APPLICATIONS OF GANS ON NON-IMAGE DATA

- INSTRUCTOR – TEJAS BODAS

- ROHIT REDDY LINGALA

2020102035

AIM:

- Understand the working of GANS.
- How GANs can be used in simulating random variables and Markov chains?

What are GANS?

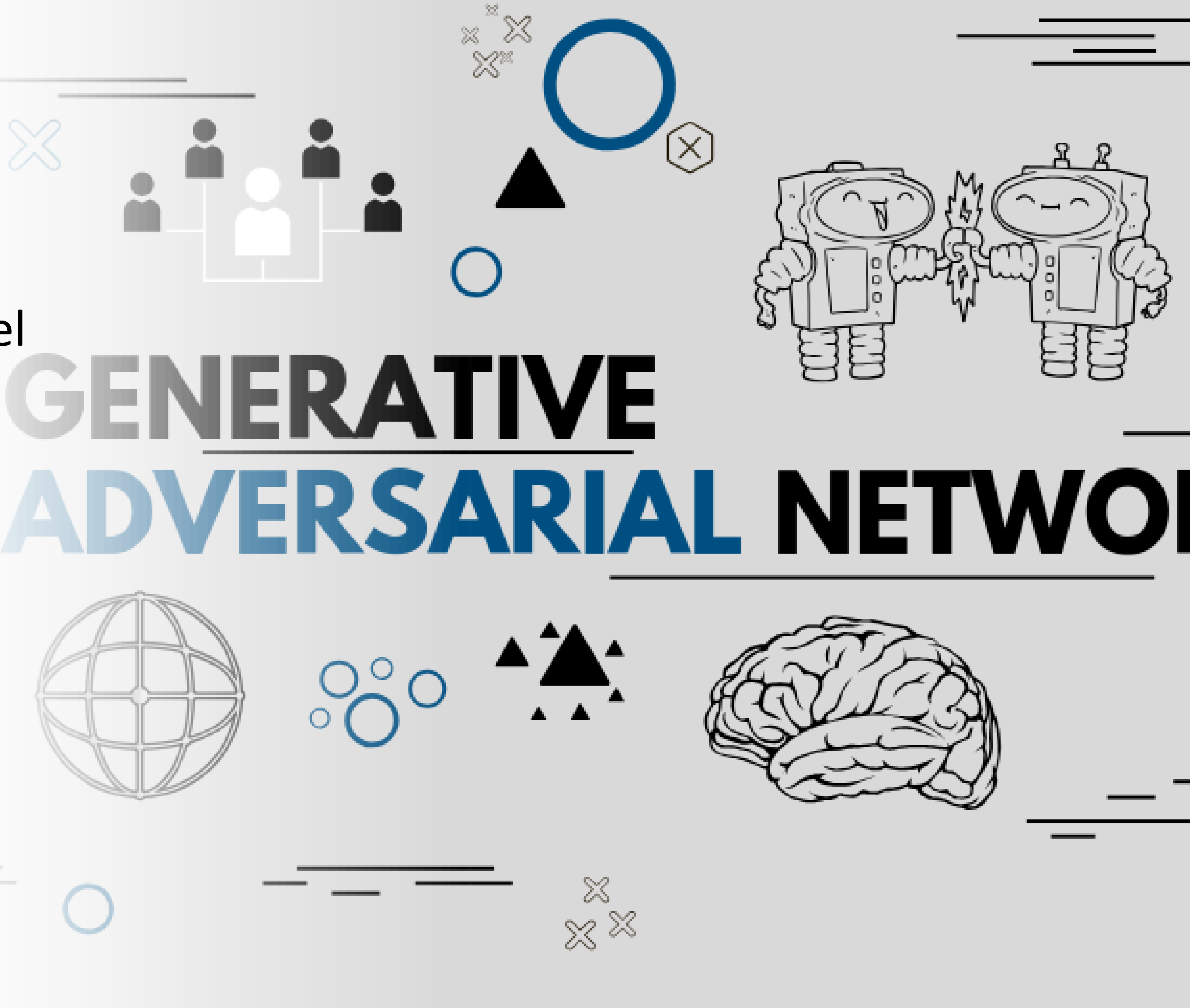
- Generative adversarial networks, or GANs, are an approach to generative modelling using deep learning methods.
- Generative modelling is an unsupervised learning task in machine learning that involves automatically discovering and learning the regularities or patterns in input data in such a way that the model can be used to generate or output new examples that plausibly could have been drawn from the original dataset.

What are GANS?

- GANs are a clever way of training a generative model by framing the problem as a supervised learning problem with two sub-models:

- The generator model that we train to generate new examples.

- The discriminator model that tries to classify as real (from the dataset) or fake (generated by generator).



- GANs work like a two-player game.
- Both the generator and discriminator models are trained together.
- In each step, the generator generates a batch of samples, and these, along with the real samples (from our dataset), are classified to the discriminator to be classified as real or fake.
- Then the discriminator is then updated to get better at discriminating real and fake samples in the next round, and the generator is updated based on how well, the generated samples were able to fool the discriminator.

