Austin Smothers

CISP 430

Assignment 13

Still waiting to ditch these training wheels

### Part 1: Implementation

/\*

A nueral net simulation used as a string acceptor.

Dan Ross

Dec 2009, tweeked Apr 2016

"Trained" by Austin Smothers

December 2017

\*/

#include <iostream>

#include <fstream>

using namespace std;

#define N 4 // nodes per layer (columns)

#define M 4 // layers (rows)

struct node {

int threshold; // a value above which the neuron will fire

int effWeight; // a sum of inputs from previous layer

int linkWeight[N]; // strength on connections to next layer (not used on last layer)

bool fire; // if effWeight > threshold

};

node net[M][N];

void initNet(void)

{

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\* THRESHOLDS \*\*\*\*\*\*

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// \*\*\* LAYER ZERO \*\*\*

net[0][0].threshold = 10;

net[0][1].threshold = 10;

net[0][2].threshold = 10;

net[0][3].threshold = 10;

// \*\*\* LAYER ONE \*\*\*

net[1][0].threshold = 10;

net[1][1].threshold = 10;

net[1][2].threshold = 10;

net[1][3].threshold = 10;

// \*\*\* LAYER TWO \*\*\*

net[2][0].threshold = 10;

net[2][1].threshold = 10;

net[2][2].threshold = 10;

net[2][3].threshold = 10;

// \*\*\* LAYER THREE \*\*\*

net[3][0].threshold = 10;

net[3][1].threshold = 10;

net[3][2].threshold = 10;

net[3][3].threshold = 10;

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\* WEIGHTS \*\*\*\*\*\*\*\*\*

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// \*\*\* LAYER ZERO \*\*\*

net[0][0].linkWeight[0] = 1;

net[0][0].linkWeight[1] = 1;

net[0][0].linkWeight[2] = 1;

net[0][0].linkWeight[3] = 1;

net[0][1].linkWeight[0] = 1;

net[0][1].linkWeight[1] = 1;

net[0][1].linkWeight[2] = 1;

net[0][1].linkWeight[3] = 1;

net[0][2].linkWeight[0] = 1;

net[0][2].linkWeight[1] = 1;

net[0][2].linkWeight[2] = 1;

net[0][2].linkWeight[3] = 1;

net[0][3].linkWeight[0] = 1;

net[0][3].linkWeight[1] = 1;

net[0][3].linkWeight[2] = 1;

net[0][3].linkWeight[3] = 1;

// \*\*\* LAYER ONE \*\*\*

net[1][0].linkWeight[0] = 1;

net[1][0].linkWeight[1] = 1;

net[1][0].linkWeight[2] = 1;

net[1][0].linkWeight[3] = 1;

net[1][1].linkWeight[0] = 1;

net[1][1].linkWeight[1] = 1;

net[1][1].linkWeight[2] = 1;

net[1][1].linkWeight[3] = 1;

net[1][2].linkWeight[0] = 1;

net[1][2].linkWeight[1] = 1;

net[1][2].linkWeight[2] = 1;

net[1][2].linkWeight[3] = 1;

net[1][3].linkWeight[0] = 1;

net[1][3].linkWeight[1] = 1;

net[1][3].linkWeight[2] = 1;

net[1][3].linkWeight[3] = 1;

// \*\*\* LAYER TWO \*\*\*

net[2][0].linkWeight[0] = 1;

net[2][0].linkWeight[1] = 1;

net[2][0].linkWeight[2] = 1;

net[2][0].linkWeight[3] = 1;

net[2][1].linkWeight[0] = 1;

net[2][1].linkWeight[1] = 1;

net[2][1].linkWeight[2] = 1;

net[2][1].linkWeight[3] = 1;

net[2][2].linkWeight[0] = 1;

net[2][2].linkWeight[1] = 1;

net[2][2].linkWeight[2] = 1;

net[2][2].linkWeight[3] = 1;

net[2][3].linkWeight[0] = 1;

net[2][3].linkWeight[1] = 1;

net[2][3].linkWeight[2] = 1;

net[2][3].linkWeight[3] = 1;

}

void trainNet(void)

{

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\* THRESHOLDS \*\*\*\*\*\*

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// \*\*\* LAYER ZERO \*\*\*

net[0][0].threshold = 15;

net[0][1].threshold = 14;

net[0][2].threshold = 12;

net[0][3].threshold = 15;

// \*\*\* LAYER ONE \*\*\*

net[1][0].threshold = 10;

net[1][1].threshold = 10;

net[1][2].threshold = 5;

net[1][3].threshold = 25;

// \*\*\* LAYER TWO \*\*\*

net[2][0].threshold = 10;

net[2][1].threshold = 11;

net[2][2].threshold = 10;

net[2][3].threshold = 10;

