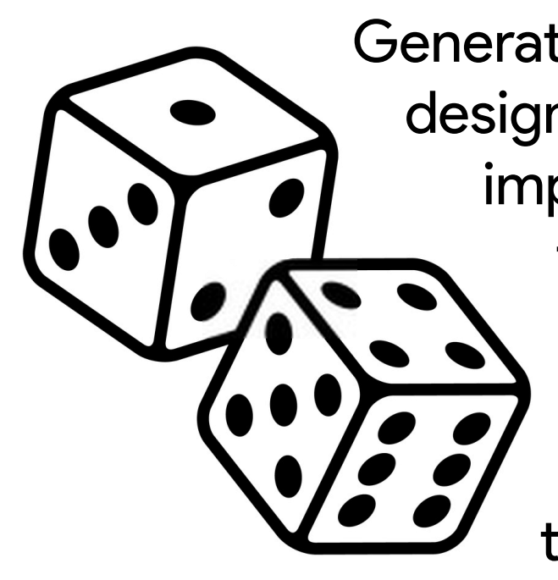
pp: #034

Using Generative Adversarial Networks for Secure Pseudorandom Number Generation

Abstract



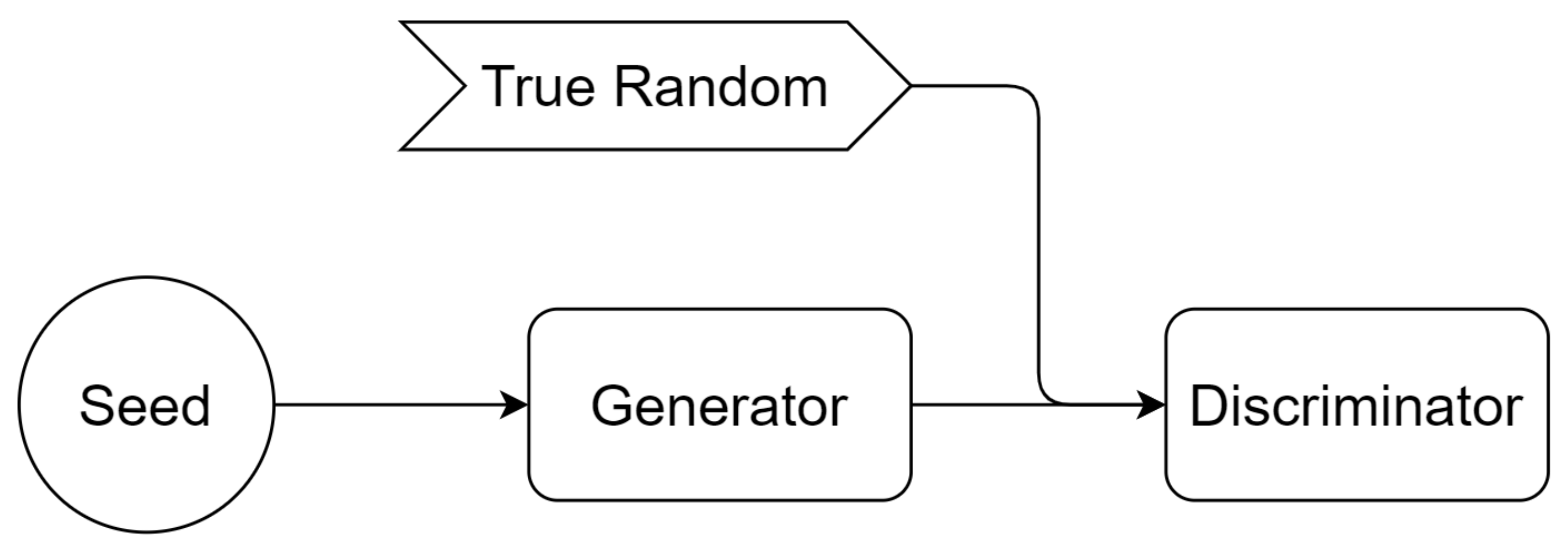
Generation of secure random numbers has always been a challenging issue in design and development of secure computer systems. Random numbers have important applications in the field of cryptography where the security of the scheme relies upon the random nature of the keys. It is not practically possible to achieve true randomness in a machine, and hence we rely upon **Pseudo Random Number Generators** (PRNGs) to produce near-true randomness. PRNGs use a mathematical function that relies upon a seed (a preset value required by the function to generate values) and it generates numbers which satisfy certain tests for randomness and appear to be random for a user having no knowledge of the generator function. To generate random numbers that can never be predicted by any observer, requires a causally non-deterministic process where events are not fully determined by prior states. Various methods to generate pseudorandomness have been employed over the years which includes using mathematical functions, keyboard typing latency of the user, network latency, memory latency, etc. In this work, we propose a new way of generating pseudorandom numbers using **Generative Adversarial Networks**. We demonstrate that a GAN can act as a **Cryptographically Secure Pseudorandom Number Generator** (CPRNG) passing 97% of **National Institute of Standards and Technology** (NIST) tests.