LOSS FUNCTIONS

- The discriminator loss is as follows:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D\left(\boldsymbol{x}^{(i)}\right) + \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right) \right]$$

- The generator loss is as follows:

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right)$$

for number of training iterations **do**

for k steps **do**

- Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
- Sample minibatch of m examples $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ from data generating distribution $p_{\text{data}}(\boldsymbol{x})$.
- Update the discriminator by ascending its stochastic gradient:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D\left(\boldsymbol{x}^{(i)}\right) + \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right)\right].$$

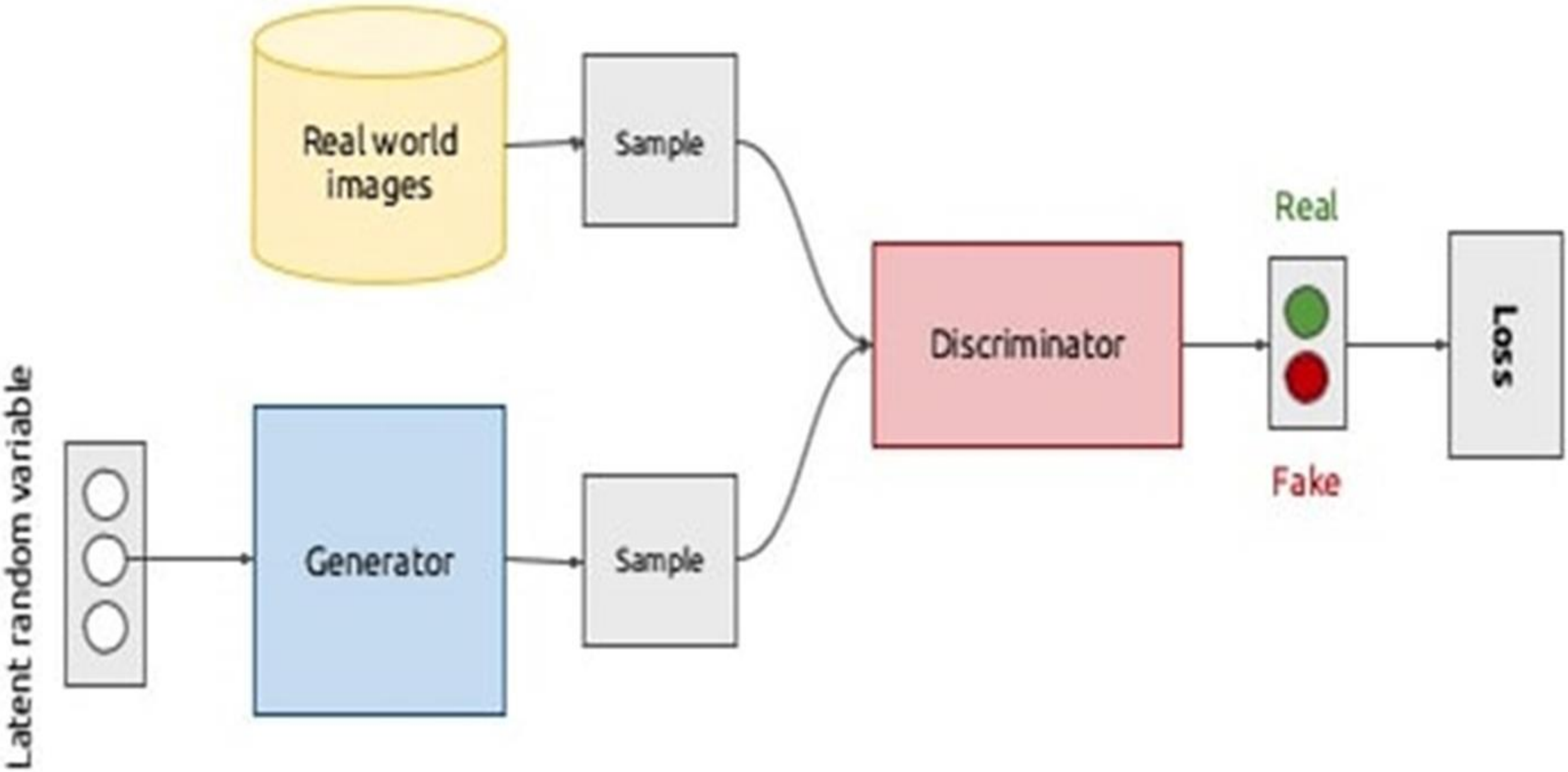
end for

- Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
- Update the generator by descending its stochastic gradient:

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right).$$

end for

The gradient-based updates can use any standard gradient-based learning rule. We used momentum in our experiments.



Real world images
Sample
Real
Discriminator
Loss
Fake
Latent random variable
Generator
Sample

But what about Random variables and Markov chains?

- GANs are extensively used in image generation. But can it be used to simulate random variables and Markov chains as well?
- The purpose of this project is to study how GANs, can be used to simulate random variables and Markov chains as it can have numerous applications.

