CCA Project

Neural Networks Performance for Generating Standard Random Distributions

Supervised by Pr. TURKI Abbas Lokmane and GRENARD Aurélien

G3 : Samy HORCHANI (28706765) & Assia MASTOR (28602563)

May 27, 2024

SCIENCES SORBONNE UNIVERSITÉ

Table of contents

1. Introduction

2. Methodology

3. Results

4. Conclusion

Introduction

Methodology

Methodology

Understanding Mathematical Concepts

Understanding Mathematical Concepts

What questions did we ask ourselves?

- What is a random variable?
- How do we define probability distributions?
- Why are these concepts important in simulations?
- How do random number generators work?

Random variables

Random variable

A real random variable (resp. a random vector) on Ω is an application $X : \Omega \to \mathbb{R}$ (resp. $\Omega \to \mathbb{R}^m$) such that for any open interval I of $\mathbb{R}$ (resp. an open box in $\mathbb{R}^m$) $X^{-1}(I)$ is a event (so in $\mathcal{B}$).

Probability Distributions

- **Threshold-based Generation:**
  - Bernoulli Distribution
- **Summation of Trials:**
  - Binomial Distribution
- **Transform Methods:**
  - Chi-Squared Distribution (Box-Muller transform)
  - Exponential Distribution (Inverse transform sampling)
- **Rejection Sampling:**
  - Poisson Distribution

How Do Random Number Generators Work?

- **Sequential Generation:**
  - Uses a seed to initialize
  - Generates a deterministic sequence
- **Linear Congruential Generators (LCG):**
  - Commonly used method
  - Formula: $X_n = (aX_{n-1} + c) \mod m$
- **Then used in Advanced Methods:**
  - Inverse Transform Sampling
  - Rejection Sampling
  - Box-Muller Transform

Methodology

Implementation

In Python

- **Why Python?**
  - Simplicity and readability
  - Extensive libraries available
- **Libraries Used:**
  - NumPy
  - PyTorch
- **Development Process:**
  - Used built-in functions for initial implementation
  - Coded our own generators for comparison

In C

- **Why C?**
  - Improved performance and efficiency
  - Low-level memory control
  - Preparing for GAN implementations with CUDA/C++
- **Translation Process:**
  - Translate algorithms from Python to C
  - Optimize for performance in C

Re-coding with the Professor's RNG

- **Integration of Professor's RNG:**
  - Adapted RNGs provided by the professor
  - Ensured compatibility with our code
- **Optimization:**
  - Fine-tuned algorithms using these RNGs
  - Enhanced performance and accuracy

What are GANs?

Generative Adversarial Networks (GANs)

- **Concept:**
  - Introduced by Ian Goodfellow in 2014
  - Composed of two neural networks: the **Generator** and the **Discriminator**
- **Generator:**
  - Generates new data instances
  - Learns to create data that mimics real data
- **Discriminator:**
  - Evaluates the authenticity of the data
  - Distinguishes between real and generated data
- **Applications:**
  - Image generation, Data augmentation, Anomaly detection ...

Results

Results

Comparison of Execution Times

On CPU — Built-in Python functions : NumPy vs PyTorch

$n\_sample = 1,000,000$ $p = 0.5$ Distribution simulated with built-in functions	Numpy with a 'for' loop	Numpy without a 'for' loop	Pytorch
Bernoulli	0.80000s	0.02087s	0.02651s
Binomial ($n = 100$)	0.81285s	0.08303s	0.22444s
Exponential ($\lambda = 2$)	0.60388s	0.01727s	0.06846s
Poisson rejection ($\mu = 5$)	0.8195s	0.13516s	0.19474s
Chi-squared (d = 10)	0.68824s	0.16115s	0.23700s
Normal	0.85456s	0.05965s	0.01545s

