Neural Network PRNG

Раньше это называли “криворукостью”. А теперь - обфускацией /Oflameron/
Обфуска́ция (от лат. obfuscare — затенять, затемнять; и англ. obfuscate — делать неочевидным, запутанным, сбивать с толку) или запутывание кода — приведение исходного кода или исполняемого кода программы к виду, сохраняющему её функциональность, но затрудняющему анализ, понимание алгоритмов работы

SOURCE CODE
Программный генератор псевдослучайных чисел (PRNG) на основе простой нейронной сети без обучения

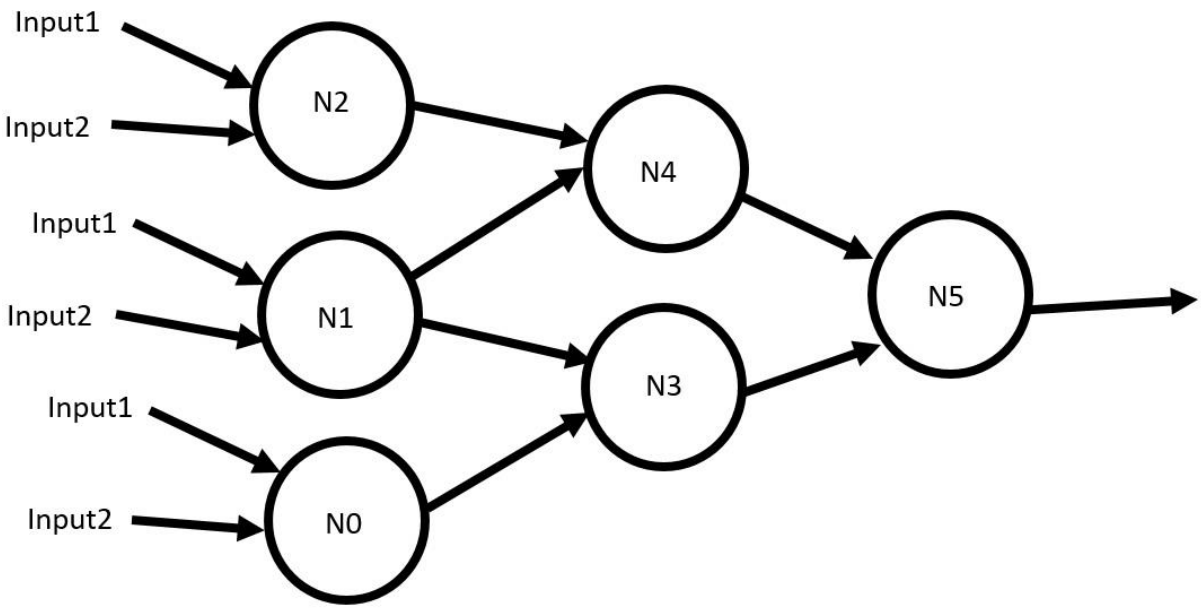
Полное описание процесса создания прототипа генератора случайных чисел на основе нейронной сети из 6 нейронов и 3 слоёв.

Используем очень простую нейронную сеть.

1.Создадим метод, с помощью которого обеспечим коррекцию ВЕСА нейрона так, чтобы значение на выходе было меньше единицы.
2. Используем один из разрядов выходного значения нейрона как одно из значений генерируемого псевдослучайного числа
3. Используем другой разряд для коррекции входных значений нейронной сети (мастер ключа) перед каждым следующим вычислении.
4.Составим из выбранных разрядов два псевдослучайных числа
5. Разделим одно число на другое и, если есть остаток от деления, возьмём бит чётности.
6. Из выбранных битов составим искомое псевдослучайное НЕОБХОДИМОГО РАЗМЕРА
7. Применение Neural PRNG в радиосвязи и радиолокации, устойчивой к РЭБ.

Приведенный здесь листинг приложения является прототипом нейронного PRNG. Но сможет (см.пункт 6) генерировать псевдослучайные числа, вполне пригодные использования в коммерческом шифровании. Потом рассмотрим способы повышения качества генерирования.