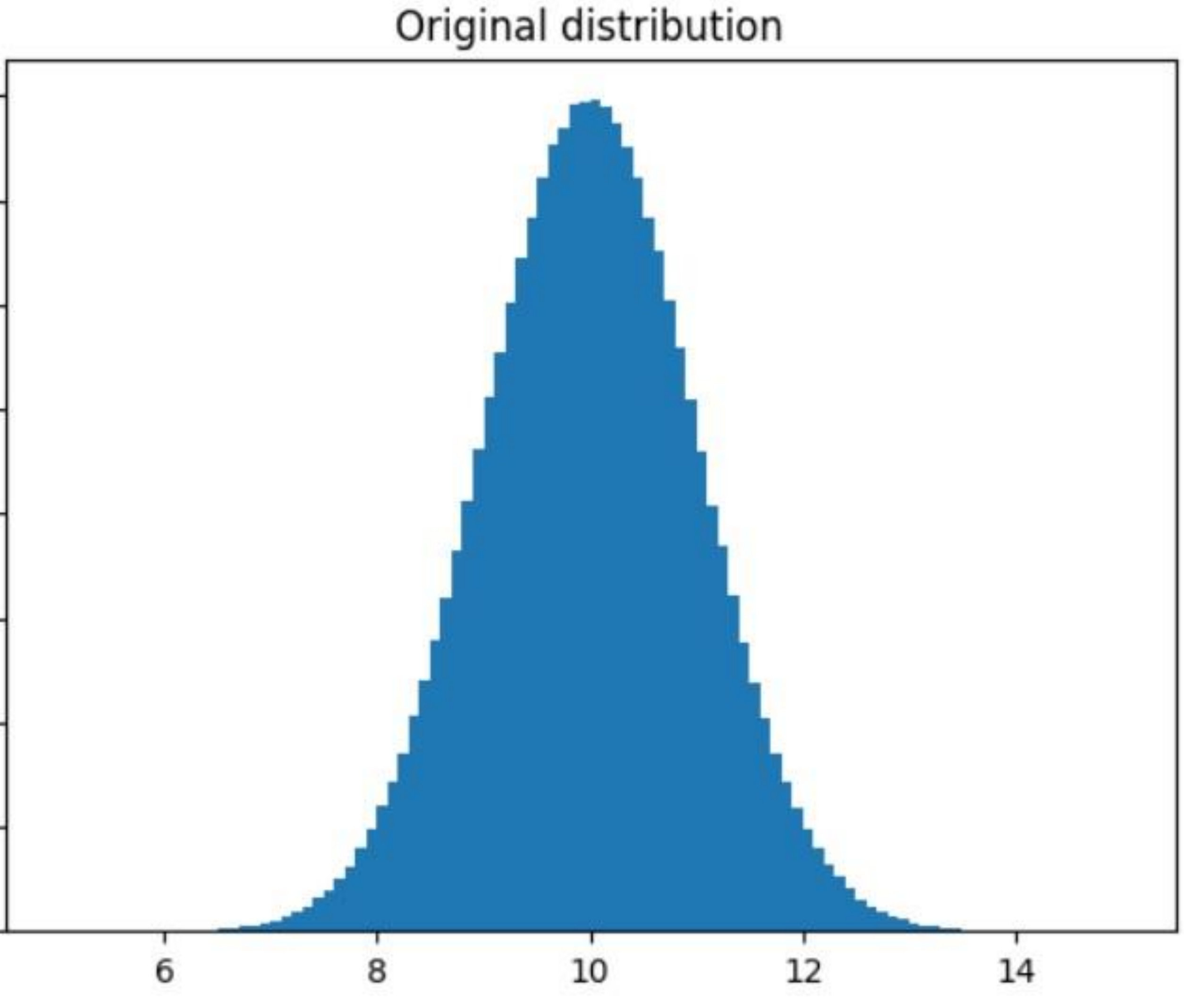
- It can be used to generate financial time series data. It can be used to simulate future price scenarios.

