IT3105 – Module 6

Deep Learning for Game Playing

Topology

This is my initial configuration:

1 hidden layer, 40 hidden nodes, activation function=RLU, learning rate=0.1, momentum=0.9

Next, I will vary one parameter at a time and see how it affects how well the ANN plays 2048. Average ANN tile stats are going to be based on 500 runs of the 2048 game. I will use preprocessing method number 2 (more details about that later) for these stats.

Number of hidden layers

Number of hidden layers	**Average ANN tile**
1 hidden layer, 40 nodes	291
2 hidden layers, 40 nodes each	298

The difference between those two values is not very large. I'd say that the two configurations are equally good. For the sake of keeping things simple, I'm going to choose *one* hidden layer.

Number of nodes in each hidden layer

Number of hidden nodes	**Average ANN tile**
1 hidden layer, 10 nodes	292
1 hidden layer, 40 nodes	291
1 hidden layer, 160 nodes	297

Surely this proves that a really small ANN can do the job almost as well as a larger variant. I could as well have 10 hidden nodes instead of 40. While the variant with 160 nodes seems to perform ever so slightly better, I'll stick to the smaller variant with 10 hidden nodes for the sake of simplicity. I've previously (in module 5) learned that smaller networks tend to be better at generalizing what they learn.

Activation function

With the number of hidden layers and number of nodes set in stone, I'll train two networks where the only difference is the activation function:

Activation function	**Average ANN tile**
RLU	292
tanh	281

RLU seems to perform slightly better, while it's also a simpler function in terms of complexity, so I'll call out RLU as the winner of this test.

Learning rate

I tried three different learning rates: 0.02, 0.1 and 0.5. I ran the training procedure for 100 epochs with each of the three configurations. The smallest learning rate took the network a long time to converge, and would probably require much more than 100 epochs. The medium learning rate (0.1) was able to converge within 100 epochs, and so was the highest learning rate (0.5). The highest learning rate was able to achieve a slightly higher test set accuracy within 100 epochs, but the accuracy also "bounced around" a bit more after convergence. That did not happen as much with the medium learning rate, so therefore I'm going to go with the medium learning rate (0.1). I did not run any 2048-games on the ANNs because with the same number of epochs for each training session it would be unfair.

Momentum

An appropriate value for is important to avoid getting stuck in local minima, while also being able to find the global minima.

Momentum value	**Average ANN tile**
0.1	279
0.5	289
0.9	292

The actual play results indicate that the momentum matters a little. Since a momentum of 0.9 seemed to yield the best results, that's what I'm going to use from now on.

Conclusion

The variations in network topology only yield small differences in the play results. Other aspects of this problem, such as the data set and form of preprocessing, seem to be more important. Anyway, this is the ANN configuration that I ended up with:

1 hidden layer, 10 hidden nodes, activation function=RLU, learning rate=0.1, momentum=0.9

Two forms of preprocessing

Method 1: Simple

Description

The first 16 input nodes are simply the flattened values of the board, and they are divided by the max tile on the board. The next 4 nodes represent the allowed directions to move: Up, right, down, left. If a direction is allowed, the corresponding node gets a value of 1. Else the value becomes 0. This technique allows the network to learn more easily what directions are allowed, and therefore make fewer mistakes.

Results

500 runs of 2048 yielded an average max tile of 266. The 1024 tile was not achieved.

Method 2: More sophisticated

Description

Node 1-16

As in the first method, the 16 first nodes are the flattened values of the board. But there's a slight difference: this time I take the $\log_2$ of each value before normalizing against the highest tile on the board. This means that smaller tiles matter more than in method 1.

Node 17-20

Same as in method 1. These represent the allowed moves in the given board state.

Node 21

This node represents the number of empty tiles:

$$tanh(num\_empty\_tiles \ / \ 8)$$

Node 22

This node represents the sum of the tiles on the board:

$$tanh(tile\_sum \ / \ 2048)$$

Node 23-38

These nodes show which nodes are immediately mergeable. For each node of the 16 tiles:

$$value = \begin{cases} 1 \text{ if there is a neighbour with the same value} \\ 0 \text{ else} \end{cases}$$

In a way, one can say that these nodes indicate where on the board there is smoothness.

Node 39

The value of this node becomes 1 if the max tile is in a corner. Else 0.

Node 40-43

These four input nodes are probably useful. They represent the average tile position on the board, i.e. They get values based on where most of the tiles are located on the board. I could have one node for each axis (x, and y), but I decided it's better to use four nodes for representing this. One for each direction.

- Node 40: How far *east* from the center
- Node 41: How far *west* from the center
- Node 42: How far *south* from the center
- Node 43: How far *north* from the center

These nodes are "RLU-like", i.e. they are clipped at 0, so they don't get negative values. For example, if node 40 gets a value of 0.3, then node 41 gets 0, not -0.3. See the code for more details.

Results

500 runs of 2048 yielded an average max tile of 292. The 1024 tile was achieved three times. This means that the second form of preprocessing is better than the first. That's not surprising, because the extra board representation nodes make it easier for the ANN to extract significant data about each board state.

Playing style

The playing style that generated the data set

The data set is based on 100 games of 2048 played by a player that plays using a simple technique:

- Loop 1: Right, up, left, up, repeat
- Whenever down is the only option, go down, right, up and then go back to loop 1.

Because each move is determined by not only the current board state, but also the two previous moves, the ANN could not be trained to perfectly mimic the playing style of the player. If the ANN would take in board states and/or previous moves, then it would be able to better mimic the player. In other words, it would achieve a higher accuracy. My ANN currently gets an accuracy of around 75 %, which is probably close to the theoretical max given that the ANN does not know about the previous two moves. Also, one other point is that the ANN will not be smarter or better than the player that generated the data set.

The AI

The AI is pretty good at choosing legal moves. In most cases left, right or up is chosen based on the current board state. It doesn't pull down unless it has to. This is important for the "success" of the AI. The fact that the AI is playing against one side (up) makes it easily better than a random player.

An interesting fact about the playing style is that the max tile is almost never in the corner. Instead it is in one of the center tiles in the upper row. This works well for tiles up to 256 and often 512, but getting a higher tile typically requires more sophisticated technique.

Next are some screenshots from a game of 2048 played by the AI:

Board state number 20

16	16	8	2
4	4		
2			

Board state number 80

16	64	32	16
4	16	4	8
8	2	4	
4			2

Board state number 120

8	64	128	8
	32	16	4
		4	4
		4	2

Board state number 160

2	32	256	16
2	8	32	4
		2	4
			2

Board state number 240

32	64	256	32
8	32	64	8
2	8	16	2
	2	8	2

An unwise move that ended the game

At board state number 320 the AI is pretty close to achieving the 512 tile. It only has to move left, up and right. However, since the AI is not able to think ahead and see that pattern, it doesn't see that left is obviously the best move in the current situation. For some reason, it chooses right. This mistake means the difference between game over and getting the 512 tile.

Board state number 320

32	128	256	64
16	16	128	16
8	4	16	8
4	2	8	4

Board state number 321

32	128	256	64
2	32	128	16
8	4	16	8
4	2	8	4