***Министерство образования Республики Беларусь***

***Учреждение образования***

***«Брестский государственный технический университет»***

***Кафедра ИИТ***

**Лабораторная работа №3**

**По дисциплине АОИС за III семестр**

**Тема: «Моделирование ассоциативная памяти**

**при помощи нейронных сетей»**

**Выполнил:**

Студент группы ИИ-15 (1)

2-го курса

Волк И. А.

**Проверил:**

Михно Е. В.

Брест 2018

Цель работы: Изучить обучение и функционирование релаксационных ИНС в качестве ассоциативной памяти при решении задач распознавания образов.

Задание:

1. Написать на любом ЯВУ программу моделирования ИНС для распознавания векторов согласно варианту. ИНС содержит n нейронных элементов в первом слое и m во втором слое. Если n меньше размерности вектора, тогда из вектора использовать только первые n элементов.
2. Провести исследование полученной модели. При этом на вход сети необходимо подавать искаженные образы, в которых инвертированы некоторые биты. Критерий эффективности процесса распознавания - максимальное кодовое расстояние (количество искаженных битов) между исходным и поданным образом.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Вариант | n | m | № векторов | Модель ИНС |
| 4 | 13 | 13 | 3,8,4,9 | Хопфилда\* |

\* реализовать оба режима функционирования

Код программы:

//main.cpp

#include <iostream>

#include "hopfnet.h"

constexpr int N = 13;

constexpr int SIZE\_OF\_SET = 4;

int main()

{

using namespace HopfieldNetwork;

#ifdef SYNCH

std::cout << "::Synchronous mode::" << std::endl;

#else

std::cout << "::Asynchronous mde::" << std::endl;

#endif

Sample<N> sample3 = { 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0 };

Sample<N> sample8 = { 1, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1 };

Sample<N> sample4 = { 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1 };

Sample<N> sample9 = { 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0 };

SampleSet<N, SIZE\_OF\_SET> set = { sample3, sample8, sample4, sample9 };

Net<N, SIZE\_OF\_SET> net(set);

Sample<N> test1 = { 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0 };

test1.view("Test", 2);

net.process(test1);

test1.view("Processed test");

sample3.view("Sample", 1);

std::cout << std::endl;

Sample<N> test2 = { 1, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 1 };

test2.view("Test", 2);

net.process(test2);

test2.view("Processed test");

sample8.view("Sample", 1);

std::cout << std::endl;

Sample<N> test3 = { 1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1 };

test3.view("Test", 2);

net.process(test3);

test3.view("Processed test");

sample4.view("Sample", 1);

std::cout << std::endl;

Sample<N> test4 = { 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0 };

test4.view("Test", 2);

net.process(test4);

test4.view("Processed test");

sample9.view("Sample", 1);

std::cout << std::endl;

std::cin.get();

}

//hopfnet.h

#pragma once

#include <iostream>

#include <vector>

namespace HopfieldNetwork

{

template<int N>

class Sample

{

std::vector<int> sample;

public:

Sample()

: sample(N)

{

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

sample[i] = -1;

}

Sample(int sample[N])

: sample(N)

{

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

this->sample.push\_back(sample[i]);

}

Sample(std::initializer\_list<int> list)

: sample(N)

{

int i = 0;

for (int x : list)

if(i<N)

sample[i++] = x;

for (; i < N; i++)

sample[i] = -1;

}

int& operator[](const int id)

{

try

{

if (id < 0)

throw 1;

if (id >= N)

throw 2;

return sample[id];

}

catch (int e)

{

std::cout << "int& HopfieldNetwork::Sample::operator[](const int id)\n"

<< "Error - " << e << std::endl;

system("pause");

exit(0);

}

}

bool operator==(const Sample<N>& other) const

{

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

if (sample[i] != other.sample[i])

return false;

return true;

}

Sample& operator=(const Sample<N>& other)

{

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

this->sample[i] = other.sample[i];

return (\*this);

}

int size() const

{

return N;

}

void view(const std::string& message = "\0", const int tab = 0)

{

if(message!="\0")

std::cout << message.c\_str() << ": ";

for (int i = 0; i < tab; i++)

std::cout << "\t";

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

std::cout << sample[i] << " ";

std::cout << std::endl;

}

};

template<int N, int SIZE\_OF\_SET>

class SampleSet

{

std::vector<Sample<N>> samples;

public:

SampleSet(Sample<N> samples[SIZE\_OF\_SET])

