Generating Heuristic-Driven Data for Neural Network Training (Part 1)

[](https://medium.com/@enterprisews?source=post_header_lockup)

[Robert Wang](https://medium.com/@enterprisews)

Jun 3

**Challenge of Data Acquisition**

**As**more neural network (In this article, I’ll call it “neunet” to shorten the form and to distinguish it from biological neural networks) related mathematical models have been introduced by scientists, machine learning software designers/engineers still need a significant amount of data to train neunets designed by them. It’s very common that the amount of training data directly affects the accuracy rate of a developed neunet and it has become a challenge at the initial stage of neunets development.

One way to quickly acquire such a training data set is to use a random number generator with a predefined range and format. However, this method can cause an unexpected result during the neunet training. Let’s use tic-tac-toe game as an example. I may need as many as 9! (=362,880) sets of data to train a single winning game when a random number generation method is utilized.

A more effective way to generate a required data set is to simulate a sequence of situations for the target object and automatically save the data into a repository in a specific format you want. Regardless the acquired data set is used for supervised or unsupervised neunet training, we want the artificially generated sequence of situations are close to the real one as much as possible. If it’s feasible, the classical heuristic algorithm of artificial intelligence usually best suits such a simulation. Again, I’ll use a tic-tac-toe game to demonstrate this technique for generating such a data set through out this article.

**Creating the Simulation**

Since the goal is to simulate a tic-tac-toe game playing, I need to develop a computer program that allows two players, Max “X” and Min “O”, automatically take turns during the game playing until a win or draw state has reached. In order to train the neunet as a winner, not a “drawer”, I also need to make sure the simulated games can reach a winning state as many as possible. The easiest way to design such a game playing is to arrange an experienced player again a novice player.

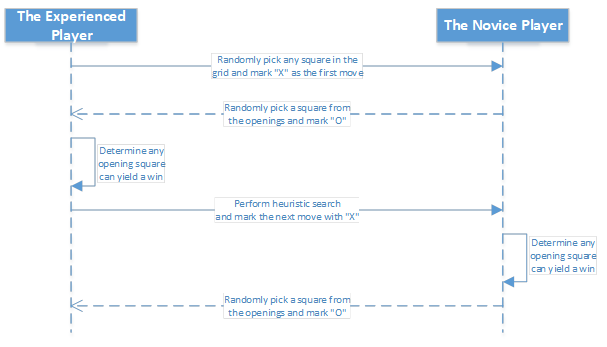
**The experienced player:**Reasoning is the earliest adopted artificial intelligence development technique that simulates how human’s thought process works to solve problems. The text book, “[Artificial Intelligence: A Modern Approach](https://www.amazon.com/Artificial-Intelligence-Approach-Stuart-Russell/dp/9332543518/ref=sr_1_1?keywords=artificial+intelligence+modern+approach&qid=1559452798&s=gateway&sr=8-1)” has listed many reasoning and searching algorithms that can be implemented to form classical artificial intelligence. I have developed a computer program in Python language to simulate the experienced player with minimax algorithm as described in the book. The source code tictactoe\_minimax.py located on [Github](https://github.com/EnterpriseWS/machine_learning_microservices/tree/master/tictactoe_minmax" \t "_blank) is an implementation of such a player.

**The novice player:** This is still a simulated player with minimal intelligence — what symbol to begin with, i.e. Max “X” or Min “O”, how to take turn and how to determine end-game state. There is no reasoning or any strategical logic implemented into this player at all. Instead, I use a random number generator to generate a number, between 1 and 9, to represent the location on the board for the next move. The class TicTacToeRandom python code in tictactoe\_gameplay.py file located on Github is to implement this player.

**Put them together:** Since the nature of the game is an interaction between two players and they have to take turns, this behavior can be implemented with typical synchronous programming. The function generate\_game\_play()coded in tictactoe\_gameplay.py file is to simulate this behavior as below.