1. Introduction

Random numbers are needed everywhere. Encryption algorithms heavily rely on random numbers to generate secure keys. Entropy and randomness are required in initialisation of Neural Network weights, statistical analysis, computer simulations, etc. Unbreakable pseudorandom number generator functions are at the core of computer security. Thus, it is imperative to come up with better techniques for generating numbers which are near-true random.

Because true randomness cannot be achieved, computer systems generally rely on a Pseudo Random Number Generator (PRNG), which generates a stream of random numbers based on a seed value. For a function $G_k : \{0, 1\}^k \rightarrow \{0, 1\}^{p(k)}$ where G is a PRNG if and only if the next output bit of G cannot be predicted by a polynomial time algorithm.

The National Institute of Standards and Technology has proposed a set of tests to analyse the strength of pseudorandom numbers. We have tested our method against these standardised tests. The notion behind using Generative Adversarial Networks is due to the peculiar property of the two networks - **Discriminator** and **Generator**, contesting with each other. This technique allows the generation of close to true random numbers. The Generator is forced to generate samples that look real and the Discriminator learns to distinguish between generated samples and samples from real data.



2. Overview

Importance of Random Numbers

Randomness in the choice of keys and other parameters is an important reason why many systems are secure against brute force attacks by hackers. Many cases have surfaced over the years, where hackers had compromised computer systems and servers just due to weak PRNGs used in the software. The unpredictability of random number generation is also the Achilles' Heel of encryption. If the randomness becomes predictable, encryption schemes are broken.

Generative Adversarial Networks

A Generative Adversarial Network comprises of two models trained simultaneously. One is a Generative model 'G', that captures data distribution, and the other is a Discriminative model 'D' that gives the probability of the sample coming from the training data rather than 'G'. In short the model 'G' generates probable candidates, and the model 'D' evaluates the correctness of the candidate. The discriminator is trained with existing data, until acceptable accuracy is produced and then the Generator trains based on whether it succeeds in fooling the Discriminator.

3. Proposed Methodology

Our approach involves the Generator model of the GAN (Generative Adversarial Network) producing a floating point number array which is then normalised, and truncated to generate binary strings. The Discriminator model then states how likely this output came from the training set and not from the artificially generated (truly) random source.

The NIST test suite performs **188 statistical analysis tests** on these values.

The Generator model is a fully connected 3-layered neural network. The input to the network is a seed matrix (a randomly initialised matrix), and the output is a 256×1 matrix which is then passed to the Discriminator model. The Discriminator model is provided with the true source of randomness as well as the output of the Generator model. The true source of randomness is mimicked by using the values produced by a **Gaussian Distribution Function** as shown below:

$$P(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-(x-\mu)^2/2\sigma^2}$$

We trained the Generator-Discriminator model for **1000 epochs** with **batch size of 256.**
On the 1000th epoch, the generated floating point numbers were normalised using:

$$z = \frac{x - min(arr)}{max(arr) - min(arr)}$$

and rounded off to the nearest integer value. These produced bit-streams were then tested against the NIST test suite. The Generator model uses the **tanh** activation function and the Discriminator model uses the **sigmoid** activation function.
The optimiser used is the vanilla **Gradient Descent** optimizer.