Схема нейронной сети



Android JAVA Project

Т.к. мы создаём прототип приложения, то в JAVA проекте мы не будем тратить время на пользовательский интерфейс и все результаты будем показывать в **LOG**

В **MainActivity.java** ничего нет, кроме ыфзова класса **App.java** в котором и выполняется вся обработка

```
App cls2 = new App(); // Объявляем класс с нейронной сетью
cls2.trainAndPredict(); // Вызов метода из App.java
```

Если Вы предполагаете создать пользовательский интерфейс и получить полнофункциональное приложение, то с ним можно будет работать в **MainActivity.java**

Рассмотрим **App.java**

Входные данные нейронной сети (датасет)

```
data[0][0] = 115; // input1
data[0][1] = 66; // input2

data[1][0] = 175; // input1
data[1][1] = 78; // input2

data[2][0] = 205; // input1
data[2][1] = 72; // input2

data[3][0] = 120; // input1
data[3][1] = 67; // input2
```

Здесь 4 пары входных значений. Но реально мы используем только первую пару значений, которая будет использована только на первом шаге прямого вычисления в нейронной сети.
Сразу же в каждом нейроне будут проверяться выходные значения. Если они будут 0 или 1, то будет использоваться метод коррекции весов для этого нейрона до тех пор, пока выход нейрона не станет отличаться от 0 или 1.
Остальные входные значения использовать не будем.

В отличие от классической инициализации нейронной сети, в этом проекте весовые коэффициенты зададим статически

```
neurobiasweight [0][1] = 0.2921867401328437; // Нейрон 0 вес 1
neurobiasweight [0][2] = 0.6753277674739143; // Нейрон 0 вес 2
neurobiasweight [1][1] = 0.638204802420569; // Нейрон 1 вес 1
neurobiasweight [1][2] = 0.8256511687684168; // Нейрон 1 вес 2
neurobiasweight [2][1] = 0.9570075176019762; // Нейрон 2 вес 1
neurobiasweight [2][2] = 0.39564267118987084; // Нейрон 2 вес 2
neurobiasweight [3][1] = 0.2463716717762578; // Нейрон 3 вес 1
neurobiasweight [3][2] = 0.8565685160656823; // Нейрон 3 вес 2
neurobiasweight [4][1] = 0.8529762783770741; // Нейрон 4 вес 1
neurobiasweight [4][2] = 0.7818602822337215; // Нейрон 4 вес 2
neurobiasweight [5][1] = 0.36204206369478864; // Нейрон 5 вес 1
neurobiasweight [5][2] = 0.6998272379065108; // Нейрон 5 вес 2
// В этот же массив запишем смещение bias
neurobiasweight [0][0] = 0.6372796901523922; // Нейрон 0 смещение
bias
neurobiasweight [1][0] = 0.24087856177175287; // Нейрон 1 смещение
bias
neurobiasweight [2][0] = 0.20294311087331438; // Нейрон 2 смещение
bias
neurobiasweight [3][0] = 0.6005802079109952; // Нейрон 3 смещение
bias
neurobiasweight [4][0] = 0.24210376151411686; // Нейрон 4 смещение
bias
neurobiasweight [5][0] = 0.3291532687509765; // Нейрон 5 смещение
bias
```

Значения **neurobiasweight** будут корректироваться сразу же после первого вычисления нейронной сети, если будут равны 0 или 1.

```
public double computeOOS(double input1, double input2, int nnum){ //
Обратная связь для нейрона nnum
// Если output нейрона №num равен 1 или 0, то корректировать вес1 и
вес2 этого нейрона
   while (compute(input1, input2,nnum) == 1.0) {// Вызов
стандартного COMPUTE
       app.neurobiasweight [nnum][1] -= 0.021E1; // Вес1 нейрона
№nnum
       app.neurobiasweight [nnum][2] -= 0.027E1; // Вес2 нейрона
№nnum
```

```
    }
    while (compute(input1, input2,nnum) == 0.0) {// Вызов
стандартного COMPUTE
        app.neurobiasweight [nnum][1] += 0.027; // Вес1 нейрона №nnum
        app.neurobiasweight [nnum][2] += 0.021; // Вес2 нейрона №nnum
    }
    double output = compute(input1, input2,nnum);
    return output;
} // computeOOS
```

Значения для корректировки веса не большие и выбраны экспериментально.

Запуск вычисления в нейронной сети

В `public class App` выполняем

```
Double Te = network1000.predict(data[0][0],data[0][1]);
```

В **Double Te** - результат прямого вычисления нейронной сети - выход нейрона №5

Вычисления в нейронной сети - метод **predict**

```
public Double predict(Integer input1, Integer input2){ // На вход
подавать input1 и input2
    return neurons.get(5).computeOOS( // Где neurons - это
List<Neuron> neurons = Arrays.asList(new Neuron(), new Neuron(), new
Neuron(),new Neuron(), new Neuron(),new Neuron());
            neurons.get(4).computeOOS( // Compute - начальная
активация. class Neuron
                    neurons.get(2).computeOOS(input1, input2, 2), //
compute - здесь - СИГМОИД - Util.sigmoid(preActivation)
                    neurons.get(1).computeOOS(input1, input2, 1),4
// compute - здесь - СИГМОИД - Util.sigmoid(preActivation)
            ),
            neurons.get(3).computeOOS(
                    neurons.get(1).computeOOS(input1, input2,1), //
Получить из списка neurons ПЕРВЫЙ элемент (нейрон) с входными
значениями input1, input2
                    neurons.get(0).computeOOS(input1, input2,0),3  //
compute - здесь - СИГМОИД - Util.sigmoid(preActivation)
            ),5
    );
}
```

