Министерство образования Республики Беларусь

Учреждение образования

«Брестский Государственный технический университет»

Кафедра ИИТ

Лабораторная работа №3

По дисциплине: «Аппаратное обеспечение интеллектуальных систем»

Тема: «Моделирование ассоциативная памяти при помощи нейронных сетей»

Выполнил:

Студент 2 курса

Группы ИИ-23

Макаревич Н. Р.

Проверил:

Михно Е.В.

Брест 2024

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

Задание:

1. Изучить теоретические сведения.
2. 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.
3. 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.

Варианты заданий приведены в следующей таблице:

Вариант	n	m	№ векторов	Модель ИНС
3	12	8	2,8,3	Двунаправленная**

```
class BidirectionalNN
{
private:
	int n, m;
	std::vector<std::vector<double>> Weight;
	std::vector<std::vector<double>> WeightTransposed;

	std::vector<std::vector<double>> multiplyMatrices(const
std::vector<std::vector<double>>& matrix1, const std::vector<std::vector<double>>& matrix2);
	std::vector<std::vector<double>> transposeMatrix(std::vector<std::vector<double>>);
	std::vector<std::vector<double>> roundVector(std::vector<std::vector<double>>);
public:
	BidirectionalNN(int n, int m);
	std::vector<std::vector<double>> function(bool mode, std::vector<std::vector<double>>);
	void initializeWeight(std::vector<std::vector<double>> X, std::
vector<std::vector<double>> Y);
};
```

```
#include "BidirectionalNN.h"

BidirectionalNN::BidirectionalNN(int n, int m) {
	this->n = n;
	this->m = m;
	Weight = std::vector<std::vector<double>>(m , std::vector<double>(n, 0));
	WeightTransposed = std::vector<std::vector<double>>(n, std::vector<double>(m, 0));
}

std::vector<std::vector<double>> BidirectionalNN::roundVector(std::vector<std::vector<double>>
vec) {
	std::vector<std::vector<double>> res = vec;
	for (int i = 0; i < vec.size(); i++)
		for (int j = 0; j < vec[i].size(); j++) {
			if (res[i][j] > 1)
				res[i][j] = 1;
			if (res[i][j] < -1)
				res[i][j] = -1;
		}
	return res;
}

std::vector<std::vector<double>> BidirectionalNN::multiplyMatrices(const
std::vector<std::vector<double>>& matrix1, const std::vector<std::vector<double>>& matrix2) {
    if (matrix1.empty() || matrix2.empty() || matrix1[0].size() != matrix2.size())
        return {};

    int rows1 = matrix1.size();
    int cols1 = matrix1[0].size();
    int cols2 = matrix2[0].size();
```

```
    std::vector<std::vector<double>> result(rows1, std::vector<double>(cols2, 0));

    for (int i = 0; i < rows1; i++)
        for (int j = 0; j < cols2; j++)
            for (int k = 0; k < cols1; k++)
                result[i][j] += matrix1[i][k] * matrix2[k][j];

    return result;
}

std::vector<std::vector<double>>
BidirectionalNN::transposeMatrix(std::vector<std::vector<double>> matrix) {
      std::vector<std::vector<double>> transposedMatrix(matrix[0].size(),std::vector<double>(
matrix.size()));
      for (int i = 0; i < matrix.size(); i++)
            for (int j = 0; j < matrix[i].size(); j++)
                  transposedMatrix[j][i] = matrix[i][j];
      return transposedMatrix;
}

void BidirectionalNN::initializeWeight(std::vector<std::vector<double>> X,
std::vector<std::vector<double>> Y) {
      X = transposeMatrix(X);
      Weight = multiplyMatrices(X, Y);
      WeightTransposed = transposeMatrix(Weight);
}

std::vector<std::vector<double>> BidirectionalNN::function(bool mode,
std::vector<std::vector<double>> startVector) {
      if (mode) // true - find Y
            return roundVector(multiplyMatrices(startVector, Weight));
      else
            return roundVector(multiplyMatrices(startVector, WeightTransposed));
}
```

```
Original X vector:
1       -1      1       1       1       -1      1       1       1       -1      1       -1
Result:
1       1       1       -1      -1      -1      1       1
Corrupted with 9 bits X vector :
1       1       1       1       -1      1       1       1       -1      -1      1       1
Result:
-1      -1      -1      -1      1       1       1       1
Recognition efficiency:
5

Original X vector:
-1      -1      -1      1       1       -1      1       1       -1      -1      -1      1
Result:
1       1       1       1       -1      -1      -1      -1
Corrupted with 10 bits X vector :
-1      -1      -1      1       1       -1      1       1       1       1       1       -1
Result:
1       1       1       1       -1      -1      -1      -1
Recognition efficiency:
0

Original X vector:
1       1       -1      -1      1       1       1       1       -1      1       1       -1
Result:
1       1       1       1       -1      -1      -1      -1
Corrupted with 11 bits X vector :
-1      1       1       1       1       -1      -1      -1      -1      1       -1      -1
Result:
-1      -1      -1      -1      1       1       1       1
Recognition efficiency:
8
```

