

main.cpp

```
#include<iostream>
#include<vector>
#include <random>

#include <algorithm>

#include "patterns.h"

using namespace std;

void writeVector(vector<int> *v) {

    cout << "[";
    int index = 0;
    for (auto &s : *v) {
        if (index % W == 0) {
            cout << "\n[";
        }

        cout << ((s == -1) ? 0 : 1) << "\t";

        if (index % W == (W - 1)) {
            cout << "],\n";
        } else {
            cout << ',';
        }

        index++;
    }

    cout << "]\n";
}

inline int sgn(double x) {
    if (x < 0)
        return -1;
    return 1;
}

class MatchStatus {
public:
    bool isFound;
    bool isReverse;
    char pattern;
};

class Hopfield {
public:
    double w[N][N] = {0};

    void train(vector<int> *p) {
        auto pattern = *p;
        for (int i = 0; i < N; i++) {
```

```

        for (int j = 0; j < N; j++) {
            if (i == j)
                w[i][j] = 0;
            else
                w[i][j] += ((double) 1 / N) * pattern[i] * pattern[j];
        }
    }

}

vector<int> *classify(vector<int> *pattern) {
    auto *state = new vector<int>(*pattern);

    bool inSteadyState = false;

    while (!inSteadyState) {

        auto *previous_state = new vector<int>(*state);

        for (int neuron = 0; neuron < N; neuron++) {

            double res = 0;
            for (int j = 0; j < N; j++) {
                res += w[neuron][j] * (*state)[j];
            }

            (*state)[neuron] = sgn(res);
        }

        //check if we reached steady state
        if (equal(previous_state->begin(), previous_state->end(), state->begin())) {
            inSteadyState = true;
        } else {
            delete previous_state;
        }
    }

    return state;
}

};

vector<int> *matrix2vector(int matrix[H][W]) {
    auto v = new vector<int>();
    for (int i = 0; i < H; i++)
        for (int j = 0; j < W; j++)
            v->push_back(matrix[i][j]);

    return v;
}

bool match(vector<int> *first, vector<int> *second) {

```

```

    for (int i = 0; i < N; i++) {
        if ((*first)[i] != (*second)[i])
            return false;
    }
    return true;
}

void reverseVector(vector<int> *v) {
    for (int &i : *v)
        i = i == -1 ? 1 : -1;
}

MatchStatus *findMatch(vector<int> *output, vector<vector<int>*> *patterns) {
    auto result = new MatchStatus();

    result->isFound = false;

    char patternIndex = '1';
    for (const auto &p : *patterns) {
        if (match(output, p)) {
            result->isFound = true;
            result->isReverse = false;
            result->pattern = patternIndex;
            break;
        }

        reverseVector(p);
        if (match(output, p)) {
            result->isReverse = true;
            result->isFound = true;
            result->pattern = patternIndex;
            break;
        }

        patternIndex++;
    }

    return result;
}

int test_1[H][W] = {{-1, 1, 1, -1, -1, -1, -1, 1, 1, -1},
                    {-1, 1, 1, -1, -1, -1, -1, 1, 1, -1},
                    {-1, 1, 1, -1, -1, -1, -1, 1, 1, -1},
                    {-1, 1, 1, -1, -1, -1, -1, 1, 1, -1},
                    {-1, 1, 1, -1, -1, -1, -1, 1, 1, -1},
                    {-1, 1, 1, -1, -1, -1, -1, 1, 1, -1},
                    {-1, 1, 1, -1, -1, -1, -1, 1, 1, -1},
                    {-1, 1, 1, 1, 1, 1, 1, 1, 1, -1},
                    {-1, 1, 1, 1, 1, 1, 1, 1, 1, -1},
                    {-1, -1, -1, -1, -1, -1, -1, -1, 1, 1, -1},
                    {-1, -1, -1, -1, -1, -1, -1, -1, 1, 1, -1},
                    {-1, -1, -1, -1, -1, -1, -1, -1, 1, 1, -1},
                    {-1, -1, -1, -1, -1, -1, -1, -1, 1, 1, -1},
                    {-1, -1, -1, -1, -1, -1, -1, -1, 1, 1, -1},
                    {1, 1, 1, 1, 1, 1, 1, -1, -1, 1},
                    {1, 1, 1, 1, 1, 1, 1, -1, -1, 1}};

