

Predicting Pseudo Random Values Using Convolutional Neural Networks

Najib Ali

*Department of Computer Science
Middle Tennessee State University
Murfreesboro, Tennessee
email address*

Jacob Anderson

*Department of Computer Science
Middle Tennessee State University
Murfreesboro, Tennessee
email address*

Spencer Arnold

*Department of Computer Science
Middle Tennessee State University
Murfreesboro, Tennessee
email address*

Matthew Hawks

*Department of Computer Science
Middle Tennessee State University
Murfreesboro, Tennessee
email address*

Ryan Hines

*Department of Computer Science
Middle Tennessee State University
Murfreesboro, Tennessee
email address*

Tae Kweon

*Department of Computer Science
Middle Tennessee State University
Murfreesboro, Tennessee
email address*

Abstract—This document is a model and instructions for \LaTeX . This and the `IEEEtran.cls` file define the components of your paper [title, text, heads, etc.]. ***CRITICAL: Do Not Use Symbols, Special Characters, Footnotes, or Math in Paper Title or Abstract.** This document is a model and instructions for \LaTeX . This and the `IEEEtran.cls` file define the components of your paper [title, text, heads, etc.]. ***CRITICAL: Do Not Use Symbols, Special Characters, Footnotes, or Math in Paper Title or Abstract.** This document is a model and instructions for \LaTeX . This and the `IEEEtran.cls` file define the components of your paper [title, text, heads, etc.]. ***CRITICAL: Do Not Use Symbols, Special Characters, Footnotes, or Math in Paper Title or Abstract.** This document is a model and instructions for \LaTeX . This and the `IEEEtran.cls` file define the components of your paper [title, text, heads, etc.]. ***CRITICAL: Do Not Use Symbols, Special Characters, Footnotes, or Math in Paper Title or Abstract.**

Index Terms—component, formatting, style, styling, insert

I. INTRODUCTION

Start typing here [1].

II. BACKGROUND

Start typing here [2].

III. METHODS

A. Seeding Method

We went with a seed generation method that allowed a way to introduce some minor level of entropy to avoid letting the neural network aimlessly swim through the entropy of a strong seed instead of gaining stochastic insight on the data from the PRNG.

The seed generation method we chose derives from the concept of using the system time as an element for seed generation. The specific implementation we chose took inspiration from Microsoft's .NET system.datetime.ticks property. [1] We chose to single out this method due to its documentation and unique simplicity. System time is widely used as a parameter for

modern seed generation methods. To put it more trivially: "a pseudo random number generator is a deterministic algorithm that, given an initial number (called seed), generates a sequence of numbers that adequately satisfies statistical randomness tests. Since the algorithm is deterministic, the algorithm will always generate the exact same sequence of numbers if it's initialized with the same seed. That's why system time (something that changes all the time) is usually used as the seed for random number generators." [2]

According to the Microsoft documentation, "A single tick represents one hundred nanoseconds or one ten-millionth of a second. There are 10,000 ticks in a millisecond, or 10 million ticks in a second. The value of this property represents the number of 100-nanosecond intervals that have elapsed since 12:00:00 midnight, January 1, 0001 in the Gregorian calendar." [1]

We used a fairly similar Python port, explained below. [3]

```
def ticks(dt):  
    return (dt-datetime(1,1,1)).  
        total_seconds()*10000000
```

Porting side effects include:

- 1) UTC times are assumed.
- 2) The resolution of the datetime object is given by datetime.resolution, which is datetime.timedelta(0, 0, 1) or microsecond resolution (1e-06 seconds). CSharp Ticks are purported to be 1e-07 seconds.

Additional changes we made to our implementation:

- 1) Changing start time from January 1, 0001 to January 1, 1970, which effectively reduced the length of the seed for experimental purposes.
- 2) Slicing last 6 digits of the ticks result to acquire more digit variation for frequent invocation.

This final method allows enough spread between frequently retrieved ticks, where we are assuming reasonable pseudo-

unpredictability. This serves as a simplistic but constantly changing control mechanism for being able to seed PRNGs and test experimental outcomes. While not the most cryptographically strong, we needed a way to have some controlled aspect of seed generation to feed into generators of varying cryptographic complexity (to have some baseline of comparison).

