Учреждение образования

«Белорусский государственный технологический университет»

**Отчет по лабораторной работе №14**

по дисциплине: «Защита информации и надежность информационных систем»

Исполнитель:

Воликов Д. А., ФИТ 4-7

Руководитель:

Асс. Савельева М. Г.

Минск 2024

# Лабораторная работа №12. Согласование криптографических ключей на основе технологий искусственных нейронных сетей

**Цель**: изучение основ построения и функционирования искусственных нейронных сетей (ИНС), а также использования ИНС в криптографии; приобретение практических навыков программной реализации алгоритма согласования ключевой информации на основе технологии ИНС.

**Задачи**:

1. Закрепить теоретические знания по основам построения и функционирования ИНС.
2. Усвоить особенности построения, основные алгоритмы взаимного обучения двух связанных нейронных сетей на основе модели ТРМ.
3. Разработать приложение для реализации модели ТРМ с целью согласования двумя сторонами совместного тайного ключа.
4. Познакомиться с методиками оценки криптостойкости алгоритма на основе ТРМ.
5. Результаты выполнения лабораторной работы оформить в виде описания разработанного приложения, методики выполнения экспериментов с использованием приложения и результатов эксперимента.

## Теоретические сведения

Современные суперкомпьютеры превосходно справляются с задачами математических вычислений. Однако существует группа задач, решение которых не является для компьютеров простым. К основным из таких общих функциональных задач относятся: распознавание образов (букв, форм, сигналов), классификация и идентификация объектов, ассоциации.

Как известно, мозг человека состоит, главным образом, из большого числа соединенных друг с другом элементарных нервных клеток, называемых нейронами.

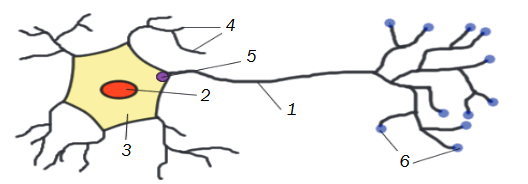


Рисунок 1 – Биологическая модель нейрона: 1 – аксон; 2 – ядро; 3 – тело клетки; 4 – дендриты; 5 – основание аксона; 6 – синапсы

На основании вышеуказанного описания (рисунок 1) спроектирован искусственный нейрон, называемый также персептроном или перцептроном, симулирующий работу биологического нейрона.

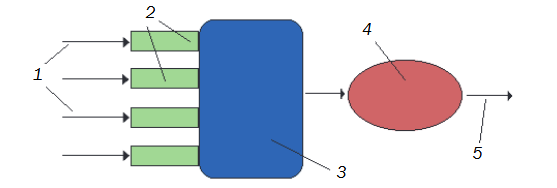


Рисунок 2 – Модель искусственного нейрона – персептрона: 1 – входы; 2 – веса; 3 – суммирующий блок; 4 – блок активации; 5 – выход

Математически работу персептрона можно описать так:

*S* = sum(*w*i ⋅ *x*i),

где *x*i и *w*i – соответственно *i*-e входной сигнал и весовой коэффициент нейрона.

В соответствии с вышеизложенным нейронная сеть (соединение отдельных нейронов), как элементарный эквивалент человеческого мозга, должна состоять из нейронов. Такую самую простую сеть с прямой связью составляет одиночный нейронный слой. В таком слое каждый нейрон получает одинаковый набор входных сигналов *X* = {*x*1, *x*2, …, *x*n}, и каждый из их имеет свой собственный вектор весов *W* = {*w*1, *w*2, …, *w*n}. Следовательно, выходной сигнал m-го нейрона может быть подсчитан следующим образом:

*S*m = sum(*w*m,i ⋅ *x*i).

Таким образом, значение выхода нейрона равно скалярному произведению входных значений на векторы весовых коэффициентов (нейрон с линейной функцией активации). Действие такой сети основывается на вычислении выходов каждого из нейронов на основе общего входного вектора *X*.

Пусть даны два персептрона, каждый из которых получает на вход случайно выбранный вектор входных значений x. Оба преобразовывают свои внутренние векторы весов в соответствии с принятым правилом обучения. При этом также учитываются собственные выходные величины *s*. Вычисляется эта величина следующим образом:

*s* = sign(*w* ⋅ *x*),

где *w* – это нормированный *N*-размерный входной вектор.

Начальное состояние векторов весов *w*A и *w*B в обеих сетях случайное. Затем на каждом шаге обучения на вход обеих сетей подается один (случайно сгенерированный) входной вектор *x*, с помощью которого обе сети вычисляют состояния своих выходов (*s*A и *s*B). На каждом шаге обучения весовые коэффициенты подвергаются следующему преобразованию, представленному на рисунке ниже.