// \*\*\* LAYER THREE \*\*\*

net[3][0].threshold = 10;

net[3][1].threshold = 10;

net[3][2].threshold = 10;

net[3][3].threshold = 10;

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\* WEIGHTS \*\*\*\*\*\*\*\*\*

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// \*\*\* LAYER ZERO \*\*\*

net[0][0].linkWeight[0] = 8;

net[0][0].linkWeight[1] = 9;

net[0][0].linkWeight[2] = -10;

net[0][0].linkWeight[3] = 0;

net[0][1].linkWeight[0] = 10;

net[0][1].linkWeight[1] = 10;

net[0][1].linkWeight[2] = 0;

net[0][1].linkWeight[3] = 10;

net[0][2].linkWeight[0] = 11;

net[0][2].linkWeight[1] = 0;

net[0][2].linkWeight[2] = 16;

net[0][2].linkWeight[3] = 16;

net[0][3].linkWeight[0] = -30;

net[0][3].linkWeight[1] = -30;

net[0][3].linkWeight[2] = -30;

net[0][3].linkWeight[3] = -30;

// \*\*\* LAYER ONE \*\*\*

net[1][0].linkWeight[0] = 1;

net[1][0].linkWeight[1] = 5;

net[1][0].linkWeight[2] = 3;

net[1][0].linkWeight[3] = 4;

net[1][1].linkWeight[0] = 5;

net[1][1].linkWeight[1] = 5;

net[1][1].linkWeight[2] = 7;

net[1][1].linkWeight[3] = 8;

net[1][2].linkWeight[0] = 9;

net[1][2].linkWeight[1] = 5;

net[1][2].linkWeight[2] = 9;

net[1][2].linkWeight[3] = 9;

net[1][3].linkWeight[0] = 11;

net[1][3].linkWeight[1] = 0;

net[1][3].linkWeight[2] = 0;

net[1][3].linkWeight[3] = 0;

// \*\*\* LAYER TWO \*\*\*

net[2][0].linkWeight[0] = 11;

net[2][0].linkWeight[1] = 0;

net[2][0].linkWeight[2] = 0;

net[2][0].linkWeight[3] = 0;

net[2][1].linkWeight[0] = -30;

net[2][1].linkWeight[1] = -30;

net[2][1].linkWeight[2] = -30;

net[2][1].linkWeight[3] = -30;

net[2][2].linkWeight[0] = 0;

net[2][2].linkWeight[1] = 10;

net[2][2].linkWeight[2] = 11;

net[2][2].linkWeight[3] = 12;

net[2][3].linkWeight[0] = 11;

net[2][3].linkWeight[1] = 0;

net[2][3].linkWeight[2] = 0;

net[2][3].linkWeight[3] = 0;

}

void printnet(void)

{

/\* prints for each node:

threshold

effective weight

fire

list of link weights

\*/

for (int row = 0; row < M; row++)

{

// print for each node in this layer

for (int col = 0; col < N; col++)

{

cout.width(3 \* N);

cout << net[row][col].threshold << " ";

}

cout << endl;

// print effective weight for each node in this layer

for (int col = 0; col < N; col++)

{

cout.width(3 \* N);

cout << net[row][col].effWeight << " ";

}

cout << endl;

// print fire flag for each node in this layer

for (int col = 0; col < N; col++)

{

cout.width(3 \* N);

cout << net[row][col].fire << " ";

}

cout << endl;

// print the weights for each node in this layer

if (row < N - 1)

for (int col = 0; col < N; col++)

{

for (int wt = 0; wt < N; wt++)

{

cout.width(3);

cout << net[row][col].linkWeight[wt];

}

cout << " ";

}

cout << endl;

}

}

/\*

Receives a string.

Feeds it into the net.

Processes it thru the net.

Eats a cheese sandwich.

\*/

void netIN(char \* buf, int size)

{

// feed in the ASCII value of each character

// into each of the first layer nodes

for (int col = 0; col < size; col++)

{

net[0][col].effWeight = buf[col] - 'a';

if (net[0][col].effWeight > net[0][col].threshold)

net[0][col].fire = 1;

else

net[0][col].fire = 0;

}

// crunch it thru the net

int tempSum = 0;

//PROF YOU HAD THIS AS ROW <= M?! WRITING TO A 5th ROW THAT DOESN'T EXIST?