Здесь используются входные значения и номер нейрона - **input1, input2, 0**

Вычисление выходных значений нейронов при первом прямом вычислении

```
public double compute(double input1, double input2, int nnum){ //
Вычисляем выходные данные нейрона nnum
```

```
   double preActivation = (app.neurobiasweight [nnum][1] * input1) +
(app.neurobiasweight [nnum][2] * input2) + app.neurobiasweight
[nnum][0]; // Вес 1*Вход 1 + Вес 2*Вход 2 * Смещение
   double output = Util.sigmoid(preActivation); // Диапазон значений
output –

   normalize(output); // Для организации обратной связи
   correctinput(); // Если не чётное, то вычесть из Input


   return output;
}
```

nnum - номер нейрона

После первого вычисления выходных значений нейронов, определяем, равны ли какие-либо из них 0 или 1. Если “да”, то начинаем корректировать весовые коэффициенты этого нейрона пока его выходное значение не станет больше 0 и меньше 1.
Это выполняется сразу же при первом вычислении в нейронной сети.

После этого, можно выбрать ПО ОДНОЙ ЦИФРЕ из каждого выходного значения каждого нейрона и получить 6 цифр ПСЕВДОСЛУЧАЙНОГО ЧИСЛА.

Причём здесь надо проверить, что сгенерированных выходных значениях нейронов достаточно цифр (после запятой). **ПИСАТЬ КОД ДЛЯ ПРОЕКТА** (сделаем это позже)

Тестирование прототипа с выборкой одного разряда из выходного значения каждого нейрона показало слабость такого подхода. Нейроны входного слоя дают близкие значения и в псевдослучайном числе получается много посторов цифр подряд.

==>>> >>> >>>== PRNG OUTPUT pseudornd=**8898**1291921

Можно попробовать выбирать цифры из разных позиций, но мы попробуем другое решение - будем использовать один разряд выходного значения нейронной сети после каждого вычисления.
Это работает медленнее, но генерируемое (накапливаемое) число выглядит лучше.

==>>> >>> >>>== PRNG OUTPUT pseudornd=9870945791

Листинг:

(http://templates.oflameron.ru/page00014.htm)

При попытке генерировать очень большое число (больше 100 символов) приложение виснет.