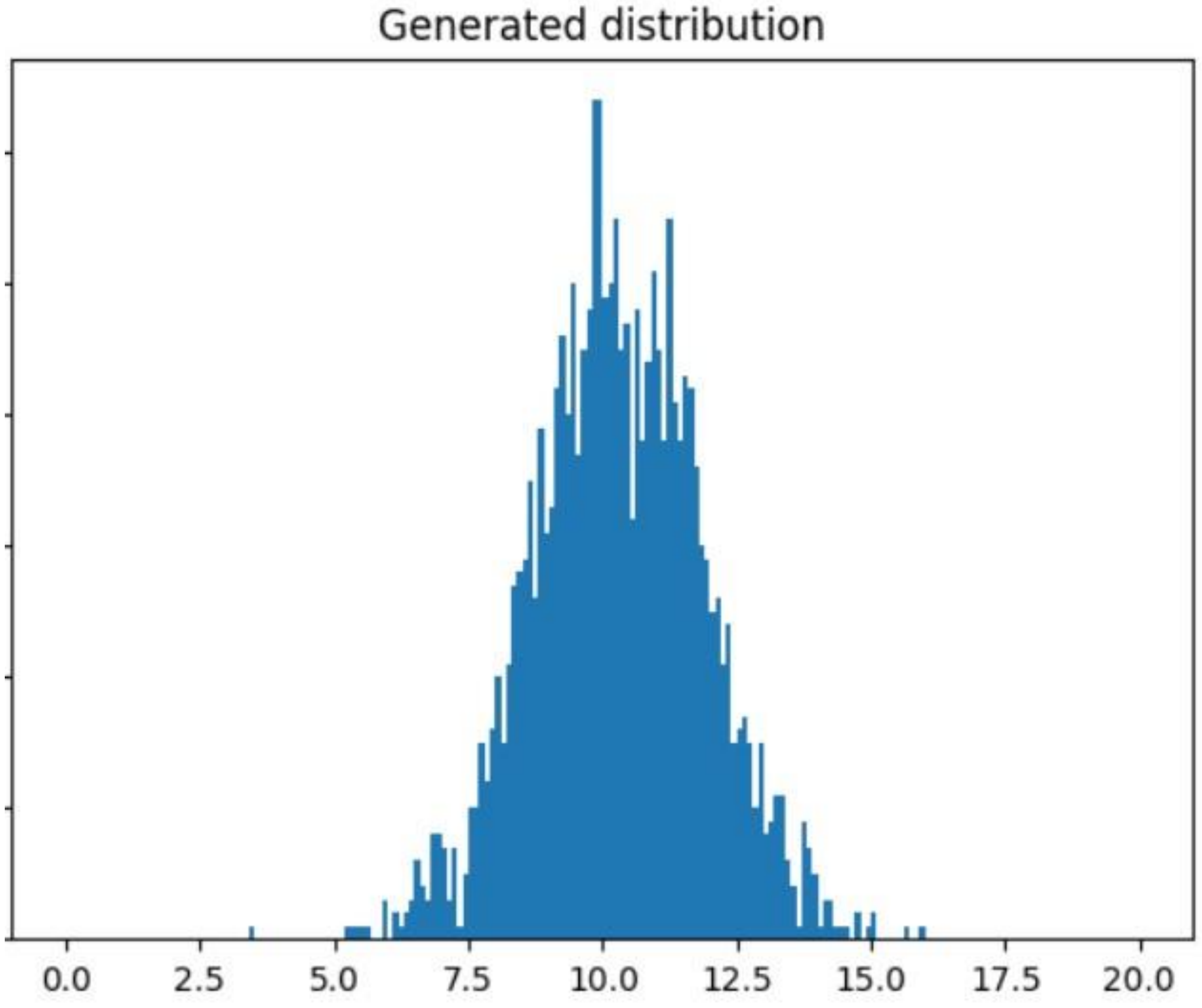
- It can also be used to generate musical sequences. It can be used to model the transition between musical notes and chords.

- It can also be used to generate synthetic climatic data

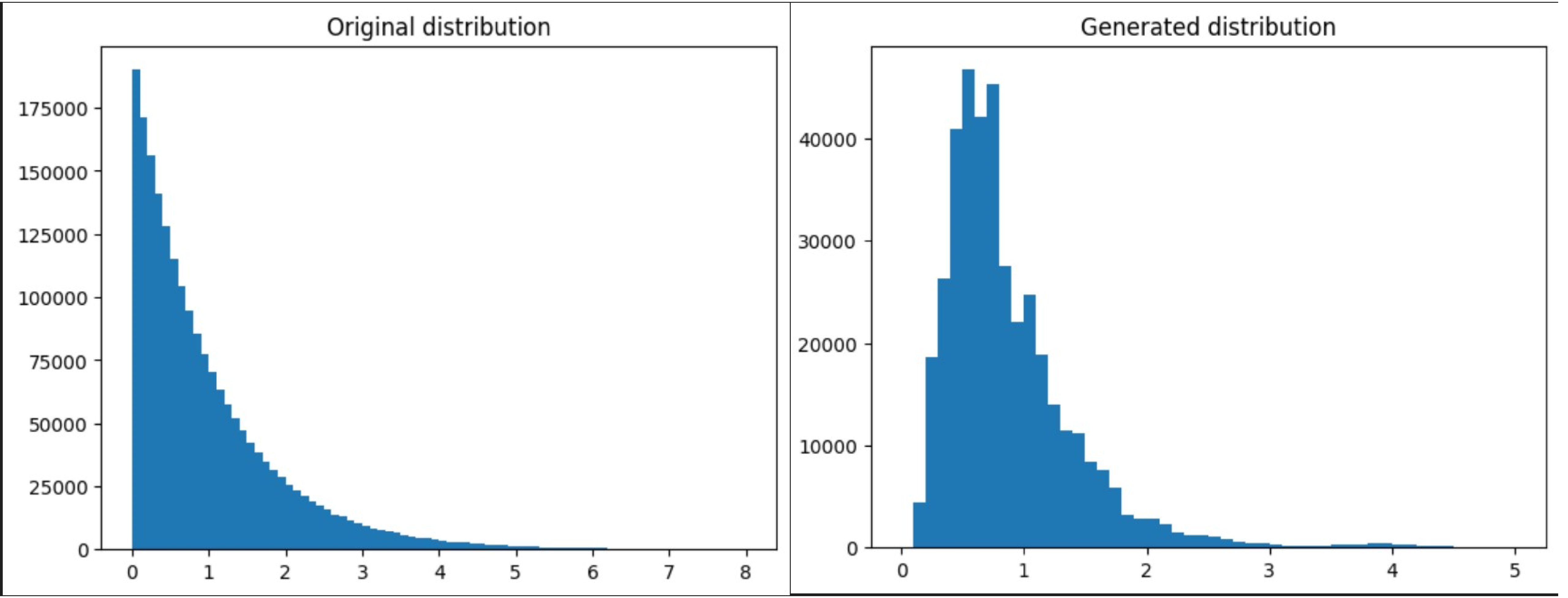
AND MANY MORE APPLICATIONS…….