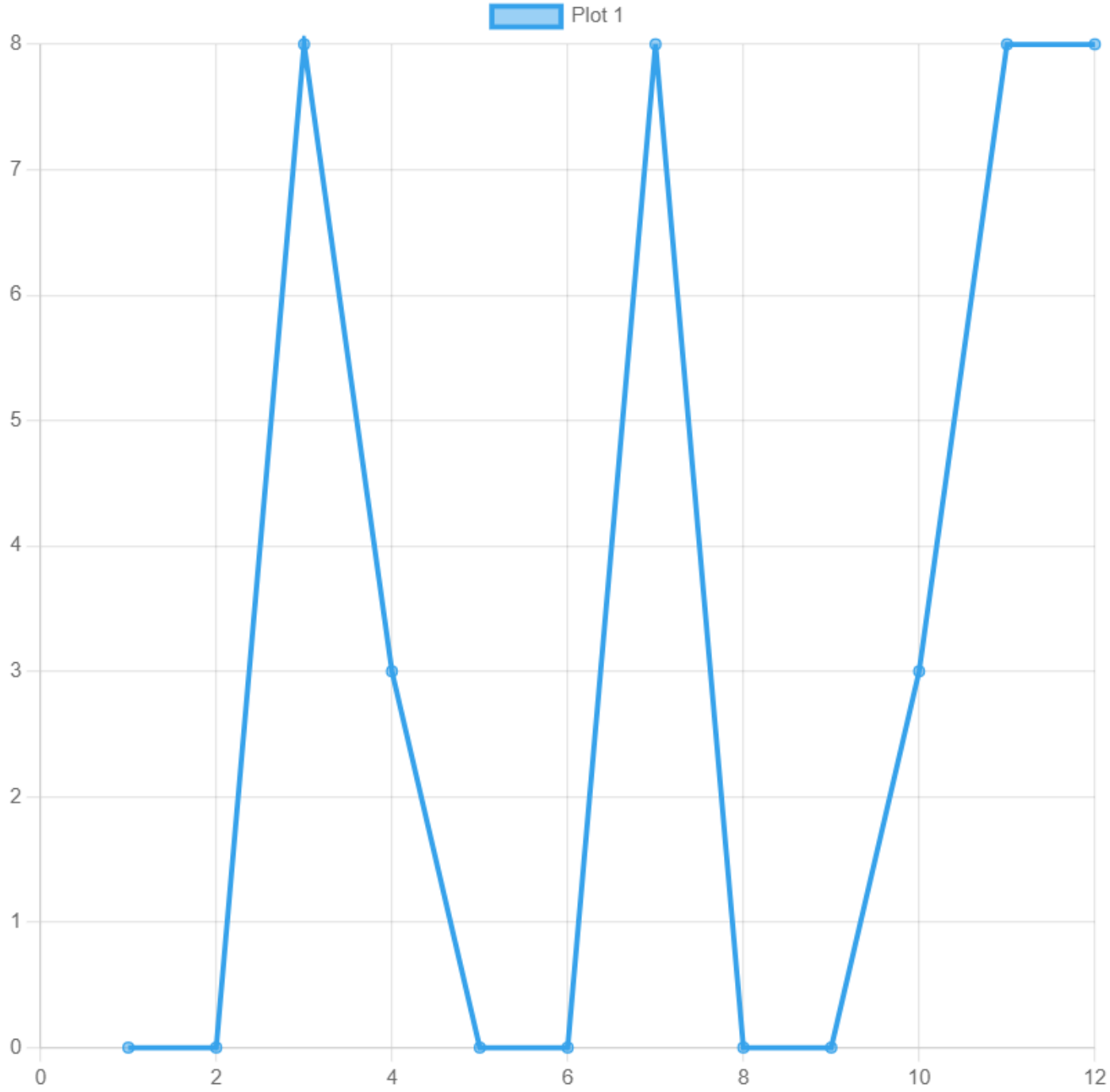

График Критерия эффективности процесса распознавания

Вывод: изучил обучение и функционирование релаксационных ИНС в качестве ассоциативной памяти при решении задач распознавания образов.