Полный JAVA код для данного этапа 23-08-2024
App.java

```
package com.example.neuroprng1;


import android.util.Log;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
import java.util.Random;

// Neural PRNG. Minimal Java Code - Minimal working Java code
// 23-08-2024 Valery Shmelev
// Java code is written for student work as a demonstration prototype
// http://templates.oflameron.ru/page00014.htm
// https://docs.google.com/document/d/1wQMjcCZXPgPtuc-s0QSABNItSGmaTdF6Py1-
mRGYcBI/edit?usp=sharing

//---------------------------------------------------------------
public class App {
    public static Double neurobiasweight [][] = new Double[6][3]; // Array of weights for neurons [weight1][weight2]
    public static Integer deltainput1 = 0; // Correcting input1 via feedback
    public static Integer deltainput2 = 0; // Correcting input2 via feedback
    public static Integer data [][] = new Integer[4][4]; // Two-dimensional array of INPUT DATA (mpster-key) -
input1 and input2
    public static Integer neuronnum = 0; // Neuron Number
    public static Double outputnnum [] = new Double[6]; // Output values of neurons
    public static String pseudornd = ""; // Here we will accumulate a pseudo-random number - String

    public static void main( String[] args ) {

    }


    public void trainAndPredict() {
        // Fill in input1 and input2 with values

        data[0][0] = 115; // input1
        data[0][1] = 66; // input2

        // Let's assign a static weight value to each neuron. To ensure repeatability of results across different
instances of the application NPRNG
        // The numbers were selected during neural network testing
        neurobiasweight [0][1] = 0.2921867401328437; // Neuron 0 weight 1
        neurobiasweight [0][2] = 0.6753277674739143; // Neuron 0 weight 2
        neurobiasweight [1][1] = 0.638204802420569; // Neuron 1 weight 1
        neurobiasweight [1][2] = 0.8256511687684168; // Neuron 1 weight 2
        neurobiasweight [2][1] = 0.9570075176019762; // Neuron 2 weight 1
        neurobiasweight [2][2] = 0.39564267118987084; // Neuron 2 weight 2
        neurobiasweight [3][1] = 0.2463716717762578; // Neuron 3 weight 1
        neurobiasweight [3][2] = 0.8565685160656823; // Neuron 3 weight 2
        neurobiasweight [4][1] = 0.8529762783770741; // Neuron 4 weight 1
        neurobiasweight [4][2] = 0.7818602822337215; // Нейрон 4 weight 2
        neurobiasweight [5][1] = 0.36204206369478864; // Neuron 5 weight 1
        neurobiasweight [5][2] = 0.6998272379065108; // Neuron 5 weight 2
        // We will write the offset into the same array
        neurobiasweight [0][0] = 0.6372796901523922; // Neuron 0 bias
        neurobiasweight [1][0] = 0.24087856177175287; // Neuron 1 bias
        neurobiasweight [2][0] = 0.20294311087331438; // Neuron 2 bias
        neurobiasweight [3][0] = 0.6005802079109952; // Neuron 3 bias
        neurobiasweight [4][0] = 0.24210376151411686; // Neuron 4 bias
```

```
        neurobiasweight [5][0] = 0.3291532687509765; // Neuron 5 bias
        // Fill the array of output values of neurons with any initial numbers
        outputnnum [0] = 0.0;
        outputnnum [1] = 0.0;
        outputnnum [2] = 0.0;
        outputnnum [3] = 0.0;
        outputnnum [4] = 0.0;
        outputnnum [5] = 0.0;

        // We create a network of 6 neurons, set the input values and write the result to the screen - the variable Te
        Network networkprng = new Network(); // We create a neural network networkprng

        Double Te = networkprng.predict(data[0][0],data[0][1]); // predict(inpur1,input2) - Return Double
        Log.i("== NNETWORK 1 Output =", " == == data[0][0] и data[0][1]   Input1=" + String.valueOf(data[0][0]) + "
и Input2=" + String.valueOf(data[0][1]));

        for (int n=0; n<10; n++) { // Some direct calculations for accumulating a pseudo-random number
            // The following direct calculation
            Te = networkprng.predict(data[0][0], data[0][1]); // predict(inpur1,input2) - Return Double
            Log.i("== NNETWORK 2 Output =", " == == data[0][0] и data[0][1]   Input1=" + String.valueOf(data[0][0]) +
"  и Input2=" + String.valueOf(data[0][1]));
            income5bigint(); // Accumulate a pseudo-random number from only the output values of the neural
network (neuron #5)
        }

        Log.i("== PSEUDORANDOM =", " ==>>> >>> >>>== PRNG OUTPUT pseudornd=" + pseudornd);

    }

    public static Integer czech(){ // Write output values of neurons to Log (for debugging)
        Log.i("== NNUM OUTPUT =", " ==>>> >>> >>>============ OUTPUT nnum0=" +
String.valueOf(outputnnum [0]));
        Log.i("== NNUM OUTPUT =", " ==>>> >>> >>>============ OUTPUT nnum1=" +
String.valueOf(outputnnum [1]));
        Log.i("== NNUM OUTPUT =", " ==>>> >>> >>>============ OUTPUT nnum2=" +
String.valueOf(outputnnum [2]));
        Log.i("== NNUM OUTPUT =", " ==>>> >>> >>>============ OUTPUT nnum3=" +
String.valueOf(outputnnum [3]));
        Log.i("== NNUM OUTPUT =", " ==>>> >>> >>>============ OUTPUT nnum4=" +
String.valueOf(outputnnum [4]));
        Log.i("== NNUM OUTPUT =", " ==>>> >>> >>>============ OUTPUT nnum5=" +
String.valueOf(outputnnum [5]));
        return (null);
    }

    public static Integer incomebigint(){ // Accumulate pseudorandom number in pseudornd

    for (int g=0; g<6; g++){ // View the entire outputnnum array
        String str = String.valueOf(outputnnum [g]); // Convert Double to String
        String substr = str.substring(5, 6); // One digit
        pseudornd += substr;
    }
        Log.i("== PSEUDORANDOM =", " ==>>> >>> >>>== PRNG OUTPUT pseudornd=" + pseudornd);

        return (null);
    }

    public static Integer income5bigint(){ // Accumulate pseudorandom number in pseudornd only from OUTPUT
neuron #5

            String str = String.valueOf(outputnnum [5]); // Convert Double to String
```

```
            String substr = str.substring(5, 6); //  One digit
            pseudornd += substr;

        return (null);
    }