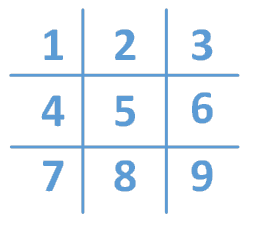
def generate\_game\_play(rounds, is\_machine\_x):  
 random\_move\_first = True  
 dbop = TicTacToeDbOp()  
 if is\_machine\_x:  
 random\_move\_first = False  
 for count in range(rounds):  
 game\_play = TicTacToeGamePlay()  
 game\_play.start\_new\_game(is\_machine\_x)  
 while not game\_play.game\_over:  
 # print(' Machine is ' + game\_play.machine\_mover + ' Game state is ' + str(game\_play.game\_state))  
 game\_play.get\_next\_move(TicTacToeRandom.get\_next\_move(random\_move\_first, game\_play.game\_state))  
 if game\_play.game\_state.get('winner') is None:  
 dbop.write(json.dumps(game\_play.game\_data), '')  
 else:  
 dbop.write(json.dumps(game\_play.game\_data), game\_play.game\_data['winner'])  
 print('Round: ' + str(count) + ' = ' + str(game\_play.game\_data))  
 # need to avoid keeping DB open when is not needed  
 dbop.close()

The diagram below illustrates what has happened within the program for some initial moves.



**Labeling the Training Data**

For the initial stage neunet training, it is important to guide the neunet with expected output, i.e. label each neuron on the output layer. This is called labeled data training. By doing so, the neunet learns to distinguish neurons and their own representation within the output layer. Since the game board is not too complex, I have labeled each cell/block on the grid with a number — 1 through 9 as displayed below.

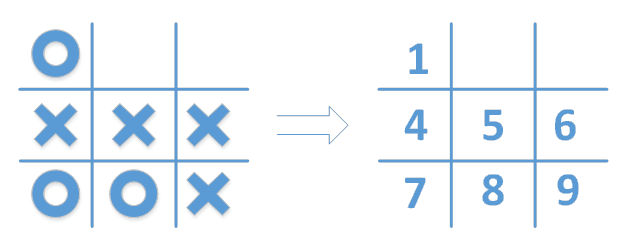


It just happens to that input and output are using the same dimension — the game board. We can use the exactly same labels for those input layer neurons as well.

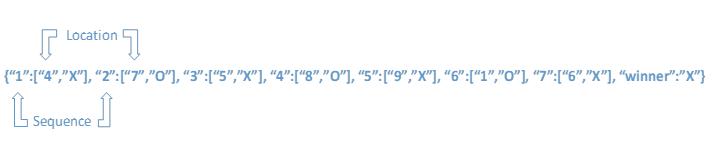
Now, let’s think what the goal is for a game play — “win the game”, right? How to reach that goal? Here are some questions we need to answer:

* ***Who* will have the first move, i.e. Max “X”, the experienced player or the novice player?**
* **At *where*, i.e. the location 1 through 9?**
* **If there is a win, *who* wins the game and at what state?**
* ***How* to make it happen, i.e. what’s the expected sequential moves in the game?**

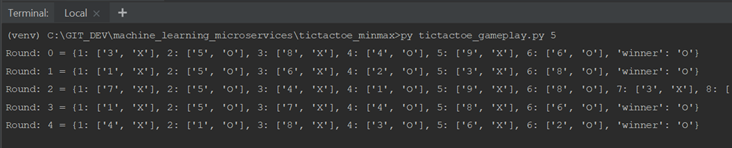
How do we capture all these information into a single data set? Here is an example of the final game state may be.



Below is the data set, in JSON format, that I have assembled to answer the above “who”, “where” and “how” questions.



In the code example, I used Sqlite to store the data set for simplicity. The Python file tictactoe\_dbop.py contains the complete database operation. You may run the command line just like below as many as you want.



**Conclusion**

I consider an artificial neural network development is quite a “brutal-force” approach. The more and the closer correlated training data sets yield the better results. I always use an analogy to describe how it works — a good neunet is just like a “muscle-memory” which needs a lot of practice to reach its optimal state. The Part 2 of this article will dig into more details about constructing such a neural network and demonstrate how the data set is used and how to determine the accuracy rate of the trained neural network.