RESULTS – GAUSSIAN



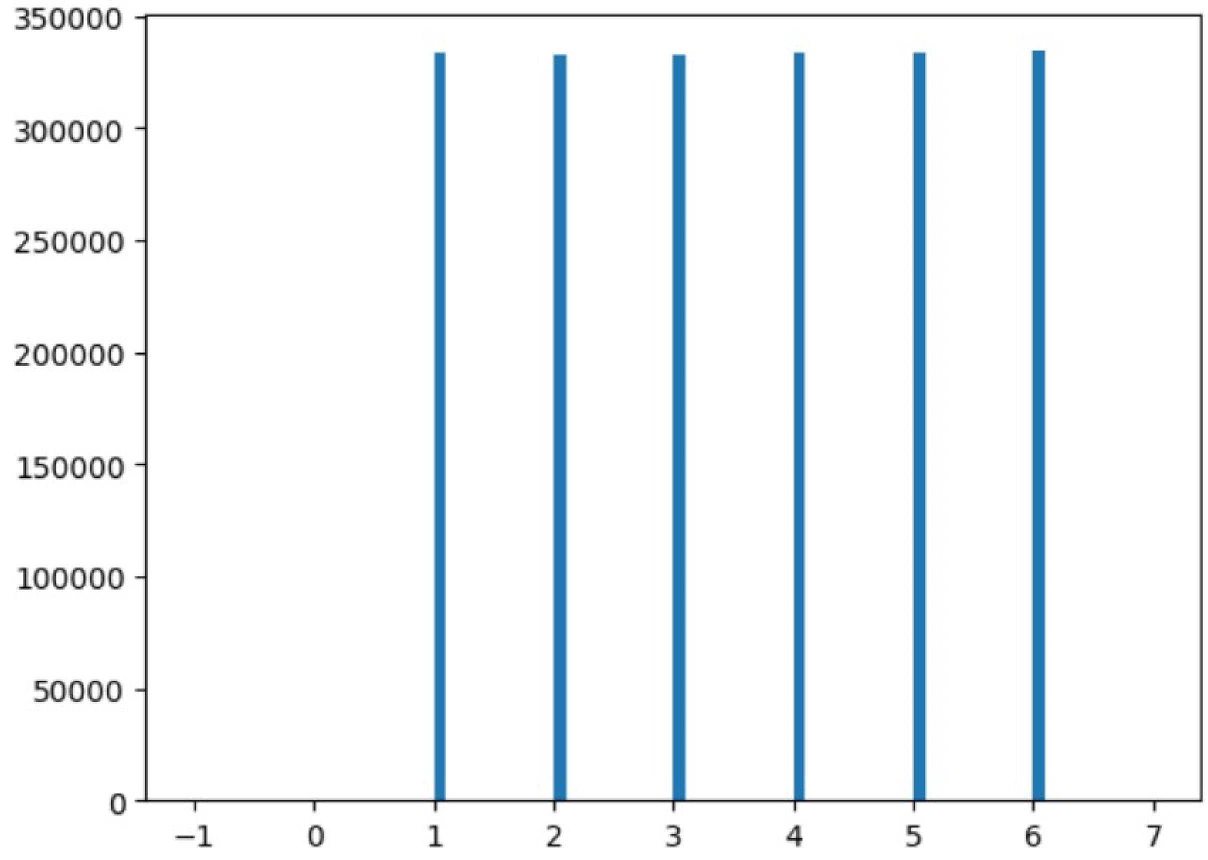


RESULTS – EXPONENTIAL RANDOM VARIABLE

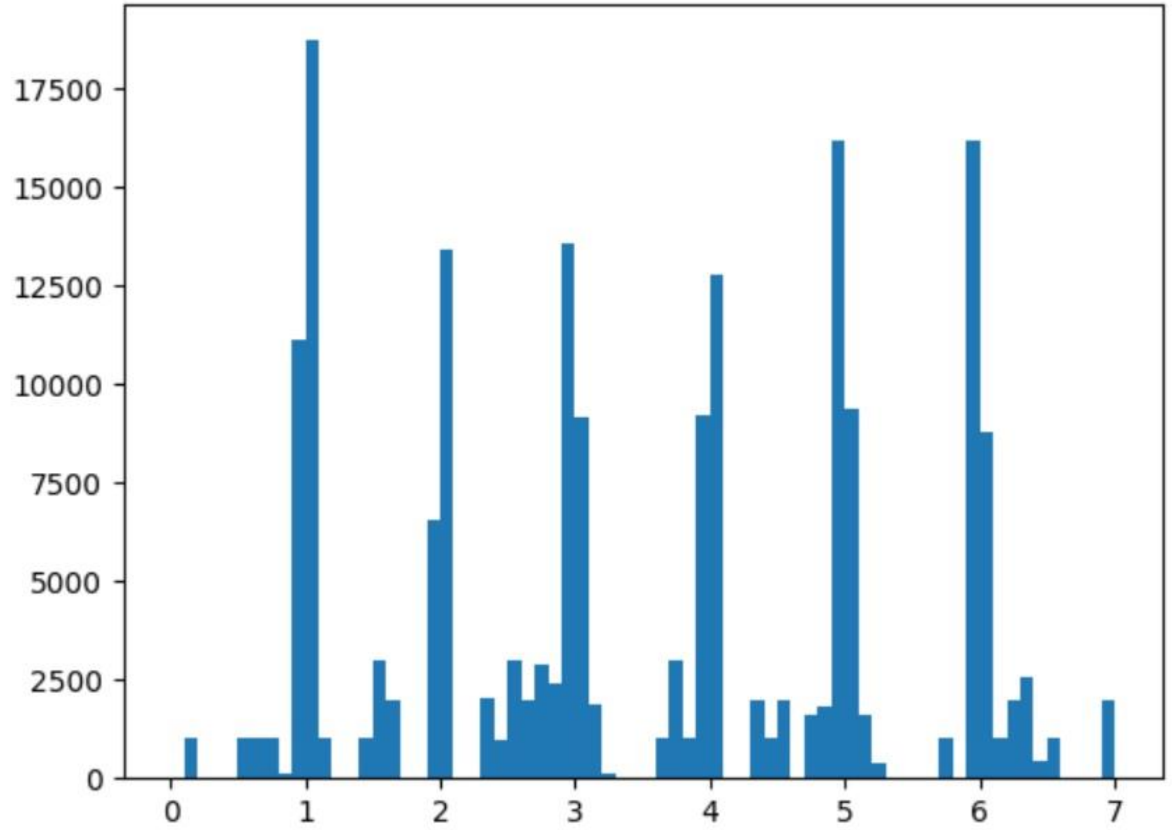


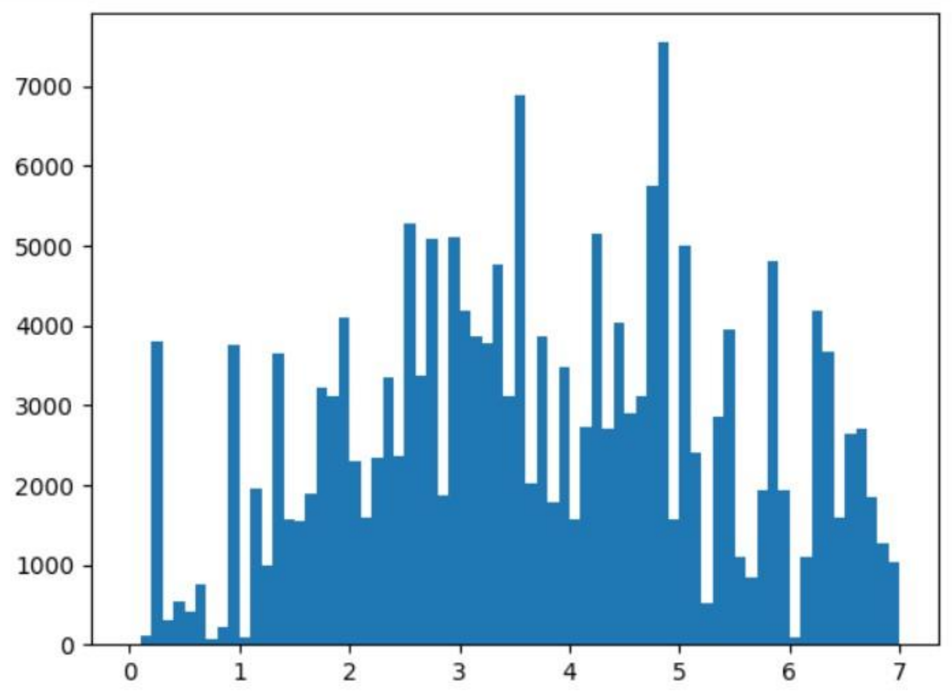
RESULTS – FOR A DICE ROLLING EXPERIMENT

ACTUAL DISTIBUTION



GENERATED DISTRIBUTION



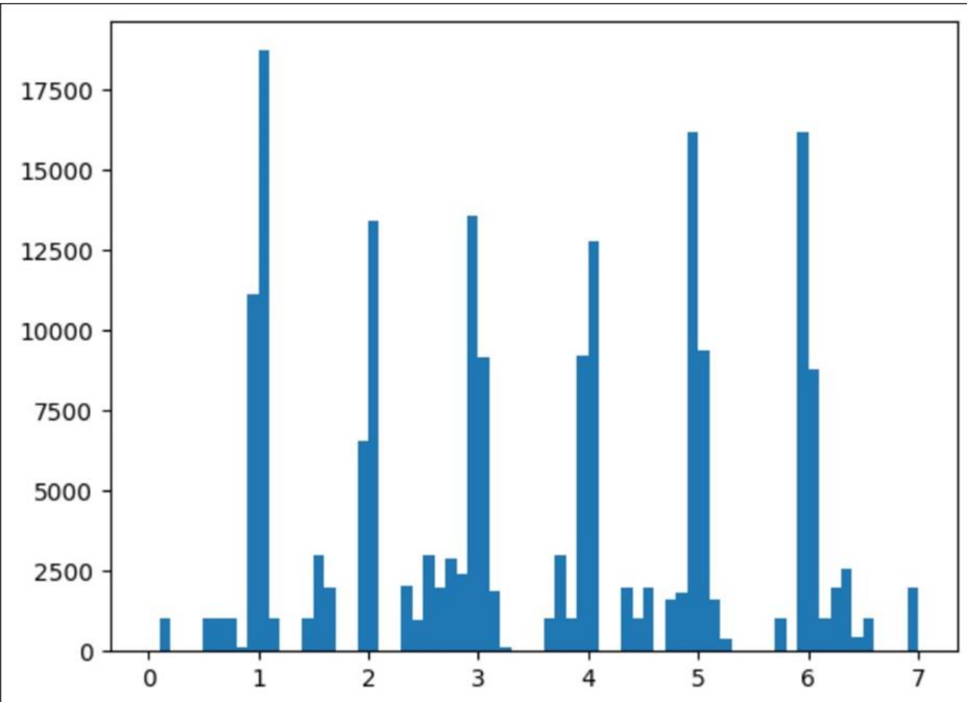


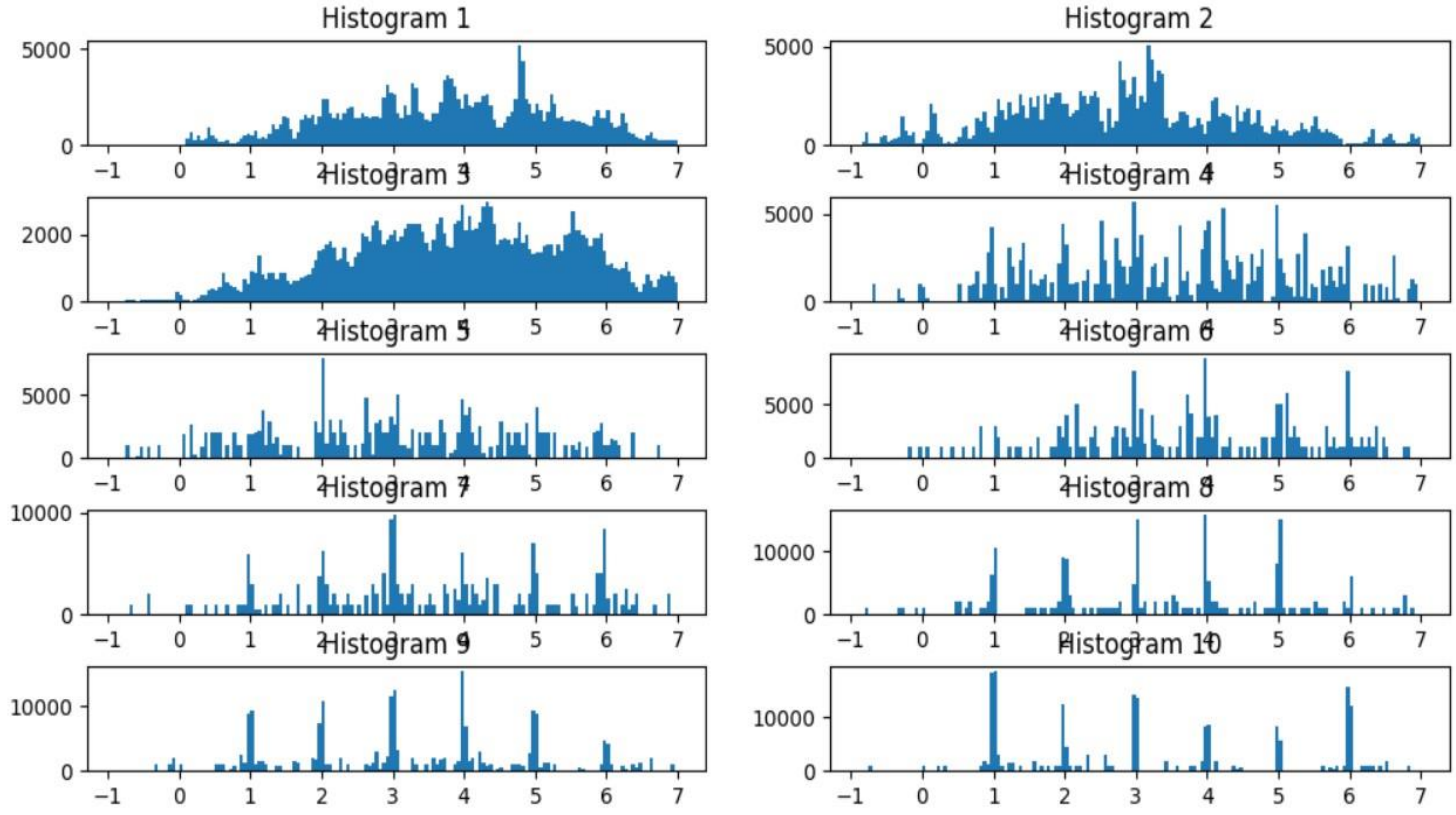
In this dice rolling experiment I have have 6 states, 1, 2,3 ,4 5,6. The above plot is when I generate a sequence using a normal GAN. You can see that it is continuous, and it is expected as we are dealing with a real numbers space. But do not want this.

So, I added an Integer Penalty term in the loss function, which penalizes the generator if the generated value is far from an integer. So, if basically, extra term added is:

$$Extra\ Loss = \sum penalty\ weight * (Integer\ distance\ of\ generated[i])$$

So, using this integer Penalty the GAN learns that it has to produce a number closer to the integer. So, I got the plot as shown below which is much more realistic.