} // App

//---------------------------------------------------------------------

class Network { // Called from the App class
    List<Neuron> neurons = Arrays.asList( // An extensible list of neurons from the Neuron class
            new Neuron(), new Neuron(), new Neuron(),   // Entry level
            new Neuron(), new Neuron(),                 // Hidden level
            new Neuron());                              // Output level

    //----- DIRECT (MAIN) CALCULATION -------------------------
    public Double predict(Integer input1, Integer input2){ // Input1 and input2 are used as input.
        Log.i("== NCOMPUTE =", " ==
========================================================================== == ");
        return neurons.get(5).computeOOS( // Where neurons is List<Neuron> neurons = Arrays.asList(new
Neuron(), new Neuron(), new Neuron(),new Neuron(), new Neuron(),new Neuron());
                neurons.get(4).computeOOS( // Compute - initial activation. class Neuron
                        neurons.get(2).computeOOS(input1, input2, 2), // compute - here is SIGMOID -
Util.sigmoid(preActivation)
                        neurons.get(1).computeOOS(input1, input2, 1),4  // compute - here is SIGMOID -
Util.sigmoid(preActivation)
                ),
                neurons.get(3).computeOOS(
                        neurons.get(1).computeOOS(input1, input2,1), // Get the FIRST element (neuron) from the list of
neurons with input values input1, input2
                        neurons.get(0).computeOOS(input1, input2,0),3  // compute - here is SIGMOID -
Util.sigmoid(preActivation)
                ),5
        );
    }

} // Network

//---------------------------------------------------------------------

class Neuron {

    //----- Calculate the output of neuron nnum ---------------------------
    public double compute(double input1, double input2, int nnum){ // Calculate the output of neuron nnum
        App app = new App(); // Class instance
        double preActivation = (app.neurobiasweight [nnum][1] * input1) + (app.neurobiasweight [nnum][2] * input2)
+ app.neurobiasweight [nnum][0]; // Вес 1*Вход 1 + Вес 2*Вход 2 * Смещение
        // Pass through Sigmoid
        double output = Util.sigmoid(preActivation); // Range of output values - [0,1].

        normalize(output); // To organize feedback
        correctinput(); // If not even, then subtract from Input

        return output;
    }

    // Executes before public double compute. If neuron output = 0 or 1, then adjust each weight and recalculate
    public double computeOOS(double input1, double input2, int nnum){ // Correction of weight coefficients for
neuron nnum
```

```
   // If the output of neuron nnum is 1 or 0, then adjust weight1 and weight2 of this neuron
      App app = new App(); // Class instance

      while (compute(input1, input2,nnum) == 1.0) {// Calling the standard COMPUTE
         app.neurobiasweight [nnum][1] -= 0.021E1; // Weight 1 of neuron nnum
         app.neurobiasweight [nnum][2] -= 0.027E1; // Weight 2 of neuron nnum
         //Log.i("== TRAIN =", "
==>>>========================================================================== == ");
      }

      while (compute(input1, input2,nnum) == 0.0) {// Calling the standard COMPUTE
         app.neurobiasweight [nnum][1] += 0.027E1; // Weight 1 of neuron nnum
         app.neurobiasweight [nnum][2] += 0.021E1; // Weight 2 of neuron nnum
         //Log.i("== TRAIN =", " ==>>>
>>>========================================================================== == ");
      }

      double outputoos = compute(input1, input2,nnum);

      app.outputnnum[nnum]=outputoos; // Remember the output values of a neuron nnum

      return outputoos;
   } // computeOOS

   // From the output values of neurons, we select individual bits to adjust the master key (input1 and input2)
   public Integer normalize(double neuroutput) { // Check the output value of the neuron that it is not 0 and not 1
      App app = new App(); // Class instance
      Integer result= 0;
      if (neuroutput != 0){
         if (neuroutput !=1) {
            String str = String.valueOf(neuroutput); // Convert Double to String
            // Here you need to check the length of the resulting string, otherwise these digits may not be there -
method Czech_Line
            String substr = str.substring(7, 8); // One digit
            result = Integer.valueOf(substr); // Single-Digit Integer
            app.deltainput1 = Integer.valueOf(str.substring(7, 8)); // Single-Digit Integer for correction weight1
            app.deltainput2 = Integer.valueOf(str.substring(8, 9)); // Single-Digit Integer for correction weight2
            ////Log.i("== NORMALIZE =", " == == Integer.valueOf(substr) == == substr=" + str.substring(7, 8));
            ////Log.i("== NORMALIZE =", " == == Integer.valueOf(substr) == == substr=" + str.substring(8, 9));
         }
      }
      return (null);
   }

   // Correct master key (input1 and input2)
   public Integer correctinput() { // If deltaweight is even, then with a plus, if not even, then with a minus
      App app = new App(); // Class instance
      if (app.deltainput1 % 2 != 0){ // The number is not even
         app.deltainput1 *= -1; // Take with a minus sign
       }
      if (app.deltainput2 % 2 != 0){ // The number is not even
         app.deltainput2 *= -1; // Take with a minus sign
      }
      app.data[0][0] += app.deltainput1; // Correct input1
      app.data[0][1] += app.deltainput2; // Correct input2

      return (null);
   }

} // Neuron
```

```
//----- Activation function - SIGMOID ----------------------------
class Util {
    public static double sigmoid(double in){ // Activation function - SIGMOID. Range of values [0,1]
        // Here in is preActivation = input1 * weight1 + input2 * weight2 + bias

        return 1 / (1 + Math.exp(-in));  // Sigmoid. Compresses values into the range from 0 to 1
    }

    //    (1 / (1 + Math.exp(-in)))* (1 - in)
} // Util class
```

