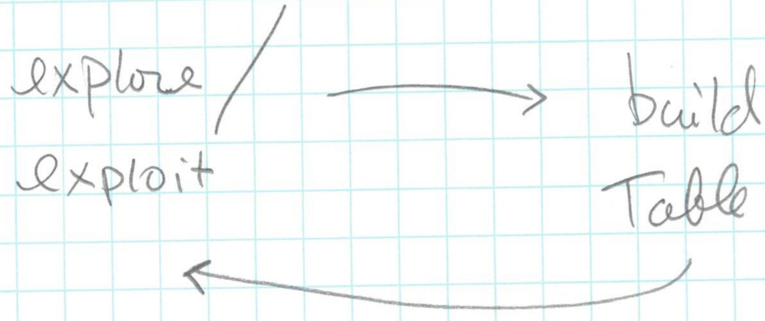
0

Q-Learning / Reinforcement notes

- Generalization using q tables and machine learning



Regular - managable state space

- here we try to build the table completely, pretty much

3

- With careful exploration, it seems like we could cover all states
    - this is possible because the states are of a manaable finite #
    - We could predict total number of states ahead of time
        - use our random starting place and work back to goal in the exploration.
        - We could identify non visited states and start there
        - We could intentionally take less than optimall paths

* Once table is built, how do we generalize our knowledge?

1

0 1 2 3 4 5 6 7 8 9 10 1 2 3

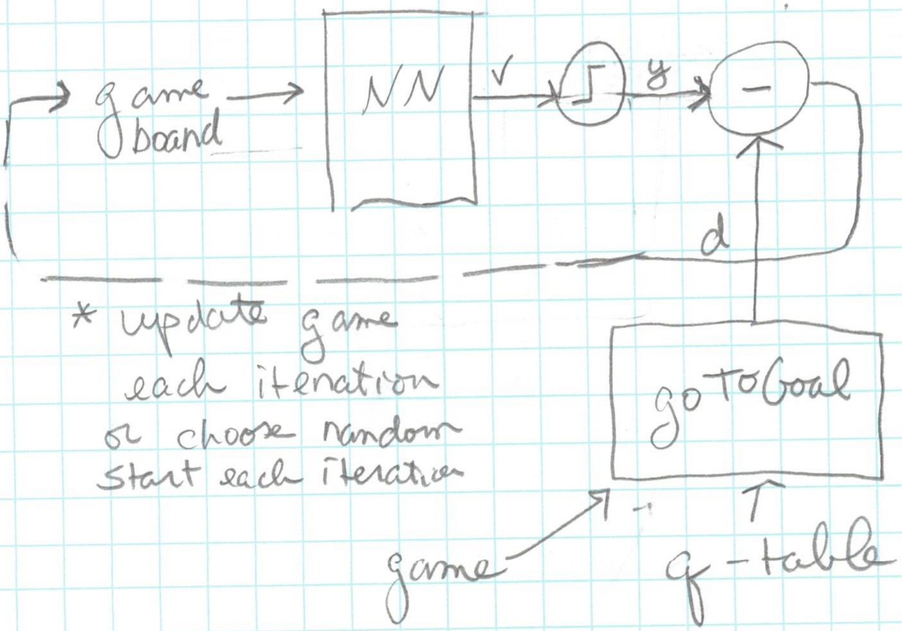
O = = = P = = = = = C =

A table for this board configuration will not work for this board configuration

= C = = = = = P = = = O

It would cause a loss every time.

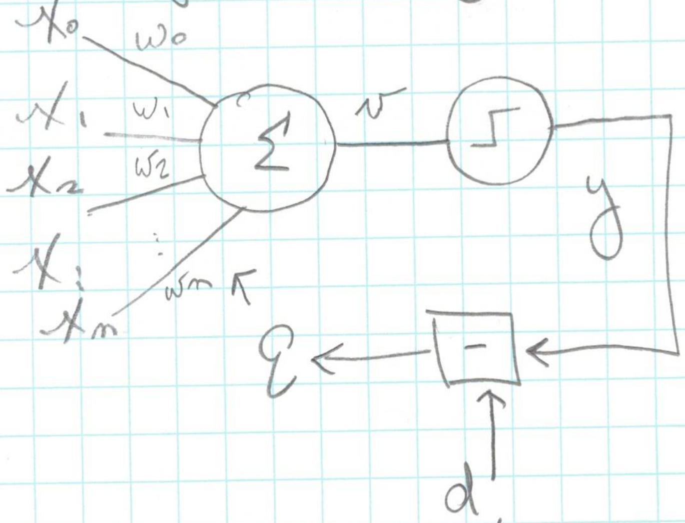
Two approaches - if 'P' has more info maybe or if we train a neural network from the tables / say we use 6 tables and then see if it generalizes


game board
NN
v
y
−
d
goToGoal
game
q-table
* update game each iteration or choose random start each iteration

input - game board

output - player direction

architetures for training the NN



Here we need input / desired value pairs

The next step is to let the q Table and the goTo Goal function provide the desired value - d

- We can switch back and forth between 2 or more games and their q tables

O=P===C	cheese on right
C==P==O	cheese on left.

Neural net should generalize and always send P to cheese

③ Implementation Details

Step 1

- Develop q-table for cheese on right side of game

> play a game. During the game,

- record each game board - these will be inputs
- each action taken as guided by the q-table - these will become desired values

- Do the above for a game with cheese on the left.

Step 2

- Create an input set with the games played, that is, the recorded steps of the games played, each step has its matching desired value, as directed by the q table.

inputs = [[$input_0$] [$input_1$] ---
--- [$input_{n-1}$]]

desired = [[d_0], [d_1] -- [d_{n-1}]]

- Using neurolab, create a perceptron and train it with the inputs and desired values
- use net.sim to test the network