Histogram 1
Histogram 2
Histogram 3
Histogram 4
Histogram 5
Histogram 6
Histogram 7
Histogram 8
Histogram 9
Histogram 10

RESULTS - 2 STATE MARKOV CHAIN

Original Transition Matrix:

	State 1	State 2
State 1	0.15	0.85
State 2	0.4	0.6

Generated Transition Matrix:

	State 1	State 2
State 1	0.156366	0.843634
State 2	0.400106	0.599894

RESULTS – 3 STATE DETERMINISTIC

```
1. 2. 3. 1. 2. 3. 1. 2. 3. 1. 2. 3. 1. 2. 3. 1. 2.
```

Generated Transition Matrix:

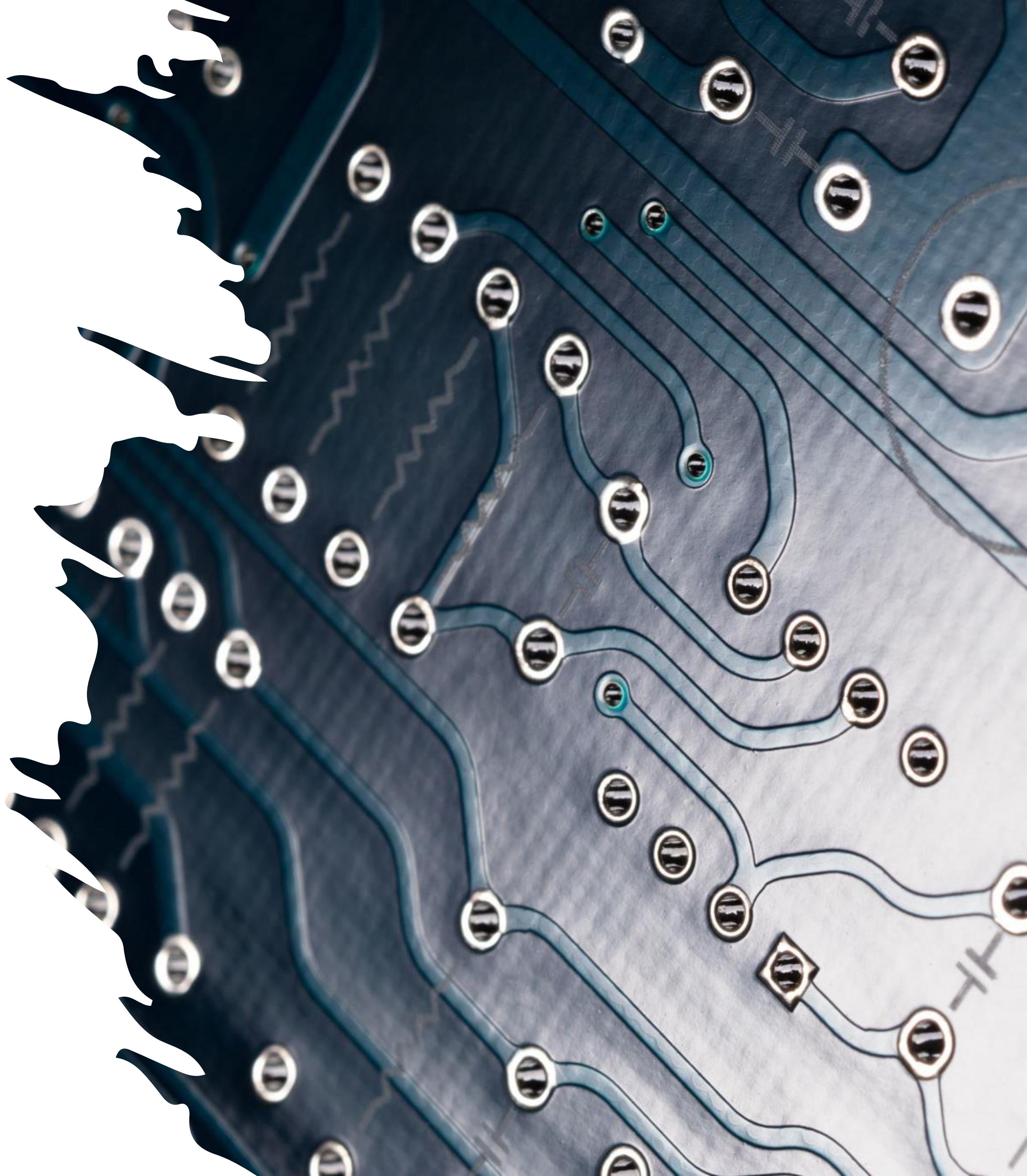
	State 1	State 2	State 3
State 1	0	1	0
State 2	0	0	1
State 3	1	0	0

Generated Transition Matrix:

	State 1	State 2	State 3
State 1	0	1	0
State 2	0	0	1
State 3	1	0	0

Conclusions

- So, we can conclude that GANs can be used in simulating continuous random variables, discrete random variables and also Markov chains.
- Till now a lot of applications have been built for GANs in the generation of images but now we can see that it can also be used to generate random time series data which can have lot of applications.




Thank you