Utrecht University

Stock Price Simulation under Jump-Diffusion Dynamics:

A WGAN-Based Framework with Anomaly Detection

Presented by: Renren Gan

Supervised by: Prof. dr. ir. C.W. Oosterlee
Dr. ir. L.A. Grzelak
N.T. Mücke

Master Thesis Presentation

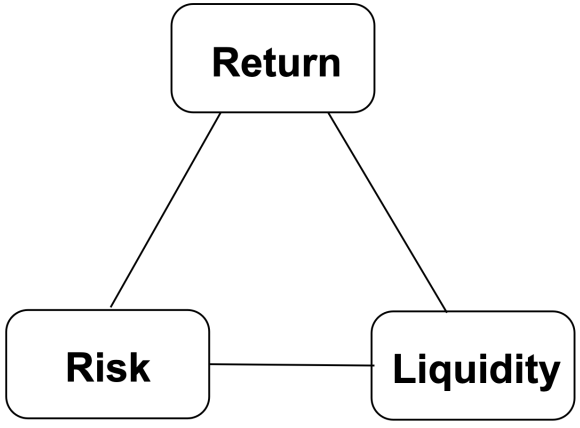
January 20, 2023

Outline

Motivation

Stock Price Simulation in Investment

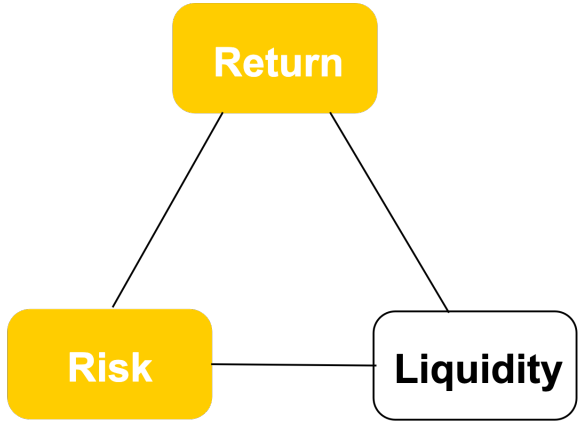
- The magic triangle of investment:



Motivation

Stock Price Simulation in Investment

- The magic triangle of investment:

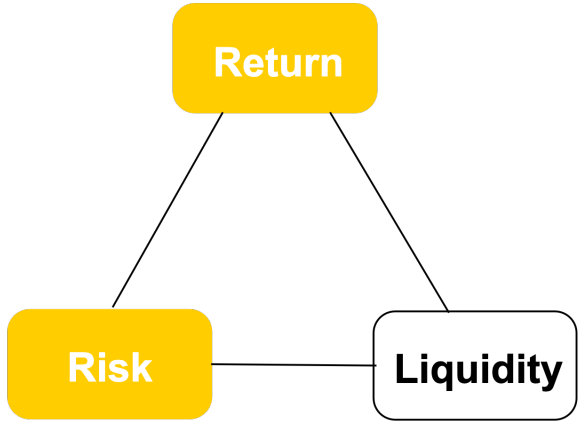


- Investors would like to have the highest possible return with acceptable risk.

Motivation

Stock Price Simulation in Investment

- The magic triangle of investment:

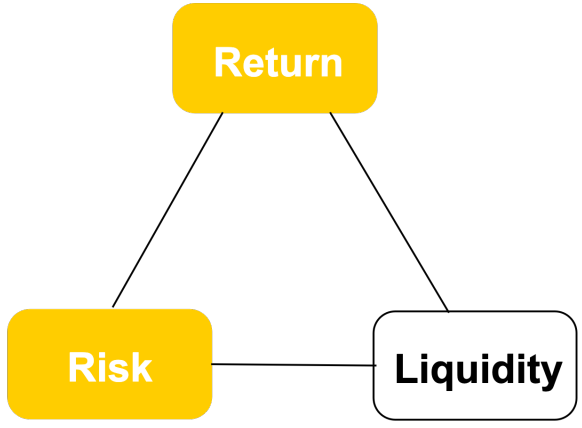


- Investors would like to have the highest possible return with acceptable risk.
- Stock is the basic asset class.

