W		

The first method I tried: One-layer network

Training data:





Training model: One layer neural network.

Conclusion: linear combination does not work for normal parameters. To check whether my network is validated, I trained the network to learn an identity matrix.

The network works for learning an identity matrix.

Input: 3*3

```
input, current state:
tensor([[[1., 0., 0.],
         [0., 1., 0.],
         [0., 0., 0.]],

        [[0., 0., 0.],
         [0., 0., 0.],
         [0., 0., 0.]],

        [[0., 0., 0.],
         [0., 0., 0.],
         [0., 0., 0.]],

        [[0., 1., 0.],
         [0., 1., 0.],
         [0., 0., 0.]],

        [[1., 0., 0.],
         [0., 1., 0.],
         [0., 0., 0.]],

        [[1., 0., 0.],
         [0., 1., 1.],
         [0., 1., 0.]]], device='cuda:0')
```

Parameters 3*3

```
current parameter 3 * 3
tensor([[ 0.9119, -0.4182,  0.0141],
        [ 0.3326,  1.0319,  0.1591],
        [ 0.1205, -0.1142,  0.8618]], device='cuda:0')
```

Output 3*3

```
tensor([[[ 0.9119, -0.4182,  0.0141],
         [ 0.3326,  1.0319,  0.1591],
         [ 0.0000,  0.0000,  0.0000]],

        [[ 0.0000,  0.0000,  0.0000],
         [ 0.0000,  0.0000,  0.0000],
         [ 0.0000,  0.0000,  0.0000]],

        [[ 0.0000,  0.0000,  0.0000],
         [ 0.0000,  0.0000,  0.0000],
         [ 0.0000,  0.0000,  0.0000]],

        [[ 0.3326,  1.0319,  0.1591],
         [ 0.3326,  1.0319,  0.1591],
         [ 0.0000,  0.0000,  0.0000]],

        [[ 0.9119, -0.4182,  0.0141],
         [ 0.3326,  1.0319,  0.1591],
         [ 0.0000,  0.0000,  0.0000]],

        [[ 0.9119, -0.4182,  0.0141],
         [ 0.4532,  0.9177,  1.0210],
         [ 0.3326,  1.0319,  0.1591]]], device='cuda:0')
```

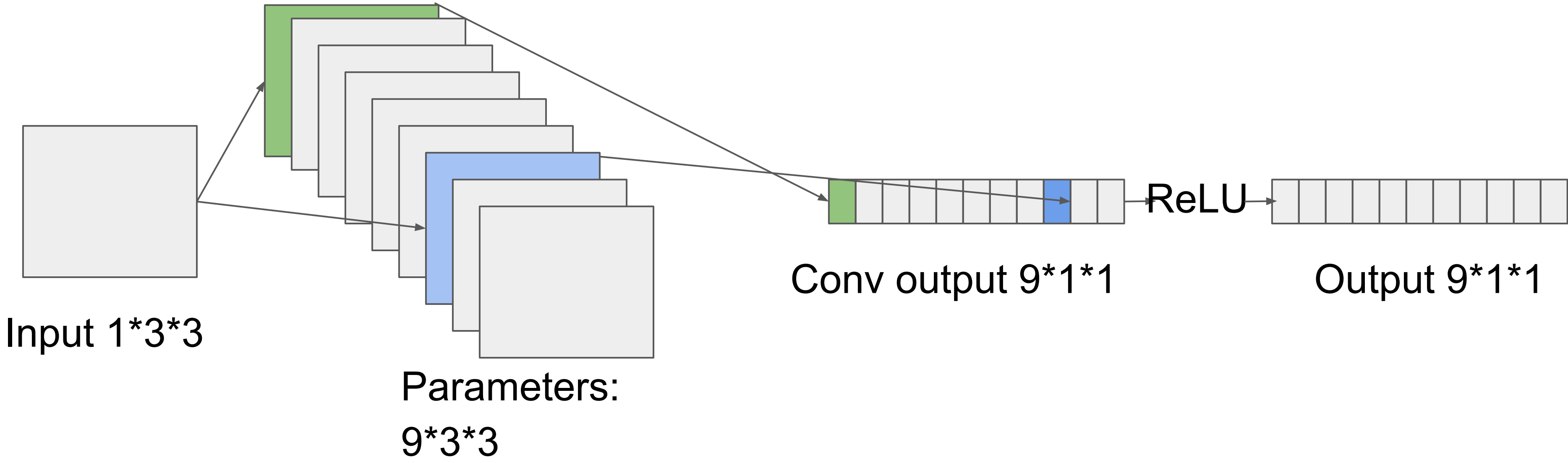
The second method I tried: Bruteforce

Try all possible mapping pairs B, check if the current state A, and the “can force a win” states C can be modeled as a matrix multiplication. Namely, C=A*B.

Conclusion: for binary parameters, such a solution does not exist.

What’s next? Multi-layer neural networks with non-linear activation functions?

The third way: Convolutional neural network



Output: 9*1*1. Each element in output corresponds to one position in the ‘can force a win’ matrix (3*3). If an element in the 9*1*1 is larger than 0.5, then it is considered as 1. Later I found that the ReLU layer can be removed.

The third way: Parameters

Can the parameters be translated into human interpretable description? Can we binalize these parameters? Can we use brute force to get the total 81 parameters? 9*2^9 = 4608.

```
tensor([[[[-1.6300, -0.3708,  0.3699],
          [-0.3699,  0.3699,  0.6301],
          [ 0.3708,  0.6301, -1.3699]]],


        [[[ 0.0334, -1.2069,  0.2219],
          [ 0.0514, -0.3086,  0.3078],
          [ 0.7169, -0.7201,  0.7870]]],


        [[[ 0.5510, -0.4490, -1.5510],
          [ 0.5510,  0.4490, -0.5510],
          [-1.4490,  0.5510,  0.4490]]],
```

```
[[[ 0.1472,  0.1150,  0.6159],
  [-1.0476, -0.2555, -0.5591],
  [ 0.2432,  0.2405,  0.6492]]]


[[[-0.0496, -0.0496, -0.0496],
  [-0.0496, -0.0496, -0.0496],
  [-0.0496, -0.0496, -0.0496]]]


[[[ 0.8869, -0.1208, -0.0968],
  [-0.9433,  0.1114, -1.3979],
  [ 0.3738, -0.1521,  0.3870]]]
```

```
[[[ 0.5001,  0.5001, -1.5000],
  [-0.4999,  0.4999,  0.5001],
  [-1.5001, -0.5000,  0.4999]]],


[[[ 0.8731, -0.9020,  0.3775],
  [-0.3685,  0.1258, -0.1242],
  [-0.1256, -1.3873,  0.6143]]],


[[[-1.3547,  0.6453,  0.3547],
  [ 0.6453,  0.3547, -0.3547],
  [ 0.3547, -0.3547, -1.6453]]]],
```