Tic Tac Toe Playing Bot: An Introduction to Deep Q Learning.

DeepMind published their paper (Link: https://arxiv.org/pdf/1312.5602v1.pdf) which could play and beat humans at many atari games. The special thing was that the method used was not unique to any one game, i.e. it could play any atari game and learn to get good, by just taking 2 inputs, the screen frames, and the score. How? By using Deep Q.

In this blog I will attempt to use Deep Q to make a tic tac toe playing bot.

The UI of the Game

The game is built using p5.js (https://p5js.org). Which is a javascript library that brings the power of processing (https://processing.org) to the web browser.

P5 | Basics

```
function setup() {
createCanvas(window.innerWidth, window.innerHeight);
}

function draw() {
colorMode(RGB);
background(0,25); // Black Background
}

function mouseClicked() {
console.log(mouseX, mouseY);
}
```

setup(): Setup function runs once, when the webpage is opened. In this createCanvas is called, which requires width and height of the canvas (where

all the stuff will be drawn) to be created. window.innerWidth and window.innerHeight are the width and height of the browser window, this makes sure that the canvas is fullscreen and that the canvas is responsive to screen size.

draw(): this function is called again and again, it is basically called every frame and in this what needs to be drawn is defined.

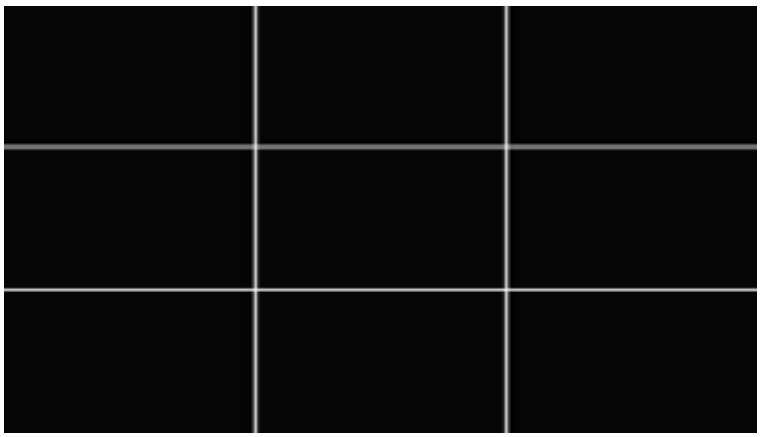
mouseClicked(): This function is called when the user clicks his/her mouse, mouseX and mouseY are the coordinates of the click.

Creating the grid

P5 provides the function line to draw lines, it takes the 4 coordinates (x1, y1, x2, y2) of the line to be drawn.

```
function drawGrid() {
    line(width/3, 0, width/3, height);
    line(2*width/3, 0, 2*width/3, height);
    line(0, height/3, width, height/3);
    line(0, 2*height/3, width, 2*height/3);
}
```

This draws the 4 lines for the grid.



So far, so good.

Creating functions to draw x and o

```
function drawO(x, y) {
    var r = (width/6)*0.8;
    var posX = (2*x + 1)*(width/6);
    var posY = (2*y + 1)*(height/6);
    var r = (width/6)*0.8;
    textSize(r);
    textAlign(CENTER);
    text("O", posX, posY+ r/2);
}
function drawX(x, y) {
    var posX = (2*x + 1)*(width/6);
    var posY = (2*y + 1)*(height/6);
    var r = (width/6)*0.8;
    textSize(r);
    textAlign(CENTER);
    text("X", posX, posY+ r/2);
}
```

The x and y sent to these functions is the x and y of the grid and not coordinates. So (0,0) means draw in top left corner of the grid.

text, textAlign and textSize are p5 functions: https://p5js.org/reference/

window.gameState = [[-1, -1, -1], [-1, -1, -1], [-1, -1, -1]];
-1 means empty cell, 0 means cell contains a ‘o’ and 1 means cell contains a ‘x’.

window.gameState basically stores the grid in matrix format.

isdone function returns whether the game is finished or not. And if it is finished it returns who won.

reset function resets the game.