```

```

int test_2[H][W] = {{1, 1, 1, 1, 1, 1, 1, 1, 1, 1},
                    {1, 1, 1, -1, -1, -1, -1, 1, 1, 1},
                    {1, 1, -1, -1, -1, -1, -1, -1, 1, 1},
                    {1, -1, -1, -1, 1, 1, -1, -1, -1, 1},
                    {1, -1, -1, -1, 1, 1, -1, -1, -1, 1},
                    {1, -1, -1, -1, 1, 1, -1, -1, -1, 1},
                    {1, -1, -1, -1, 1, 1, -1, -1, -1, 1},
                    {1, -1, -1, -1, 1, 1, -1, -1, -1, 1},
                    {1, -1, -1, -1, 1, 1, -1, -1, -1, 1},
                    {1, -1, -1, -1, 1, 1, -1, -1, -1, 1},
                    {1, -1, -1, -1, 1, 1, -1, -1, -1, 1},
                    {1, -1, -1, -1, 1, 1, -1, -1, -1, 1},
                    {1, 1, -1, -1, -1, -1, -1, -1, 1, 1},
                    {1, 1, 1, -1, -1, -1, -1, 1, 1, 1},
                    {1, 1, 1, 1, 1, 1, 1, 1, 1, 1}};

int test_3[H][W] = {{1, -1, -1, 1, -1, -1, -1, -1, 1, 1},
                    {-1, -1, -1, -1, -1, 1, 1, 1, -1, -1},
                    {-1, -1, 1, -1, -1, -1, -1, -1, -1, -1},
                    {1, 1, -1, -1, 1, -1, -1, -1, -1, 1},
                    {1, -1, -1, 1, -1, 1, 1, -1, -1, -1},
                    {-1, 1, -1, -1, 1, -1, -1, -1, -1, -1},
                    {1, 1, -1, -1, -1, -1, -1, 1, -1, -1},
                    {1, -1, 1, -1, 1, 1, 1, 1, -1, 1},
                    {-1, -1, -1, -1, 1, -1, 1, -1, -1, 1},
                    {1, -1, 1, -1, -1, -1, 1, -1, -1, -1},
                    {-1, 1, -1, -1, -1, -1, -1, 1, -1, -1},
                    {-1, -1, -1, 1, 1, 1, -1, -1, -1, 1},
                    {-1, 1, -1, 1, -1, 1, 1, -1, -1, 1},
                    {-1, -1, -1, 1, 1, -1, -1, 1, 1, 1},
                    {1, 1, -1, 1, 1, -1, -1, -1, 1, 1},
                    {-1, 1, 1, -1, -1, -1, -1, -1, 1, -1}};

```

```

int main() {
    auto hopfield = new Hopfield();

    auto patterns = new vector<vector<int> *>();

    patterns->push_back(matrix2vector(x1));
    patterns->push_back(matrix2vector(x2));
    patterns->push_back(matrix2vector(x3));
    patterns->push_back(matrix2vector(x4));
    patterns->push_back(matrix2vector(x5));

    for (const auto &pattern : *patterns) {
        hopfield->train(pattern);
    }

    auto result = hopfield->classify(matrix2vector(test_3)); //test_1, test_2

    auto match = findMatch(result, patterns);

    writeVector(result);

    if (match->isFound) {