Motivation

Stock Price Simulation in Investment

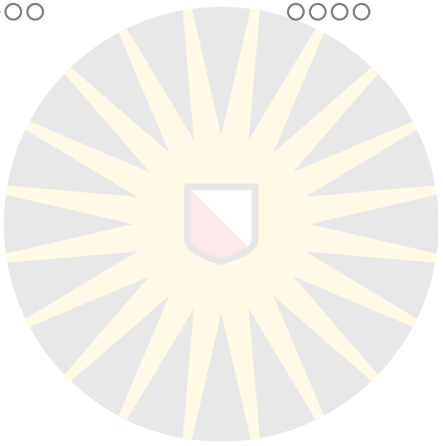
- The magic triangle of investment:



- Investors would like to have the highest possible return with acceptable risk.
- Stock is the basic asset class.
- $\Rightarrow$ We focus on the dynamics of the stock price.

Simulation Techniques

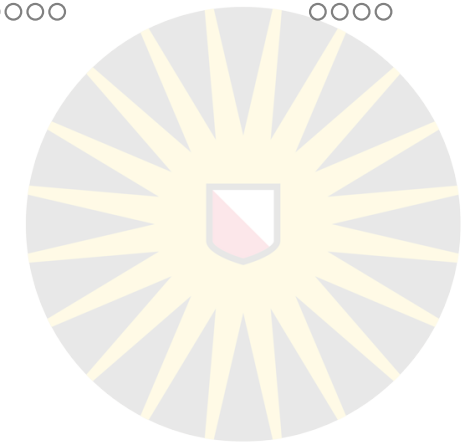
Stochastic Models



- Simulation model selection criteria:
  - ☑ realistic replica
  - ☑ tractable mathematical analysis

Simulation Techniques

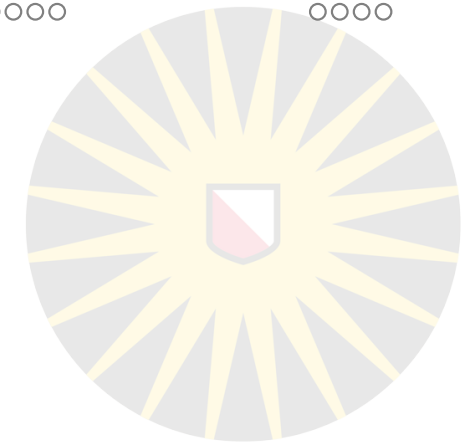
Stochastic Models



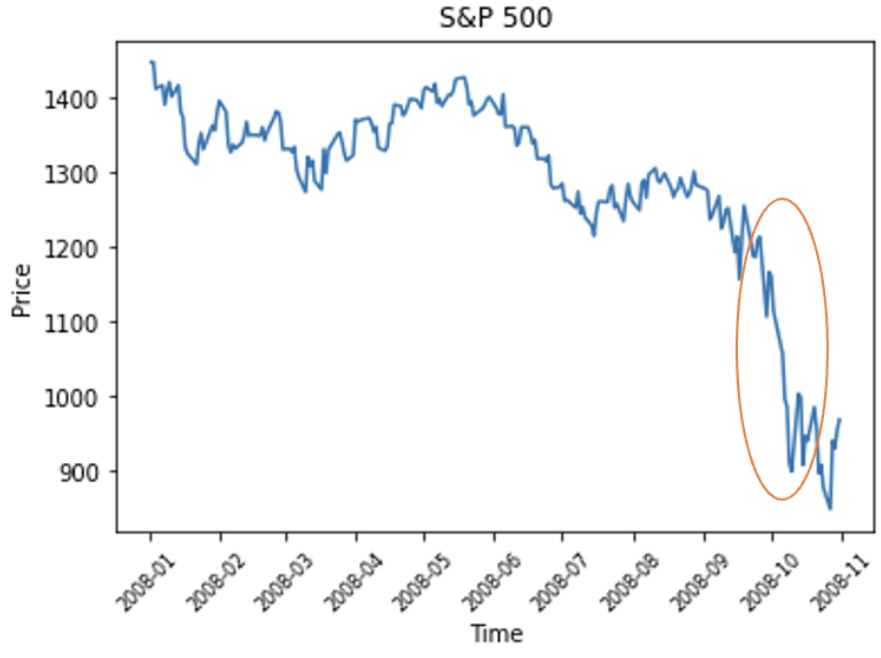
- Simulation model selection criteria:
  - ☑ realistic replica
  - ☑ tractable mathematical analysis
- $\Rightarrow$ Stochastic Markov process
  - $\Rightarrow$ Jump-diffusion process