Приведенный JAVA код не оптимален и приведен “как есть”.
Делаем далее

Нейронный PRNG

Часть 2

Модернизируем имеющийся листинг – вместо коррекции входных сигналов input1 и input2, будем каждый раз создавать их заново из нескольких разрядов выходного значения нейронной сети (нейрона №5).
Кроме того выделим исходные параметры NPRNG в класс MainActivity.java а вычисления – в класс App.java

App.java
https://drive.google.com/file/d/1tCeqStW7ypx5B0wesBYwaVn2y1IQ07NK/view?usp=sharing

MainActivity.java
https://drive.google.com/file/d/1WDq33mgUecTM5V0GXcO5NSFoQqtXTuXo/view?usp=sharing
App.java листинг

```
package com.example.neuroprng3class;


import android.util.Log;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
import java.util.Objects;
import java.util.Random;

// Сделаем Java Class для построения Нейронных PRNG
// Данный класс является простым ПРОТОТИПОМ нейронного PRNG и может быть использован в качестве одного из элементов
// композитных NPRNG, которые могут быть созданы по отдельному контракту
// (c) by Valery Shmelev

// Используется режим обучения генератора без сохранения накопленных данных.
// Можно добавить в Class возможность сохранения данных.
```

```
//---------------------------------------------------------------
public class App {
   public static String pseudornd = ""; // Здесь будем накапливать псевдослучайное число
   public static String substr = ""; // Выбранная цифра из выходного значения нейронной сети для проверки
НЕТ ЛИ ПОВТОРА
   public static String norep = ""; // Запоминаем предыдущее значение, чтобы не было повторов в
псевдослучайном числе

   public void trainAndPredict() {
      MainActivity main = new MainActivity(); // Экземпляр класса

      // Создаем сеть из 6 нейронов, задаем входные значения и пишем на экран результат - переменная
Te
      Network networkprng = new Network(); // Создаем нейронную сеть networkprng
      Double Te = networkprng.predict(main.data[0][0],main.data[0][1]); // predict(inpur1,input2) - Return Double
      //Log.i("== NNETWORK 1 Output =", " == == data[0][0] и data[0][1]   Input1=" +
String.valueOf(main.data[0][0]) + "  и Input2=" + String.valueOf(main.data[0][1]));

      for (int n=0; n<main.prngsize; n++) {
         // Следующее прямое вычисление
         Te = networkprng.predict(main.data[0][0], main.data[0][1]); // predict(inpur1,input2) - Return Double
         //Log.i("== NNETWORK 2 Output =", " == == data[0][0] и data[0][1]   Input1=" +
String.valueOf(main.data[0][0]) + "  и Input2=" + String.valueOf(main.data[0][1]));
         income5bigint(); // Накопить псевдослучайное число только из выходных значений нейронной сети
      }

      Log.i("== PSEUDORANDOM =", " ==>>> >>> >>>== NPRNG OUTPUT pseudornd=" + pseudornd);

   }

   public static Integer income5bigint(){ // Накопить псевдослучайное число в pseudornd только из
ВЫХОДНОГО нейрона №5
      MainActivity main = new MainActivity(); // Экземпляр класса
      String str = String.valueOf(main.outputnnum [5]); // Преобразовать Double в String
      substr = str.substring(5, 6); // Одна цифра

      if (!substr.equals("0")) {
         if (!Objects.equals(norep, substr)) { // Если цифра не повторяется
            //Log.i("== INCOME5BIGINT =", " ==>>> >>> >>>== norep=" + norep);
            //Log.i("== INCOME5BIGINT =", " =>>> >>> >>>== substr=" + substr);
            pseudornd += substr;
            norep = substr; // Запомнить новое значение, добавленное к псевдослучайному числу
         }
      }
      return (null);
   }

} // App

//---------------------------------------------------------------------

class Network { // Вызывается из класса App
   List<Neuron> neurons = Arrays.asList( // Расширяемый список нейронов из класса Neuron
         new Neuron(), new Neuron(), new Neuron(),   // Входной уровень
         new Neuron(), new Neuron(),                 // Скрытый уровень
         new Neuron());                              // Выходной уровень

   //----- ПРЯМОЕ (ОСНОВНОЕ) ВЫЧИСЛЕНИЕ -------------------------
   public Double predict(Integer input1, Integer input2){ // На вход подавать input1 и input2
```

```
        Log.i("== NCOMPUTE =", " ==