```

```
        cout << (match->isReverse ? '-' : ' ') << match->pattern << endl;
    } else {
        cout << "6\n";
    }

    return 0;
}
```

patterns.h

```

#ifndef PATTERNS_H
#define PATTERNS_H

#define N 160
#define W 10
#define H 16

int x1[H][W] = {
    {-1, -1, -1, -1, -1, -1, -1, -1, -1, -1},
    {-1, -1, -1, 1, 1, 1, 1, -1, -1, -1},
    {-1, -1, 1, 1, 1, 1, 1, 1, -1, -1},
    {-1, 1, 1, 1, -1, -1, 1, 1, 1, -1},
    {-1, 1, 1, 1, -1, -1, 1, 1, 1, -1},
    {-1, 1, 1, 1, -1, -1, 1, 1, 1, -1},
    {-1, 1, 1, 1, -1, -1, 1, 1, 1, -1},
    {-1, 1, 1, 1, -1, -1, 1, 1, 1, -1},
    {-1, 1, 1, 1, -1, -1, 1, 1, 1, -1},
    {-1, 1, 1, 1, -1, -1, 1, 1, 1, -1},
    {-1, 1, 1, 1, -1, -1, 1, 1, 1, -1},
    {-1, 1, 1, 1, -1, -1, 1, 1, 1, -1},
    {-1, 1, 1, 1, -1, -1, 1, 1, 1, -1},
    {-1, 1, 1, 1, -1, -1, 1, 1, 1, -1},
    {-1, -1, 1, 1, 1, 1, 1, 1, -1, -1},
    {-1, -1, -1, 1, 1, 1, 1, -1, -1, -1},
    {-1, -1, -1, -1, -1, -1, -1, -1, -1, -1}
};

int x2[H][W] = {{-1, -1, -1, 1, 1, 1, 1, -1, -1, -1},
    {-1, -1, -1, 1, 1, 1, 1, -1, -1, -1},
    {-1, -1, -1, 1, 1, 1, 1, -1, -1, -1},
    {-1, -1, -1, 1, 1, 1, 1, -1, -1, -1},
    {-1, -1, -1, 1, 1, 1, 1, -1, -1, -1},
    {-1, -1, -1, 1, 1, 1, 1, -1, -1, -1},
    {-1, -1, -1, 1, 1, 1, 1, -1, -1, -1},
    {-1, -1, -1, 1, 1, 1, 1, -1, -1, -1},
    {-1, -1, -1, 1, 1, 1, 1, -1, -1, -1},
    {-1, -1, -1, 1, 1, 1, 1, -1, -1, -1},
    {-1, -1, -1, 1, 1, 1, 1, -1, -1, -1},
    {-1, -1, -1, 1, 1, 1, 1, -1, -1, -1},
    {-1, -1, -1, 1, 1, 1, 1, -1, -1, -1},
    {-1, -1, -1, 1, 1, 1, 1, -1, -1, -1},
    {-1, -1, -1, 1, 1, 1, 1, -1, -1, -1},
    {-1, -1, -1, 1, 1, 1, 1, -1, -1, -1},
    {-1, -1, -1, 1, 1, 1, 1, -1, -1, -1}
};

int x3[H][W] = {
    {1, 1, 1, 1, 1, 1, 1, 1, -1, -1},
    {1, 1, 1, 1, 1, 1, 1, 1, -1, -1},
    {-1, -1, -1, -1, -1, 1, 1, 1, -1, -1},
    {-1, -1, -1, -1, -1, 1, 1, 1, -1, -1},
    {-1, -1, -1, -1, -1, 1, 1, 1, -1, -1},
    {-1, -1, -1, -1, -1, 1, 1, 1, -1, -1},
    {-1, -1, -1, -1, -1, 1, 1, 1, -1, -1},
    {-1, -1, -1, -1, -1, 1, 1, 1, -1, -1},
    {-1, -1, -1, -1, -1, 1, 1, 1, -1, -1},
    {-1, -1, -1, -1, -1, 1, 1, 1, -1, -1},
    {-1, -1, -1, -1, -1, 1, 1, 1, -1, -1},
    {-1, -1, -1, -1, -1, 1, 1, 1, -1, -1},
    {-1, -1, -1, -1, -1, 1, 1, 1, -1, -1},
    {-1, -1, -1, -1, -1, 1, 1, 1, -1, -1},
    {1, 1, 1, 1, 1, 1, 1, 1, -1, -1},
    {1, 1, 1, 1, 1, 1, 1, 1, -1, -1},
    {1, 1, 1, -1, -1, -1, -1, -1, -1, -1}
};

