NeuralBasedPRNG

Developing a JAVA application on Android Studio

Coherent PRNGs with neural networks

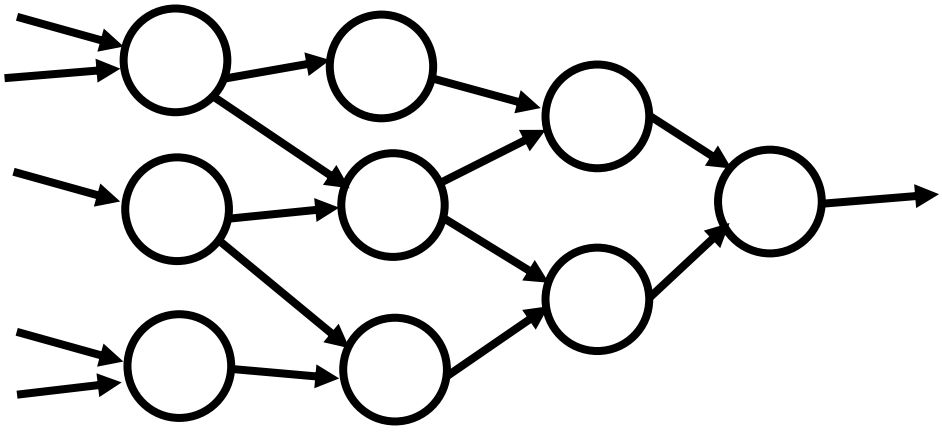
*Coherent – several PRNG instances independently generate identical arrays of pseudo-random numbers with the same starting values (with the same master key) without any interaction.

** 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 directly used 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.

Several (tens, hundreds, thousands...) training cycles are performed on a given dataset (not public).

After that, such a "neural module" can be used in PRNG.

Neural PRNG

The simplest example. The output values of each neuron (Big Int) are used to find the closest pair of prime numbers for the Blum-Blum-Shub algorithm (not crypto-resistant).

A parity bit is used from the obtained result. Then the next cycle of work is performed. The calculated and unused data from neurons is fed to the input. Another bit of the generated pseudo-random number is obtained. And so on, until a Big Int "random" number of the desired dimension is formed.

Very interesting. And this is still without using generator retraining and without reconfiguring the neural network.

Such a PRNG will be slow, not for stream encryption (unless we consider scalable NPRNGs on a single neuron or layer implemented on a separate chip). The pseudorandom number generators considered here are interesting primarily from the point of view of counteracting cryptanalysis. For every few pseudorandom numbers, a separate generation algorithm will be created. There is huge potential here.

Next, let's try to build the simplest "neural" PRNG without training. At first, it will be a simple neural network of 6 neurons. With layers of 3,2,1 neurons. Then we will add methods for obtaining "pseudo numbers", methods for extracting one bit and accumulating the output pseudorandom number. Then we will make this neural part of the NPRNG dynamically reconfigurable (with changeable parameters, connections and structure). Add a layer and connections.

Read more...

P.S. These projects are a great base for your own developments, testing ideas and training.

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