Simulation Techniques

Why Jump-Diffusion



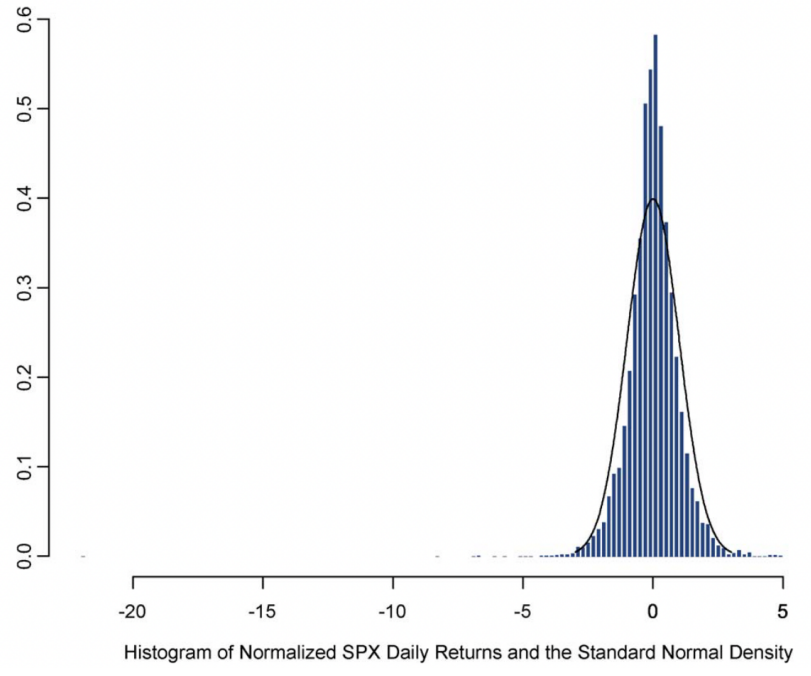
- Stock prices do jump, e.g., during a financial crisis.



Simulation Techniques

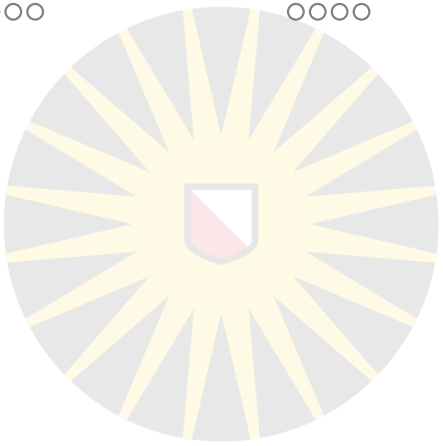
Why Jump-Diffusion

- Stock prices do jump, e.g., during a financial crisis.
- Geometric Brownian motion v.s. jump-diffusion:
  The leptokurtic feature.



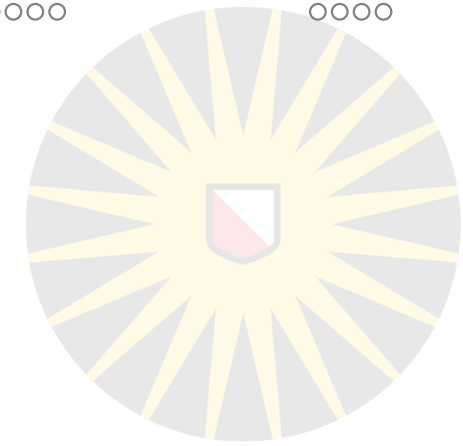
Simulation Techniques

Why Jump-Diffusion



- Stock prices do jump, e.g., during a financial crisis.
- Geometric Brownian motion v.s. jump-diffusion:
  The leptokurtic feature.
- Jump-diffusion v.s. other jump processes: simple.

Jump-Diffusion Model



- The general log-price dynamics under $\mathbb{P}$-measure:

Dynamics

$$\mathrm{d}X_J(t) = \mu\ \mathrm{d}t + \sigma\ \mathrm{d}W^{\mathbb{P}}(t) + J\, X^{\mathbb{P}}_{\mathcal{P}}(t), \text{ with } X_J(0) = \log S_J(0).$$

- where $X_J(t) = \log S_J(t)$ is the log-stock price, μ is the drift, σ is the volatility, $W(t)$ is a Wiener process, $X_{\mathcal{P}}(t)$ is a Poisson process with parameter λ_p, and J is a random variable giving the jump magnitude information.
- *Remark: $W(t)$ and $X_{\mathcal{P}}(t)$ are often independent.*

Jump-Diffusion Model

- Merton's model: $J \sim \mathcal{N}(\mu_J, \sigma_J^2)$.

