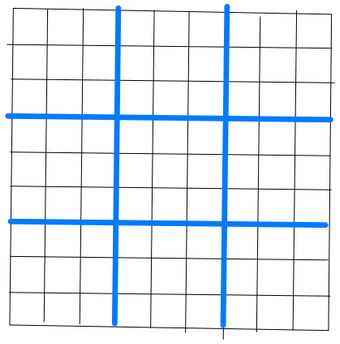
Prog2 UTTT-AI



3x3 Boards
with each 9 fields

↓↓

81 fields in total

Approach:

Input Nodes Design

- 81 input nodes representing game for symbol cross
- 81 input nodes representing game for symbol circle
- 1 input node containing the next BoardIndex to tell wether there is a specific board to be played on

162+1 Nodes

Hidden Layers Design

- Coming Soon

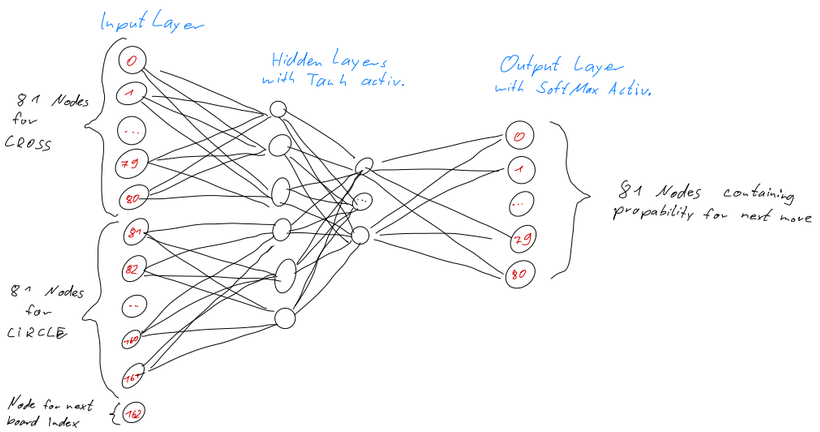
1 Layer

Output Nodes Design

- 81 output nodes since there may be 81 possibilities to place a symbol
- each node contains propability for placing at specific field
- Use softmax to make sure propabilities add up to 1

81 Nodes

Design



Java Implementation

- Matrix
  - create matrix (rows, cols)
    populate with random value
    random()*2-1;
    data stored in 2d Double array
  - elementwise addition using a value or a matrix
  - elementwise subtraction using a value
  - switch rows and columns of matrix
  - matrix multiplication of 2 matrices
  - elementwise multiplication with matrix
  - elementwise multiplication with value
  - activation functions and its derivative
  - helper function to convert matrix to game state
  - helper function to convert game state to matrix

- Neural Network
  - Variables
    weight for input layer and hidden layer
    weight for output layer and hidden layer
    bias for hidden layer
    bias for output layer
    learning rate
  - create Neural Network
    create weight matrix for input/hidden layer by number of hidden layer nodes and input nodes
    create weight matrix for output/hidden layer by number of output nodes and hidden layer nodes
    create bias matrix for hidden layer nodes and amount of layers
    create bias matrix for output layer nodes and amount of output layers
  - predict next move (forward propagation)
    get current game state and convert to matrix
    multiply hidden layer matrix weights with input
    add bias of hidden layers
    apply activation function
    multiply output layer weigh with hidden layer matrix
    add bias of output layer
    apply activation function (softmax)
    return game state
  - train (current gamestate, target Gamestate)
    get current gamestate as matrix
    create hidden matrix by multiplying
    add bias to hidden matrix
    apply activation function
    create output matrix by multiplying
    add bias to output matrix
    apply activation function softmax
    grab target gamestate as matrix
    create error matrix by subtracting (target, output)
    create gradient by applying derivative activation function to output
    multiply gradient with error
    multiply gradient with learn rate
    switch rows and columns for hidden matrix to store in new matrix
    calculate weight hidden output delta by multiplying gradient with switched matrix
    add hidden output delta to weights hidden output
    add gradient to bias output
    switch rows and cols of weight hidden output to store in new matrix
    calculate hidden errors matrix by multiplying switched matrix and error
    calculate hidden gradient by applying derivative activation function to hidden layer
    multiply with hidden errors
    multiply with learning rate
    switch rows and cols of input nodes
    calculate weights hidden input delta by multiplying hidden gradient with switched matrix
    add weight input hidden delta to input hidden weights
    add hidden gradient to bias hidden
  - train model on larger data set by doing x iterations
    iterate x times
    get random value between 0 and length of input data set
    train with input data at random value and expected data at random value

References:

https://towardsdatascience.com/understanding-and-implementing-neural-networks-in-java-from-scratch-61421bb6352c
https://www.v7labs.com/blog/neural-networks-activation-functions
https://eli.thegreenplace.net/2016/the-softmax-function-and-its-derivative/
https://www.numpyninja.com/post/neural-network-and-its-functionality