Basics Of Deep Q:

Deep Q is when a neural net approximates a Q table. The Q is Q table is the final reward, given by the bell equation.

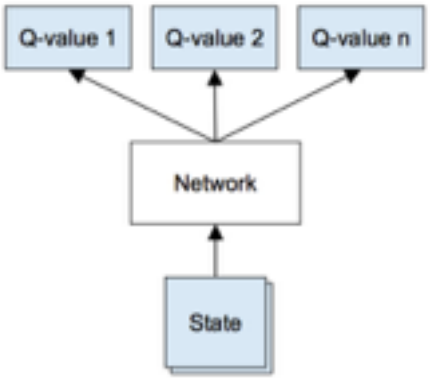
$$Q(s,a) = r + \gamma max_{a'} Q(s',a')$$

Basically rather than maximising the reward at each step, we try to maximise the final reward. Q(s,a) means the final reward (Q value) at some state and action, is equal to the immediate reward that we got + gamma * the Q value of the next state and next action.

Gamma tells us how much importance we want to give to future rewards.

So Q table has s columns and a rows (or vice versa) and each cell tells us what the final reward will be if at that state we performed that action. But the Q table quickly expands to huge size, as the task/game gets more complicated. Thats where neural network comes into play.

Using neural networks we can approximate the Q table. We basically feed the neural network the state of the system and it gives us the Q values of different actions we can perform. This architecture was proposed in the DeepMind paper mentioned above.



There are many ML libraries for javascript, for this project I used Brain.js (https://github.com/BrainJS/brain.js).

BrainJS basics

A simple neural network that approximates the xor logic is given below:

```
var net = new brain.NeuralNetwork();

net.train([{input: [0, 0], output: [0]},
          {input: [0, 1], output: [1]},
          {input: [1, 0], output: [1]},
```

```
        {input: [1, 1], output: [0]}]);

var output = net.run([1, 0]);  // [0.987]
```

By default the brainJS neural network has one hidden layer of 4 neurons, and sigmoid activation function.

The AI player: Agent Class

The agent class is the AI player. It contains the following member functions:

Remember: This function takes state, next state, action and done and stores them in memory to be used when training the network.

Replay: This function fetches random state, next state, action and done values from memory and trains the neural network.

Action: This function takes the state and returns the action to be taken, a integer between 0 and 8, representing the 9 cells of the grid.

The replay function is where all the deep Q magic happens:

```
this.replay = function(batch_size = 128) {
      batch = randomChoice(this.memory, batch_size);
      for(var i = 0; i < batch.length; i++) {
         state = batch[i][0];
         action = batch[i][1];
         next_state = batch[i][2];
         reward = batch[i][3];
         done = batch[i][4];
         finalReward = reward;
         if(done == false) {
            next_reward = argmax(this.model.run(next_state))[1];
            finalReward = reward + 0.9*next_reward;
         }
         predictions = this.model.run(state);
         predictions[action] = finalReward;
         feed_dict = {input: state, output: predictions};
         error = this.model.train(feed_dict, this.trainingPara);
         console.log("Training: ", i, error);
      }
   }
```

randomChoice takes an array and returns batch_size number of random elements from the array. Done variable contains whether this state was the last state or not, if it was the Q value is equal to the reward we got here, else it is equal to reward + gamma into the Q value of the next state (which is obtained by giving the neural network the next state and fetching the maximum reward from the response). Then the prediction's element which corresponds to the action that was taken is replaced with the calculated Q value, and the neural network is trained.

How Well Does it work?

Since the neural network is trained in the browser itself, the neural network has only one layer, so don't expect the AI to start beating you after only 2-3 games, however if you keep using the same moves to win, it will start blocking that move after couple of games.
AI blocks same move: https://imgur.com/a/xqUzN (Output GIF)

All the code for this tutorial can be found at: https://github.com/kalradivyanshu/kalradivyanshu.github.io/tree/master/ticTacToe

To see the code output: http://kalradivyanshu.github.io/ticTacToe/