The original idea was to feed each PRNG different seeds from the same seed generator; however, many PRNG algorithms impose strict seed requirements to pass tests of randomness. Out of the five PRNG methods we implemented, Lagged Fibonacci was the only one that had special seed requirements, so we created a separate seed generator based on the same fundamental ticks generation method, but modified to meet the restrictions. Other PRNGs not implemented in this research that impose seed restrictions include Wichmann-Hill (which accepts three different seeds) and Maximally Periodic Reciprocals (which requires a Sophie Prime), among others.

You might ask: won't different seed generators introduce flaws or bias in the experiment? Well, it depends on what you are testing. In our case, we are strictly testing the "complexity" of the generator itself, so supplying a seed that is not blatantly predictable but also not unpredictable was sufficient. Our goal was to allow the characteristics of the generator to be exposed, for we were cracking the "complexity" of the generation algorithm, not the complexity of an arbitrary seed.

D. Experimental Setup

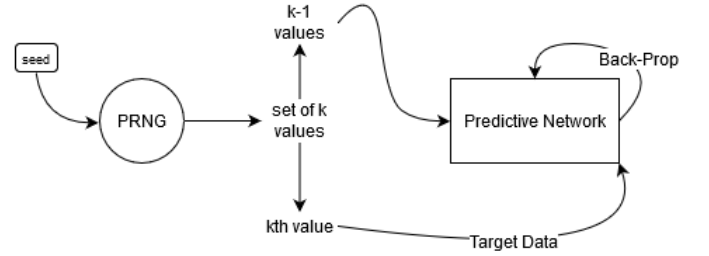


Fig. 2. Experiment Model

E. Experimental Setup

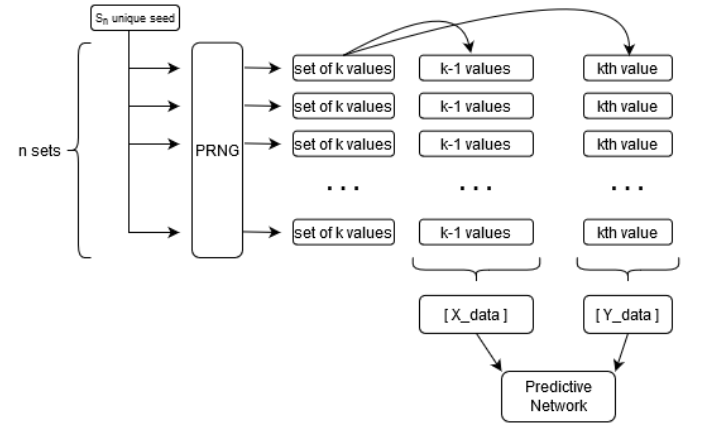


Fig. 3. Experiment Model

B. PRNG Implementations

C. Predictive Network

NOTE THE Equation will have to be re-done and diagrams re-drawn to reflect our experiment. The predictor is a convolutional neural network implementing the function ... : $P(r_{split})$ 7B(7)whersplitis the generators output vector with the last element removed.The last element is used as the corresponding label for the predictors input.

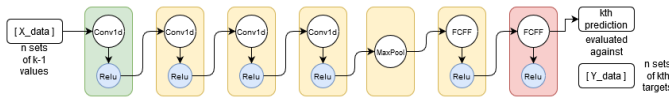
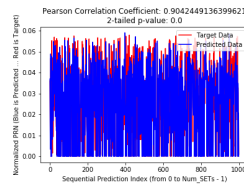
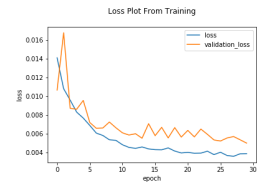


Fig. 1. Predictive Model

IV. RESULTS

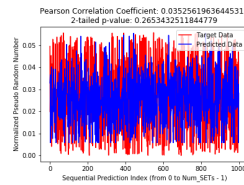


(a) Testing Regression Plot

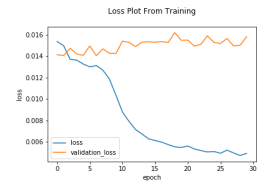


(b) Training Loss Plot

Fig. 4. Middle Square Results



(a) Testing Regression Plot



(b) Training Loss Plot

Fig. 5. Linear Congruential Results