========================================================================== == ");
        return neurons.get(5).computeOOS( // Где neurons - это List<Neuron> neurons = Arrays.asList(new
Neuron(), new Neuron(), new Neuron(),new Neuron(), new Neuron(),new Neuron());
                neurons.get(4).computeOOS( // Compute - начальная активация. class Neuron
                        neurons.get(2).computeOOS(input1, input2, 2), // compute - здесь - СИГМОИД -
Util.sigmoid(preActivation)
                        neurons.get(1).computeOOS(input1, input2, 1),4  // compute - здесь - СИГМОИД -
Util.sigmoid(preActivation)
                ),
                neurons.get(3).computeOOS(
                        neurons.get(1).computeOOS(input1, input2,1), // Получить из списка neurons ПЕРВЫЙ элемент
(нейрон) с входными значениями input1, input2
                        neurons.get(0).computeOOS(input1, input2,0),3  // compute - здесь - СИГМОИД -
Util.sigmoid(preActivation)
                ),5
        );
    }

} // Network

//----------------------------------------------------------------------

class Neuron { // Нейрон

    //----- Вычисляем выходные данные нейрона nnum -------------------------
    public double compute(double input1, double input2, int nnum){ // Вычисляем выходные данные нейрона
nnum
        MainActivity main = new MainActivity(); // Экземпляр класса
        double preActivation = (main.neurobiasweight [nnum][1] * input1) + (main.neurobiasweight [nnum][2] *
input2) + main.neurobiasweight [nnum][0]; // Вес 1*Вход 1 + Вес 2*Вход 2 * Смещение
        // Пропускаем через Сигмоид
        double output = Util.sigmoid(preActivation); // Диапазон значений output - [0,1].

        normalize(output); // Для организации обратной связи

        return output;
    }

    // Выполняется перед  public double compute. Если выход нейрона = 0 или 1, то корректировать каждый
вес и пересчитывать
    public double computeOOS(double input1, double input2, int nnum){ // Коррекция весовых коэффициентов
для нейрона nnum
        // Если output нейрона №num равен 1 или 0, то корректировать вес1 и вес2 этого нейрона
        MainActivity main = new MainActivity(); // Экземпляр класса

        while (compute(input1, input2,nnum) == 1.0) {// Вызов стандартного COMPUTE
            main.neurobiasweight [nnum][1] -= 0.021E1; // Вес1 нейрона №nnum
            main.neurobiasweight [nnum][2] -= 0.029E1; // Вес2 нейрона №nnum
        }

        while (compute(input1, input2,nnum) == 0.0) {// Вызов стандартного COMPUTE
            main.neurobiasweight [nnum][1] += 0.015E1; // Вес1 нейрона №nnum
            main.neurobiasweight [nnum][2] += 0.011E1; // Вес2 нейрона №nnum
        }

        double outputoos = compute(input1, input2,nnum);

        main.outputnnum[nnum]=outputoos; // Запомнить выходные значения нейрона №nnum

        return outputoos;
```

```
    } // computeOOS

    // Из выходных значений нейронов мы выбираем отдельные разряды для корректировки мастер-ключа
(input1  и input2)
    public Integer normalize(double neuroutput) { // Проверить выходное значение нейрона, что оно не 0 и не 1
        MainActivity main = new MainActivity(); // Экземпляр класса
        Integer result= 0;
        if (neuroutput != 0){
            if (neuroutput !=1) {
                String str = String.valueOf(neuroutput); // Преобразовать Double в String
                //-----------------------------------------------------------------
                // Возьмем input1 и input2 из выхода нейронной сети
                main.data[0][0] = Integer.valueOf(str.substring(4, 6)); // Корректировать input1
                main.data[0][1] = Integer.valueOf(str.substring(6, 8)); // Корректировать input2

            }
        }
        return (null);
    }

} // Neuron


//----- Функция активации - СИГМОИД ----------------------------
class Util {
    public static double sigmoid(double in){ // Функция активации - СИГМОИД. Диапазон значений [0,1]
        // Здесь in - это preActivation = вход1 * вес1 + вход2 * вес2 + смещение

        return 1 / (1 + Math.exp(-in));  // Сигмоид. Сжимает значения в диапазон от 0 до 1
    }

    //    (1 / (1 + Math.exp(-in)))* (1 - in)
} // Util class
```