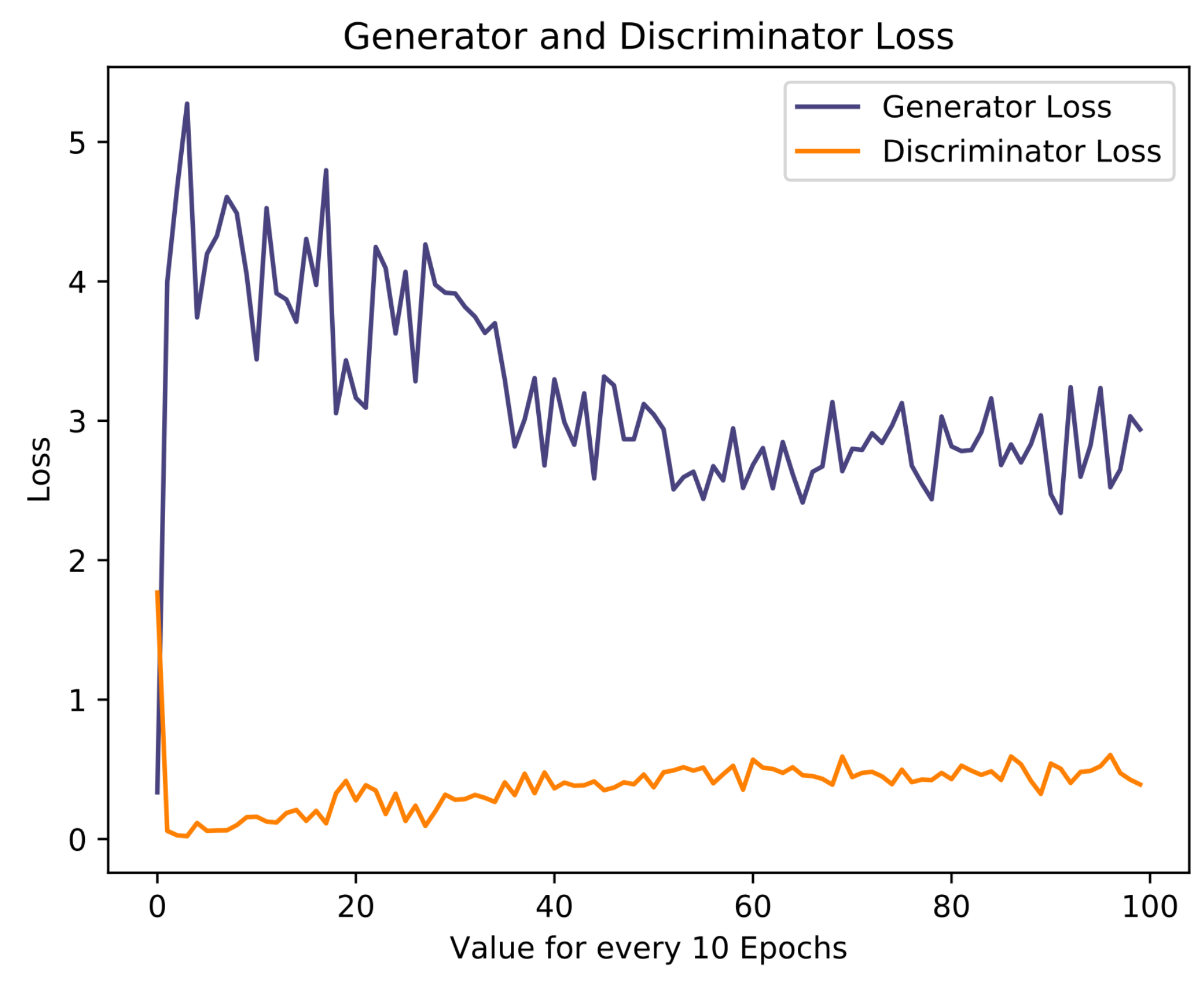
$$\tanh(x) = \frac{(e^x - e^{-x})}{(e^x + e^{-x})}$$

$$\sigma(x) = \frac{e^x}{1 + e^x}$$

4. Results

For every test, NIST reports the **p-value** of all individual instances. A test instance fails if its p-value is below a critical value (α = 0.01). An overall test fails if either the number of passed instances is below a threshold, or the p-value for the distribution is below a critical value. The Generator network fails all the tests on the first epoch and the results improve consistently over time. Following table shows the NIST test results. T is the number of tests conducted by NIST test suite, F_p is the number of tests failed and F is the percentage of failure. Initially, the model just passes a single test and the performance improves drastically after 1000 epochs of training.

i	T	F_p	F%
Before Training	188	187	99.46
After Training	188	6	3.19



Conclusion

The notion behind using a completely different approach for generating secure random numbers was to compare the performance with existing implementations and check for the reliability to deploy this system for practical and cryptographic purposes. Using a Generative Adversarial Neural Network allowed us to exploit the property of the neural network to improve its performance under the supervision of the Discriminator. This methodology for generating random numbers is currently in its experimental stage, due to the intensive amount of computation involved in generating the bit-stream. The 256,000 bits long stream generated by this network has passed 97% of the NIST tests, thereby outperforming most of the mathematical function-based CPRNGs.

Acknowledgement

This research was supported by Pune Institute of Computer Technology. We would like to thank the anonymous reviewers for their valuable insights. We are also immensely grateful to Stefan Brunthaler (Poster Chair), Jorge Blasco Alis & Daniele Sgandurra (Registration Chairs) and many others for their unwavering support, without which this research could not have been possible.

Like our work? Feel free to contact us! Scan this QR »

Authors: Chaitanya Rahalkar, Dhaval Gujar, Rajvardhan Oak