On CPU — With our custom generators in Python using Numpy and Pytorch

$n\_sample = 1,000,000$ $p = 0.5$ Distribution simulated	Custom generators using NumPy	Custom generator using PyTorch
Bernoulli ($n = 100$)	0.02893s	0.05091s
Binomiale with Bernoulli ($n = 100$)	1.22517s	1.46297s
Binomiale without Bernoulli	1.87530s	
Exponentielle ($\lambda = 2$)	3.62260s	0.10872s
Poisson rejection ($\mu = 5$)	11.88620s	3.66399s
Chi-squared (d = 10)	0.16001s	1.19594s
Compound Poisson	3.48699s	2.73748s
Noncentral chi-squared	8.63048s	

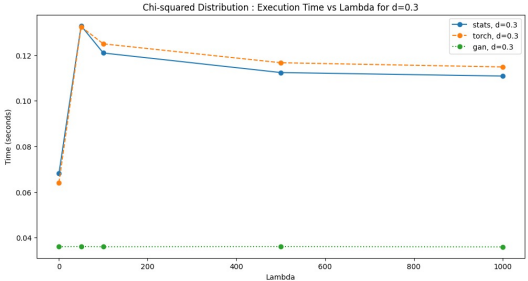
On CPU — With our custom generators in C/C++ using built-in rand() and our RNG

$n\_sample = 1,000,000$ $p = 0.5$ Distribution simulated	using built-in rand() function in C	using our RNG in C++
Bernoulli	0.08834s	0.09479s
Binomial with Bernoulli ($n = 100$)	2.11896s	0.06306s
Binomial without Bernoulli ($n = 100$)	2.75088s	1.30529s
Exponential ($\lambda = 2$)	0.05505s	0.08671s
Poisson rejection ($\mu = 5$)	0.266195s	0.01859s
Chi-squared (d = 10)	0.79669s	0.10384s

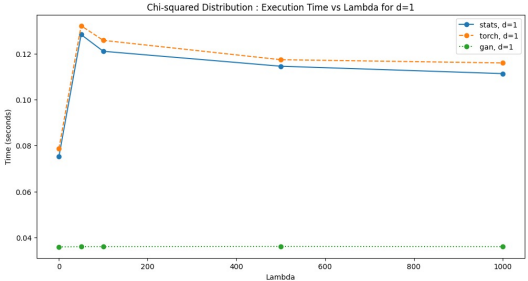
On GPU — With our custom generators using PyTorch in Python and our RNG in CUDA

$n\_sample = 1,000,000$ $p = 0.5$ Distribution simulated	in Python using PyTorch	our RNG in C using CUDA
Bernoulli	0.00054s	0.000703s
Binomial with Bernoulli ($n = 100$)	0.00052s	0.000582s
Exponential ($\lambda = 2$)	0.00036	0.000058s
Chi-squared (d = 10)	0.00568s	0.000099s
Poisson distribution ($\mu = 5$)	0.01825s	0.000202
Compound Poisson Distribution	0.019852s	