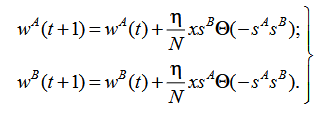


Рисунок 3 – Функции обучения персептрона для каждой сети

В соответствии с вышеуказанными формулами функция *Θ* определяется следующим образом, представленным на рисунке ниже.

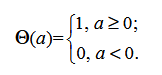


Рисунок 4 – Функция тета

Из этой формулы следует, что функция возвращает ненулевую величину для неотрицательных входных величин. Таким образом, веса активизируются только тогда, когда выходы обеих сетей противоположны.

Синхронизация дискретных персептронов. Процесс взаимного обучения, или синхронизации, начинается с инициализации весовых параметров обеих сетей подачей на вход соответствующих бинарных последовательностей. Их начальное состояние, выбираемое случайным образом, остается секретным на протяжении всего процесса обучения. Каждый последующий шаг «*t* + 1» начинается с подачи на входы обеих сетей выбранного случайным образом вектора *X*. Затем вычисляется выходная величина *О* для каждой из архитектур следующим способом, представленным на рисунке ниже.

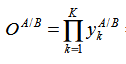


Рисунок 5 – Формула расчёта выхода сети

Индекс *A*/*B* обозначает, что данная операция касается обеих сетей *А* и *B*, а единичный индекс *А* или *B* обозначает, что операция касается соответственно только одной из сетей: *А* или *B*.

На основании обеих полученных выходных величин реализован процесс обучения. Активизация векторов весов обеих сетей происходит только тогда, когда обе выходные величины равны друг другу, т. е. *O*A = *O*B. Кроме того, внутри данной сети модифицируются веса только тех персептронов, выходная величина которых *y*k равна выходной величине *О* всей сети.

Векторы весов на каждом шаге обучения должны быть подвержены процессу нормализации. Для персептронов, у которых величины весов являются целыми числами, процесс нормализации связан с необходимостью наложения ограничений на них.

Две ИНС, обученные на основании вышеуказанной схемы, впоследствии достигают состояния синхронизации. Это означает, что их векторы весов имеют идентичные величины (*w*A = *w*B).

## Практическое задание

1. Разработать приложение, реализующее модель ТСР, – эмулятор процесса синхронизации весовых коэффициентов двух ИНС.

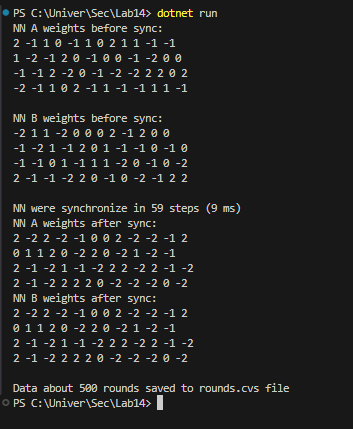


Рисунок 6 – Вывод программы синхронизации сетей

Как можно заметить, изначально веса имеют разные значения. Далее происходит процесс синхронизации. По входным данным с рисунка 6 видно, что две сети были синхронизированы за 59 шагов, а их веса имеют одинаковый вес, что и требовалось по условию задания.

1. С использованием разработанного приложения произвести не менее 500 реализаций алгоритма синхронизации сетей *А* и *В* (*N* = 10, *K* = 4, *L* = 2). Составить распределение: число синхронизаций – число шагов.

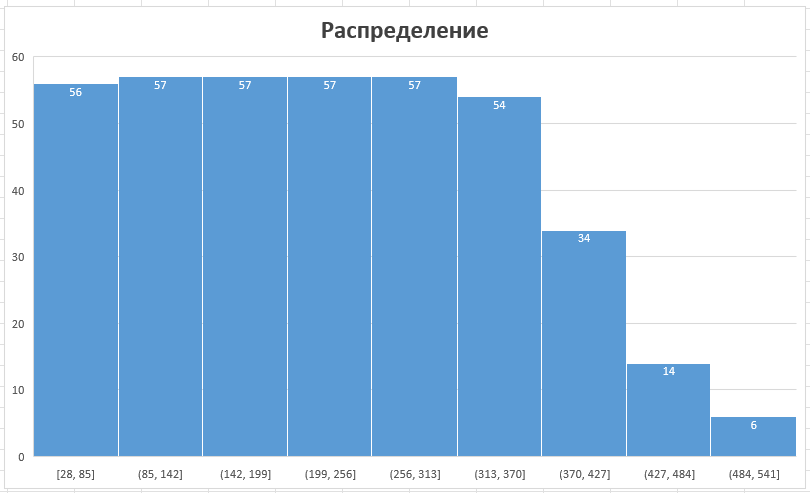


Рисунок 7 – График распределения числа шагов к количеству синхронизаций

Как можно заметить, большее количество синхронизаций наступает за количество шагов до 370, что является довольно быстро, но не самым лучшим результатом.