MainActivity.java листинг

```
package com.example.neuroprng3class;
import androidx.appcompat.app.AppCompatActivity;

import android.os.Bundle;
import android.widget.EditText;

// Вызов класса App.java для вычисления псевдослучайного числа и начальные параметры для работы
(мастер-ключ)

public class MainActivity extends AppCompatActivity {
    public static Integer data [][] = new Integer[4][4]; // Двумерный массив ВХОДНЫХ ДАННЫХ (мпстер-ключ)
По 4 значения input1 и input2
    public static Double neurobiasweight [][] = new Double[6][3]; // Массив весов для нейронов
[weight1][weight2]
    public static Double outputnnum [] = new Double[6]; // Выходные значения нейронов
    public static Integer prngsize = 0; // Размер необходимого псевдослучайного числа

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        //-----------------------------------------------------
```

```
        prngsize = 280; // Сгенерировать 280 разрядное число
        //---------------------------------------------------
        // Для работы NPRNG нужны начальные значения
        data[0][0] = 1155; // input1
        data[0][1] = 663; // input2

        // Следующие данные могут быть использованы в качестве мастер-ключа
        // Например: 2921867401328437 6753277674739143 ... 6998272379065108
        // Зададим статические значение веса для каждого нейрона. Для повторяемости результатов в
разных экземплярах приложения
        neurobiasweight [0][1] = 0.2921867401328437; // Нейрон 0 вес 1
        neurobiasweight [0][2] = 0.6753277674739143; // Нейрон 0 вес 2
        neurobiasweight [1][1] = 0.638204802420569; // Нейрон 1 вес 1
        neurobiasweight [1][2] = 0.8256511687684168; // Нейрон 1 вес 2
        neurobiasweight [2][1] = 0.9570075176019762; // Нейрон 2 вес 1
        neurobiasweight [2][2] = 0.39564267118987084; // Нейрон 2 вес 2
        neurobiasweight [3][1] = 0.2463716717762578; // Нейрон 3 вес 1
        neurobiasweight [3][2] = 0.8565685160656823; // Нейрон 3 вес 2
        neurobiasweight [4][1] = 0.8529762783770741; // Нейрон 4 вес 1
        neurobiasweight [4][2] = 0.7818602822337215; // Нейрон 4 вес 2
        neurobiasweight [5][1] = 0.36204206369478864; // Нейрон 5 вес 1
        neurobiasweight [5][2] = 0.6998272379065108; // Нейрон 5 вес 2
        // В этот же массив запишем смещение bias
        neurobiasweight [0][0] = 0.6372796901523922; // Нейрон 0 смещение bias
        neurobiasweight [1][0] = 0.24087856177175287; // Нейрон 1 смещение bias
        neurobiasweight [2][0] = 0.20294311087331438; // Нейрон 2 смещение bias
        neurobiasweight [3][0] = 0.6005802079109952; // Нейрон 3 смещение bias
        neurobiasweight [4][0] = 0.24210376151411686; // Нейрон 4 смещение bias
        neurobiasweight [5][0] = 0.3291532687509765; // Нейрон 5 смещение bias
        // Заполним массив выходных значений нейронов любыми начальными числами
        outputnnum [0] = 0.0;
        outputnnum [1] = 0.0;
        outputnnum [2] = 0.0;
        outputnnum [3] = 0.0;
        outputnnum [4] = 0.0;
        outputnnum [5] = 0.0;

        //---------------------------------------------------
        EditText cxtv = findViewById(R.id.NNeuron); // Текст по умолчанию
        cxtv.setCompoundDrawablesWithIntrinsicBounds(0, R.drawable.vall1000, 0, 0);

        com.example.neuroprng3class.App nprng = new com.example.neuroprng3class.App(); // Объявляем класс
с нейронной сетью
        nprng.trainAndPredict(); // Сгенерировать псевдослучайное число




    } // onCreate
} // MainActivity
```

Такой генератор NPRNG легко генерирует псевдослучайное число огромного размера (280 разрядов) не плохого качества.
Т.к. вычисления выделены в свой класс, можно строить NPRNG огромной сложности и исследовать нелинейность преобразования.

Можно обратить внимание на неудобный формат мастер-ключа – много значений разного формата.
Есть решения и для этой задачи.

Если включать начальные веса нейронов в мастер-ключ (менять их для каждой сессии), то можно создать отдельный псевдослучайный генератор, использующий input1 и input2 для вычисления начальных значений весов и смещений нейронов. Другой вариант – преобразование некоего исходного текста в числа: input1 и input2 и веса и смещения нейронов.

Нейронный PRNG

Часть 3

Нейроны других типов в NPRNG. Нейроны с обучением. Динамические структуры NPRNG