NeuralBasedPRNG

Developing a JAVA application on Android Studio

Coherent PRNGs with neural networks

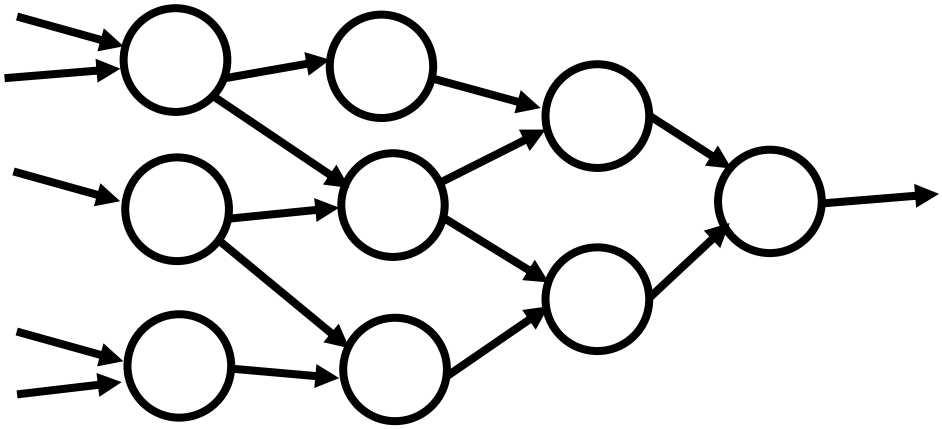
*Coherent – several PRNG instances independently generate identical arrays of pseudo-random numbers with the same starting values (with the same master key) without any interaction.

** The master key here has the meaning of the starting conditions for the neural network.

At the same time, such generators may not have a constant generation algorithm (it can change at any step) or a constant internal structure and may fundamentally never repeat (it is possible to exclude repetitions).

We will proceed from the assumption that any software generator of "random" numbers will sooner or later repeat. Therefore, we will rebuild the neural network (not only the weight and bias of each neuron, but the structure of the neural network - the number of layers, neurons in layers, the structure of connections) for maximum variability of the "pseudo-number" generation algorithm.

*** Here "pseudo-number" is the value used to calculate one bit in the generated pseudo-random number. As an example, in one step of NeuroPRNG, each neuron produces one "pseudo-number" that is used to further calculate one bit (the bit is not used directly in the generated number!).



The choice of possible NeuroPRNG (NPRNG) variants is huge.

For example. Part of the master key is used to calculate the number of neurons, layers, and to distribute neurons across layers in the neural network being created.

Another part of the master key determines the number of connections and their distribution across neurons. The third part of the master key specifies weights and biases.

An example of the interaction of 6 neurons in three layers (a well-known example on the Internet). Here, only the numbers of neurons are added

```
return neurons.get(5).compute(
    neurons.get(4).compute(
        neurons.get(2).compute(input1, input2, 2),
        neurons.get(1).compute(input1, input2, 1),4
    ),
    neurons.get(3).compute(
        neurons.get(1).compute(input1, input2,1),
        neurons.get(0).compute(input1, input2,0),3
    ),5
);
```

In this form, it is not convenient to rebuild their number and interaction scheme.

Therefore, we will rewrite the network differently. In reality, one neuron will be used, which will emulate all layers and neurons of the network. In several arrays, we will remember the input and output values for each neuron, weights, mixing and, later, the type of the function used.

This version of the neural network will work slower, but we will be able to implement any configurations dynamically.

Neural PRNG

We will test all this on a prototype of a neural crypto network without training.

Even in this case, we can get pseudo-random numbers (each number) calculated using completely different formulas and with a new repetition period each time. Crypto analysts will not like it. It is possible to achieve that even crypto-analysis using special neural networks will be useless. Moreover, this method of generating pseudo-random numbers will have a low "entry threshold". There is no need to carefully select coefficients and know the nuances of calculation to ensure crypto-resistance.

An infinite number of formulas, network structures and sizes of generated numbers.

Читаем далее…

P.S. These projects are a great base for your own tests and developments.

One of the "dynamic" projects (a consistent series of JAVA projects in development) is SimpleNNeuron neural network on Android JAVA. In fact, how to write a simple neural network and train it.

http://multidoc.oflameron.com/page004.htm

http://site.oflameron.ru/page005.htm

http://webpage.pips.ru/page0006.htm

http://referat.oflameron.ru/page0001.htm

http://picture.oflameron.ru/