Вывод: в ходе этой лабораторной работы я изучил теоретические основы структуры нейронной сети и её составляющих. Изучил основные принципы подсчёта выходных значений персептронов и сетей. Разработал программу, в которой можно создать объект нейронной сети и синхронизировать его с другим объектом нейронной сети. Программа выводит время и количество шагов, за которые две нейронные сети успешно синхронизировались, а также начальные и конечные значения весов сетей.

# Листинг «Лабораторная работа №14»

using Lab14.NeuralNetwork;

namespace Lab14.Perceptron

{

public class Perceptron(PerceptronConfig config)

{

public PerceptronConfig Config { get; } = config;

public required int[] Weights { get; set; }

public int[]? Inputs { get; set; } = null;

public int Y { get; private set; }

public void CalculateOutputNetworkValue()

{

Y = Sign(Enumerable.Range(0, Weights.Length - 1).Sum(i => Weights[i] \* Inputs![i]));

}

public void Sync(int outputA, int outputB, TrainMethod method)

{

for (int i = 0; i < Weights.Length; i++)

{

Weights[i] = LimitWeight(method switch

{

TrainMethod.Heb => Weights[i] += Y \* Inputs![i],

TrainMethod.AntiHeb => Weights[i] -= Y \* Inputs![i],

TrainMethod.RandomWalking => Weights[i] += Inputs![i],

\_ => 0

});

//Weights[i] \*= Theta(Y \* outputB) \* Theta(-outputA \* outputB);

}

}

private int LimitWeight(int weight) => Math.Abs(weight) > Config.L ? Sign(weight) \* Config.L : weight;

public int GetOutput() => Sign(Enumerable.Range(0, Weights.Length - 1).Sum(i => Weights[i] \* Inputs![i]));

private static int Sign(int x) => x <= 0 ? -1 : 1;

// private static int Theta(int x) => x >= 0 ? 1 : 0;

}

}

Листинг 1 – Класс «Perceptron»

namespace Lab14.Perceptron

{

public record class PerceptronConfig

{

public required int N { get; set; }

public required int L { get; set; }

}

}

Листинг 2 – Класс «PerceptroneConfig»

namespace Lab14.NeuralNetwork

{

public class NeuralNetwork

{

public NeuralNetworkConfig Config { get; private set; }

public Perceptron.Perceptron[] Perceptrons { get; private set; }

public NeuralNetwork(NeuralNetworkConfig config, List<int[]> weights)

{

Config = config;

Perceptrons = Enumerable

.Range(0, Config.K)

.Select(p => new Perceptron.Perceptron(Config.GetPerceptronConfig())

{

Weights = weights[p]

}

)

.ToArray();

}

public int GetOutput() => Perceptrons.Select(p => p.Y).Aggregate((x, y) => x \* y);

public void Sync(int outputB)

{

Perceptrons

.Where(p => p.Y == GetOutput())

.ToList()

.ForEach(p => p.Sync(GetOutput(), outputB, Config.Method));

}

public void SetPerceptronsInputs(List<int[]> inputs)

{

for (int i = 0; i < Perceptrons.Length; i++)

{

Perceptrons[i].Inputs = inputs[i];

}

}

public List<int[]> GetWeights() => Perceptrons.Select(p => p.Weights).ToList();

}

}

Листинг 3 – Класс «NeuralNetwork»

using Lab14.Perceptron;

namespace Lab14.NeuralNetwork

{

public class NeuralNetworkConfig : IPerceptron

{

public required int N { get; set; }

public required int K { get; set; }

public required int L { get; set; }

public required TrainMethod Method { get; set; }

public PerceptronConfig GetPerceptronConfig () => new()

{

N = N,

L = L

};

}

public enum TrainMethod

{

Heb,

AntiHeb,

RandomWalking

}

}

Листинг 4 – Класс «NeuralNetworkConfig»

namespace Lab14.Perceptron

{

public interface IPerceptron

{

public int N { get; set; }

public int L { get; set; }

}

}

Листинг 5 – Интерфейс «IPerceptrone»

using Lab14.NeuralNetwork;

static List<int[]> GetRandomInputs(int N, int K)

{

return Enumerable.Range(0, N \* K)

.Select(\_ => new Random().Next(2) == 0 ? 1 : - 1)

.Select((value, index) => new { value, index })