```

```
    {1, 1, 1, -1, -1, -1, -1, -1, -1, -1},
    {1, 1, 1, -1, -1, -1, -1, -1, -1, -1},
    {1, 1, 1, -1, -1, -1, -1, -1, -1, -1},
    {1, 1, 1, 1, 1, 1, 1, 1, -1, -1},
    {1, 1, 1, 1, 1, 1, 1, 1, -1, -1}};

int x4[H][W] = {{-1, -1, 1, 1, 1, 1, 1, 1, 1, -1},
                {-1, -1, 1, 1, 1, 1, 1, 1, 1, -1},
                {-1, -1, -1, -1, -1, -1, 1, 1, 1, -1},
                {-1, -1, -1, -1, -1, -1, 1, 1, 1, -1},
                {-1, -1, -1, -1, -1, -1, 1, 1, 1, -1},
                {-1, -1, -1, -1, -1, -1, 1, 1, 1, -1},
                {-1, -1, -1, -1, -1, -1, 1, 1, 1, -1},
                {-1, -1, 1, 1, 1, 1, 1, 1, -1, -1},
                {-1, -1, 1, 1, 1, 1, 1, 1, -1, -1},
                {-1, -1, -1, -1, -1, -1, 1, 1, 1, -1},
                {-1, -1, -1, -1, -1, -1, 1, 1, 1, -1},
                {-1, -1, -1, -1, -1, -1, 1, 1, 1, -1},
                {-1, -1, -1, -1, -1, -1, 1, 1, 1, -1},
                {-1, -1, -1, -1, -1, -1, 1, 1, 1, -1},
                {-1, -1, -1, -1, -1, -1, 1, 1, 1, -1},
                {-1, -1, 1, 1, 1, 1, 1, 1, 1, -1},
                {-1, -1, 1, 1, 1, 1, 1, 1, -1, -1}};

int x5[H][W] = {{-1, 1, 1, -1, -1, -1, -1, 1, 1, -1},
                {-1, 1, 1, -1, -1, -1, -1, 1, 1, -1},
                {-1, 1, 1, -1, -1, -1, -1, 1, 1, -1},
                {-1, 1, 1, -1, -1, -1, -1, 1, 1, -1},
                {-1, 1, 1, -1, -1, -1, -1, 1, 1, -1},
                {-1, 1, 1, -1, -1, -1, -1, 1, 1, -1},
                {-1, 1, 1, -1, -1, -1, -1, 1, 1, -1},
                {-1, 1, 1, 1, 1, 1, 1, 1, 1, -1},
                {-1, 1, 1, 1, 1, 1, 1, 1, 1, -1},
                {-1, -1, -1, -1, -1, -1, -1, 1, 1, -1},
                {-1, -1, -1, -1, -1, -1, -1, 1, 1, -1},
                {-1, -1, -1, -1, -1, -1, -1, 1, 1, -1},
                {-1, -1, -1, -1, -1, -1, -1, 1, 1, -1},
                {-1, -1, -1, -1, -1, -1, -1, 1, 1, -1},
                {-1, -1, -1, -1, -1, -1, -1, 1, 1, -1},
                {-1, -1, 1, 1, 1, 1, 1, 1, 1, -1},
                {-1, -1, -1, -1, -1, -1, -1, 1, 1, -1}};

#endif //PATTERNS_H
```