: samples(SIZE\_OF\_SET)

{

for (int i = 0; i < SIZE\_OF\_SET; i++)

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

{

this->samples[i][j] = samples[i][j];

}

}

SampleSet(std::initializer\_list<Sample<N>> list)

: samples(SIZE\_OF\_SET)

{

int i = 0;

for (Sample<N> x : list)

if (i < SIZE\_OF\_SET)

samples[i++] = x;

for (; i < SIZE\_OF\_SET; i++)

samples[i] = Sample<N>();

}

Sample<N>& operator[](const int id)

{

try

{

if (id < 0)

throw 1;

if (id >= SIZE\_OF\_SET)

throw 2;

return samples[id];

}

catch (int e)

{

std::cout << "Sample& HopfieldNetwork::SampleSet::operator[](const int id)\n"

<< "Error - " << e << std::endl;

system("pause");

exit(0);

}

}

};

template<int N, int SIZE\_OF\_SET>

class Net

{

int W[N][N];

int T[N];

public:

Net(SampleSet<N, SIZE\_OF\_SET> samples)

{

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

{

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

{

W[i][j] = 0;

for (int k = 0; k < SIZE\_OF\_SET; k++)

{

if (i == j)

{

break;

}

W[i][j] += (2 \* samples[k][i] - 1)\*(2 \* samples[k][j] - 1);

}

}

}

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

{

T[j] = 0;

for (int k = 0; k < SIZE\_OF\_SET; k++)

{

T[j] -= (2 \* samples[k][j] - 1);

}

}

}

int F(int S)

{

if (S > 0)

return 1;

else

return 0;

}

void process(Sample<N>& input)

{

int numOfIter = 0;

Sample<N> last = input;

do

{

numOfIter++;

last = input;

#ifdef SYNCH

Sample<N> copy = input;

#endif

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

{

int S = 0;

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

S += W[j][i] \* input[j];

S -= T[i];

#ifdef SYNCH

copy[i] = F(S);

#else

input[i] = F(S);

#endif

}

#ifdef SYNCH

input = copy;

#endif

} while (!(last == input));

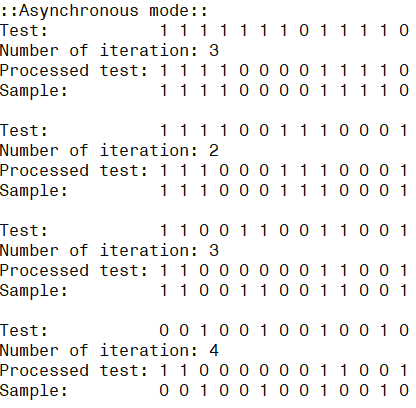
std::cout << "Number of iteration: " << numOfIter << std::endl;

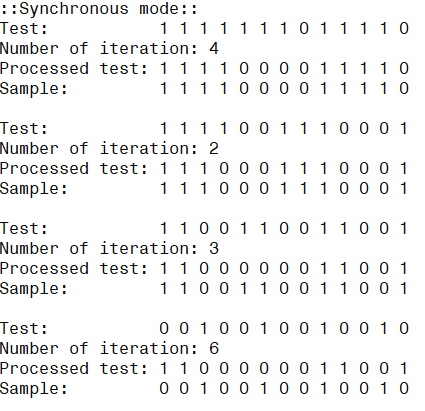
}

};

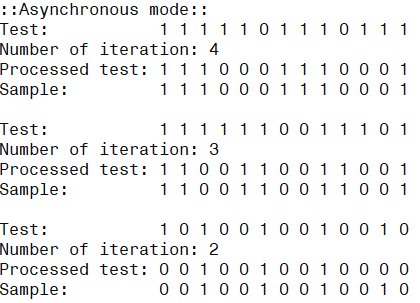
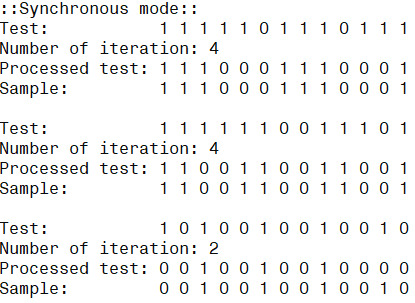
}

Результат выполнения:





При исключении первого вектора:



Вывод: изучил обучение и функционирование релаксационных ИНС в качестве ассоциативной памяти при решении задач распознавания образов. Чем меньше количество нейронов, тем меньше образов может корректно запомнить сеть Хопфилда.