.GroupBy(p => p.index / N)

.Select(p => p.Select(v => v.value).ToArray())

.ToList();

}

static List<int[]> GetRandomWeights(int N, int K, int L)

{

return Enumerable.Range(0, N \* K)

.Select(\_ => new Random().Next(-L, L + 1))

.Select((value, index) => new { value, index })

.GroupBy(p => p.index / N)

.Select(p => p.Select(v => v.value).ToArray())

.ToList();

}

static bool ComparePerceptrons(NeuralNetwork nn1, NeuralNetwork nn2)

{

for (int i = 0; i < nn1.Perceptrons.Length; i++)

{

for (int j = 0; j < nn1.Perceptrons[i].Weights.Length; j++)

{

if ( nn1.Perceptrons[i].Weights[j] != nn2.Perceptrons[i].Weights[j])

return false;

}

}

return true;

}

static void SaveToCvsFile(Dictionary<int, int> rounds, int count)

{

string filename = "rounds.csv";

File.WriteAllText(filename, "Step;Count\n");

foreach (var item in rounds)

{

File.AppendAllText(filename, $"{item.Key};{item.Value}\n");

}

Console.WriteLine($"Data about {count} rounds saved to rounds.cvs file");

}

var N = 12;

var K = 4;

var L = 2;

var config = new NeuralNetworkConfig()

{

N = N,

K = K,

L = L,

Method = TrainMethod.Heb

};

var weightsA = GetRandomWeights(N, K, L);

Console.WriteLine("NN A weights before sync:");

foreach (var item in weightsA)

{

foreach (var weight in item)

{

Console.Write(weight + " ");

}

Console.WriteLine();

}

var weightsB = GetRandomWeights(N, K, L);

Console.WriteLine("\nNN B weights before sync:");

foreach (var item in weightsB)

{

foreach (var weight in item)

{

Console.Write(weight + " ");

}

Console.WriteLine();

}

var networkA = new NeuralNetwork(config, weightsA);

var networkB = new NeuralNetwork(config, weightsB);

var steps = 0;

var found = false;

var timer = System.Diagnostics.Stopwatch.StartNew();

do

{

var inputs = GetRandomInputs(N, K);

networkA.SetPerceptronsInputs(inputs);

networkB.SetPerceptronsInputs(inputs);

networkA.Perceptrons.ToList().ForEach(p => p.CalculateOutputNetworkValue());

networkB.Perceptrons.ToList().ForEach(p => p.CalculateOutputNetworkValue());

var outputA = networkA.GetOutput();

var outputB = networkB.GetOutput();

if (outputA == outputB)

{

networkA.Sync(outputB);

networkB.Sync(outputA);

}

if (ComparePerceptrons(networkA, networkB))

{

found = true;

}

steps++;

} while (!found && steps < 10000);

timer.Stop();

Console.WriteLine($"\nNN were synchronize in {steps} steps ({timer.ElapsedMilliseconds} ms)");

Console.WriteLine("NN A weights after sync:");

foreach (var item in networkA.GetWeights())

{

foreach (var weight in item)

{

Console.Write(weight + " ");

}

Console.WriteLine();

}

Console.WriteLine("NN B weights after sync:");

foreach (var item in networkB.GetWeights())

{

foreach (var weight in item)

{

Console.Write(weight + " ");

}

Console.WriteLine();

}

Dictionary<int, int> rounds = [];

int count = 500;

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

{

steps = 0;

found = false;

var weightsAstep = GetRandomWeights(N, K, L);

var weightsBstep = GetRandomWeights(N, K, L);

var networkAstep = new NeuralNetwork(config, weightsAstep);

var networkBstep = new NeuralNetwork(config, weightsBstep);

do

{

var inputs = GetRandomInputs(N, K);

networkAstep.SetPerceptronsInputs(inputs);

networkBstep.SetPerceptronsInputs(inputs);

networkAstep.Perceptrons.ToList().ForEach(p => p.CalculateOutputNetworkValue());

networkBstep.Perceptrons.ToList().ForEach(p => p.CalculateOutputNetworkValue());

var outputA = networkAstep.GetOutput();

var outputB = networkBstep.GetOutput();

if (outputA == outputB)

{

networkAstep.Sync(outputB);

networkBstep.Sync(outputA);

}

if (ComparePerceptrons(networkAstep, networkBstep))

{

if (!rounds.TryAdd(steps, 1)) {

rounds[steps] += 1;

}

found = true;

}

steps++;

} while (!found && steps < 10000);

}

Console.WriteLine();

SaveToCvsFile(rounds, count);

Листинг 6 – Файл «Program.cs»