//I HOPE THIS WAS A TEST AND I PASSED IT :P

for (int row = 1; row < M; row++)

{

/\*

generate a sum for each node on this row,

by looking at the correspoding weight

for each node on the previous row

\*/

for (int thisRowsCol = 0; thisRowsCol < N; thisRowsCol++)

{

tempSum = 0;

for (int prevRowsCol = 0; prevRowsCol < N; prevRowsCol++)

{

tempSum = tempSum + net[row - 1][prevRowsCol].fire \* net[row - 1][prevRowsCol].linkWeight[thisRowsCol];

}

net[row][thisRowsCol].effWeight = tempSum;

if (net[row][thisRowsCol].effWeight > net[row][thisRowsCol].threshold)

net[row][thisRowsCol].fire = 1;

else

net[row][thisRowsCol].fire = 0;

}

}

}

/\*

Looks at last row of threshold bits in the NN and

combines them into a 4 bit binary number.

Greater than 7 is woody, otherwise its tinny!

\*/

bool IsWoody(void)

{

int value = net[3][0].fire \* 8 + net[3][1].fire \* 4 + net[3][2].fire \* 2 + net[3][3].fire \* 1;

if (value > 7)

return true;

else

return false;

}

void main(void)

{

char word[5];

bool woody;

trainNet();

// open source file

ifstream fin("words.txt");

if (!fin) { cout << "Input file could not be opened\n"; exit(1); }

// loop through strings in file

while (1) {

fin.getline(word, 5);

// end of file

if (!fin) break;

// process each word thru the net

cout << word << endl;

netIN(word, 4);

printnet();

if (IsWoody())

cout << word << " is WOODY" << endl << endl << endl;

else

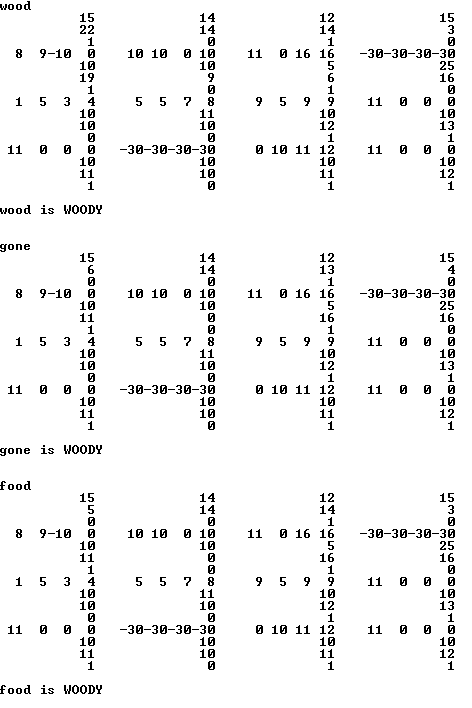
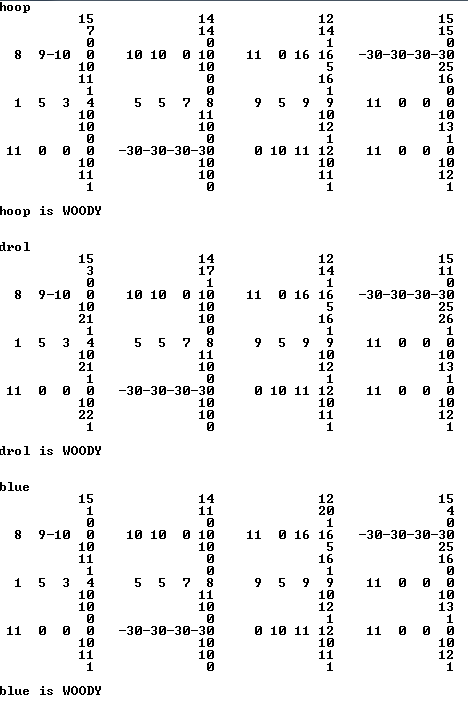
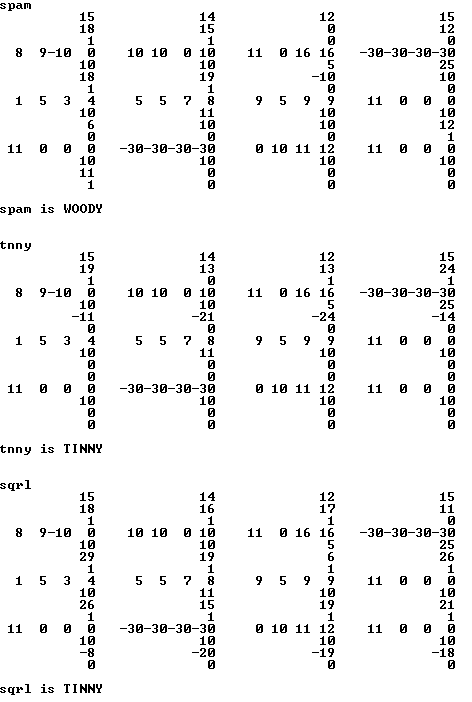
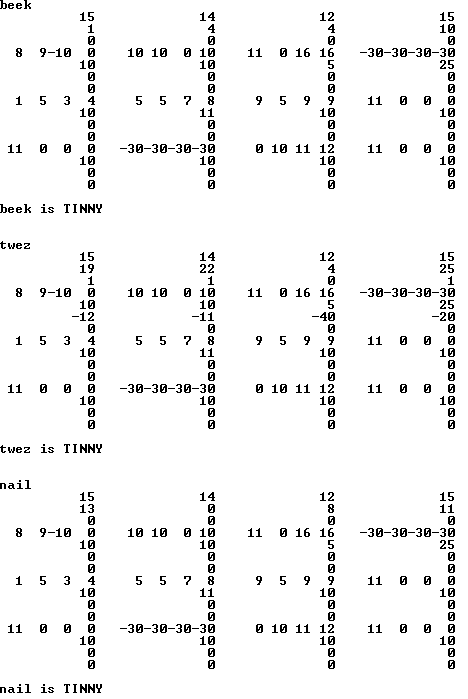
cout << word << " is TINNY" << endl << endl << endl;