GAN — Chi-squared distribution varying Degrees of Freedom

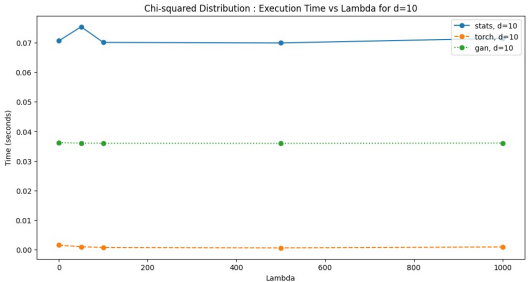


Degree of freedoms = 0.3

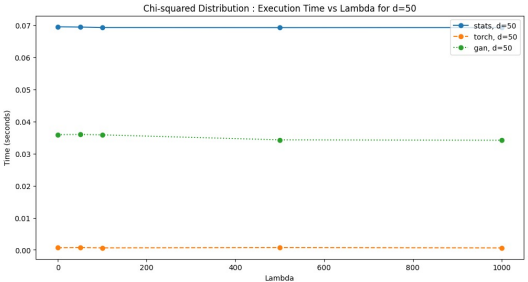


Degree of freedoms = 1

GAN — Chi-squared distribution varying Degrees of Freedom

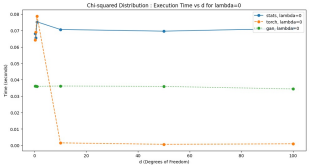


Degree of freedoms = 10

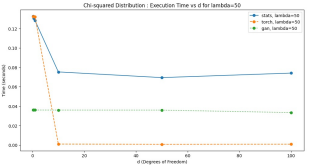


Degree of freedoms = 50

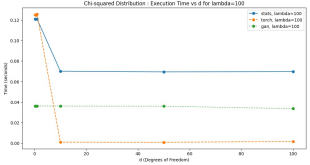
GAN — Chi-squared distribution varying Lambda



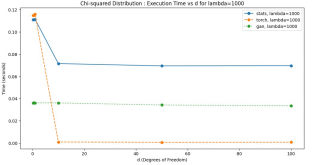
$\lambda = 0$



$\lambda = 50$

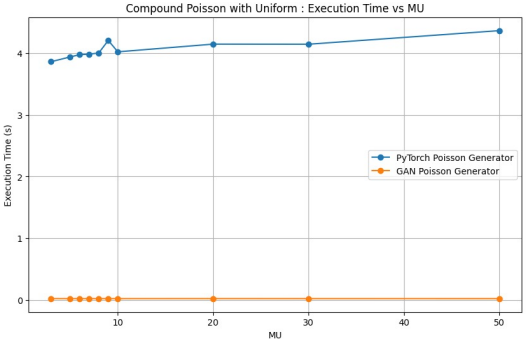


$\lambda = 100$

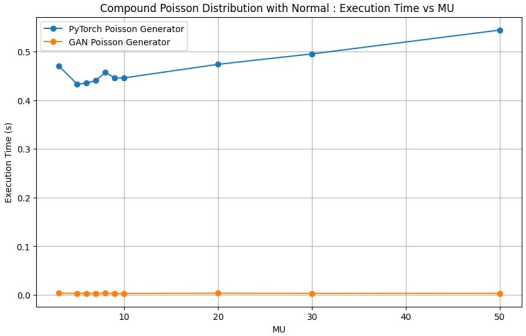


$\lambda = 1000$

GAN — Compound Poisson distribution with Uniform vs with Normal

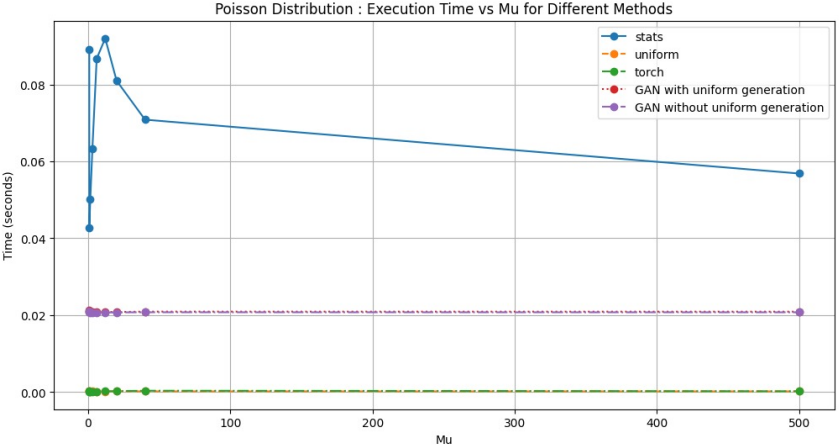


Compound Poisson with Uniform



Compound Poisson with Normal

GAN — Poisson distribution using various methods



Results

Defects and Improvements

Defects and Improvements

- **Identified Defects:**
  - Inconsistent performance across different distributions
  - All our GANs are slower than our parallelized GPU code
  - Lack of certain functions that we couldn't implement
- **Potential Improvements:**
  - Enhance parallelization
  - Optimize GAN training for better performance
  - Conduct more extensive benchmarking across varied hardware
  - Explore hybrid models combining traditional methods with GANs

Conclusion

Thank You For Your Attention