Neural Network PRNG

Android STUDIO JAVA

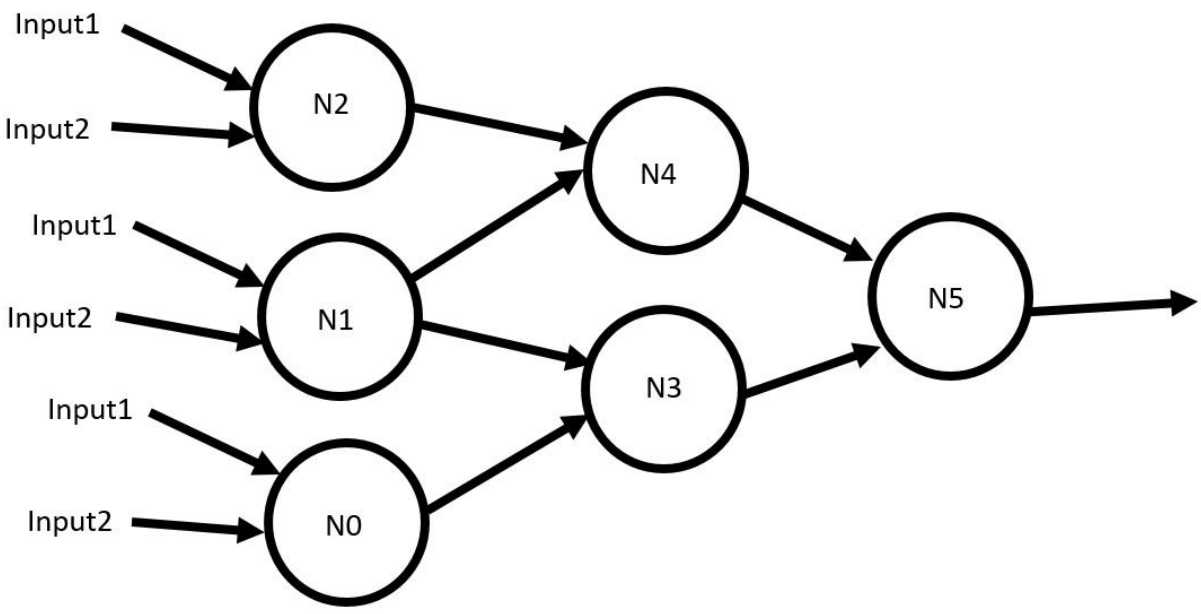
Previously, this was called “sloppily written program code.” Now it’s called obfuscation / Science doesn't yet know who said it /

Obfuscation (from the Latin obfuscare — to shade, to darken; and the English obfuscate — to make non-obvious, confusing, to confuse) or code obfuscation is bringing the source code or executable code of a program to a form that preserves its functionality, but complicates analysis, understanding of the algorithms of work

SOURCE CODE
A software pseudo-random number generator (PRNG) based on a simple neural network without training - NPRNG

Создадим простую нейронную сеть из 6 нейронов и 3 слоев. На первом этапе классическое обучение нейронной сети с обратным распространением ошибки использовать не будем. Используем модули коррекции каждого нейрона индивидуально.

Neural network scheme



Android JAVA Project

Проект программного генератора псевдослучайных чисел состоит из двух JAVA классов

MainActivity.java – содержит начальные параметры для генерирования. Фактически – мастер-ключ
App.java –вычислительная часть

Вывод результата работы выполняется только в Log

Приложение легко генерирует псевдослучайное число размерности 280 символов (разрядов) – 280x8 = 2230bit

Если исходные значения для приложения не менять, будет генерироваться одно и то же число. Т.е. генератор является синхронным – любое количество приложений при одинаковых исходных данных генерирует одно и то же число.

Использование отдельного вычислительного класса App.java позволяет создавать композитные генераторы невероятной сложности (нелинейности).

Приложение отлаживалось на смартфоне Samsung M21

Для воссоздания проекта достаточно всего два файла:

MainActivity.java

```
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
```

App.java

```
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  https://www.linkedin.com/in/valery-shmelev-479206227/

// 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


        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

            income5bigint(); // Накопить псевдослучайное число только из выходных значений нейронной сети
        }
```

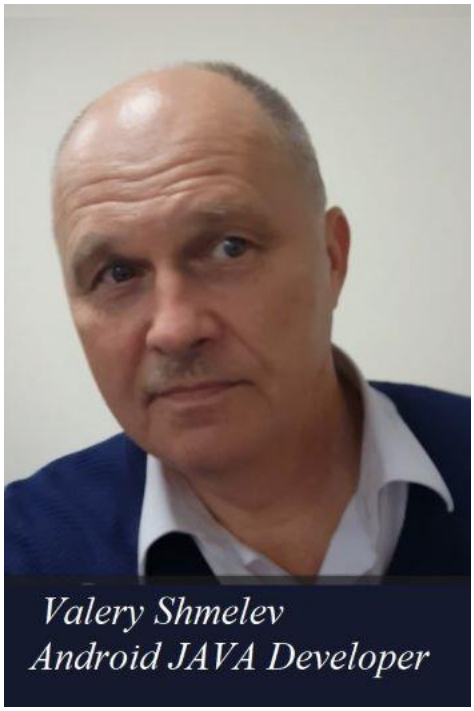
```
      Log.i("== PSEUDORANDOM =", " ==>>> >>> >>>== NPRNG OUTPUT pseudornd=" +
pseudornd);

   }

   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
```

Это очень простой NPRNG. Есть много способов его улучшения. Например, использование нейронов с особенной структурой или создание комплексных PRNG.

Этот прототип использует довольно сложный мастер-ключ, состоящий из входных значений input1 и input2 и весов и смещений для каждого нейрона.
Далее мы покажем как это можно сделать более удобным для использования – мастер-ключ будем создавать из текста.



Neural Network PRNG