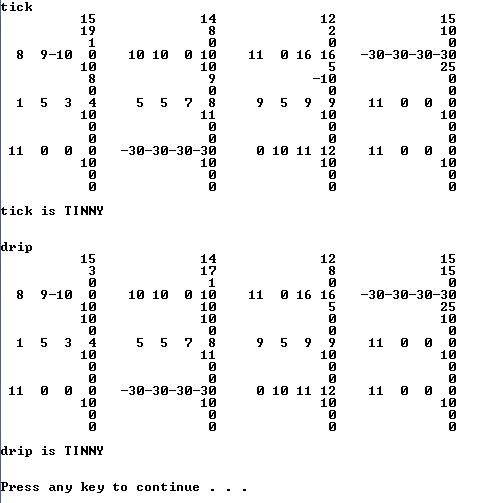
}

// close file

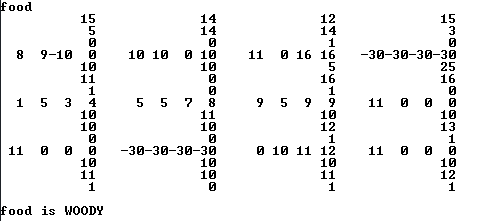
fin.close();

}





### Part 2: Food Diagram



I know this probably isn’t what you had in mind, but it is TECHNICALLY a computer drawn diagram of the neural net, and contains all of the requested elements.

### Part 3: Time/Space

The time complexity for this program, specifically the tempSum algorithm, is O(m\*n2)).

That means that, for this program, where n = 4 and m = 4, it takes roughly 64 system ticks to do the “work”.

From my knowledge of my laptop, which is a quad-core sandy-bridge i7, the processor can handle

~ 86,000 “system ticks” per second (figured that out trying to build a delay function in .586 MASM).

“Work” = ~1/86,000 = 1.1628\*10-5 [seconds]

So if I want a simple addition operation to take 1 second to complete (excluding all other code and processes normally surrounding this operation in the above program), that means the maximum value of n (assuming m = n) is ~44.

As far as space goes, I’m not gonna lie, I don’t even understand the question. My guess is that you want us to determine how many transistors we can feasibly use to calculate a NN of size NxN within a single second, and how much space such a network would take up.

My answer to that is: what year is it? Companies like Intel are constantly working to compress such technology into smaller and smaller packages due to the demand for mobile computing technology… within a year my answer will likely be obsolete and wrong.

### Part 4: How to build an Algorithm to train a neural net

The details depend entirely on the kind of neural net you are trying to train. As a general framework though, you would need to build an algorithm which accepts your initial data set, and accepts the desired “final state” of each piece of data accepted as a set of fire values. This also means you have to supply the training algorithm with the size of the neural net, and permission to edit all thresholds and linked weights. From there, the program would need a range of acceptable numeric values to test as linked weights and thresholds, and a set of initial values for each of the aforementioned to start from. The algorithm would then have to go through a process of trial and error until it arrives at a solution wherein no two data points of differing final states would have the same fire values for any given “row”. If any two such points of data did have the same final fire values, the program would have to back-track and try a new combination of thresholds and linked weights, until it arrives at a final state where no two data points of differing desired final states have the exact same fire values for any given row. The algorithm would have to repeat this process until all rows arrive at their desired final states, with the final row having the desired final state fire values either being compared to a specific set of desired fire values given by the user, or by assigning weights to each final fire value, the sum of which would be compared with a user-provided threshold.

Summatively: For any given row, the algorithm must keep the set of fire values distinct regarding variables which have different desired final states. Only variables which have the same desired final state may have the same set of fire values within a given row. The values used for linked weights and thresholds can be determined via trial and error.