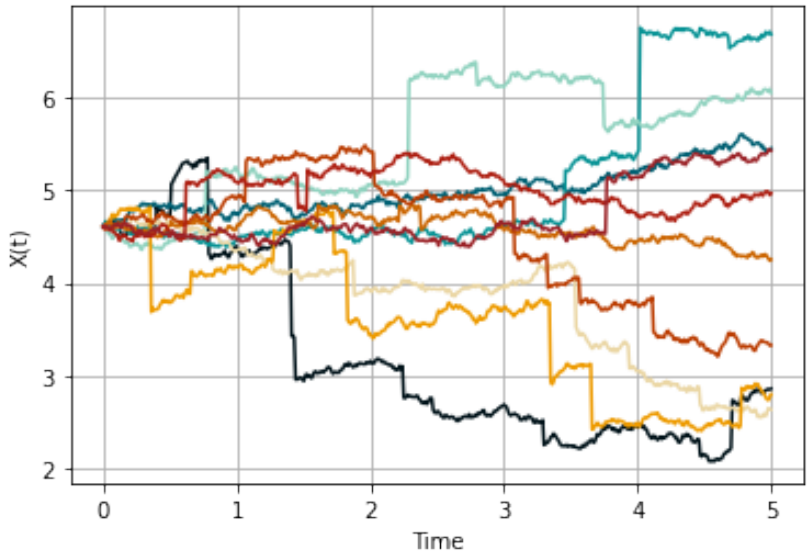
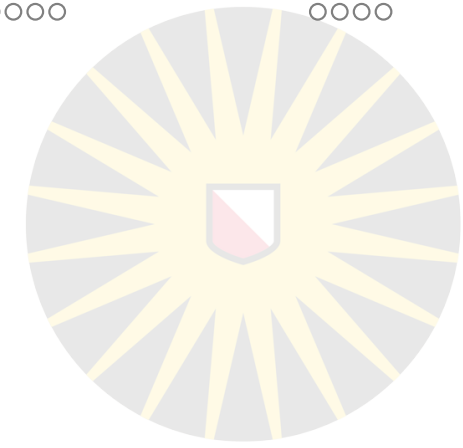


Figure: Paths of Merton's model, where $\mu = 0.05$, $\sigma = 0.2$, $S_J(0) = 100$, $\lambda_p = 1$, $J \sim \mathcal{N}(0, 0.5^2)$, $T = 5$.

Jump-Diffusion Model



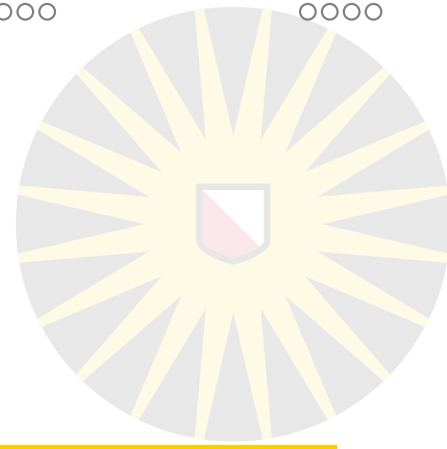
Dynamics

$$\mathrm{d}X_J(t) = \mu\ \mathrm{d}t + \sigma\ \mathrm{d}W^{\mathbb{P}}(t) + J\ X^{\mathbb{P}}_{\mathcal{P}}(t), \text{ with } X_J(0) = \log S_J(0).$$

- The solution is given as:

$$X_J(T) = X_J(0) + \mu T + \int_0^T \sigma\ \mathrm{d}W^{\mathbb{P}}(t) + \sum_{k=1}^{X^{\mathbb{P}}_{\mathcal{P}}(T)} J_k.$$

Jump-Diffusion Model



Dynamics

$$\mathrm{d}X_J(t) = \mu\ \mathrm{d}t + \sigma\ \mathrm{d}W^{\mathbb{P}}(t) + J\ X^{\mathbb{P}}_{\mathcal{P}}(t), \text{ with } X_J(0) = \log S_J(0).$$

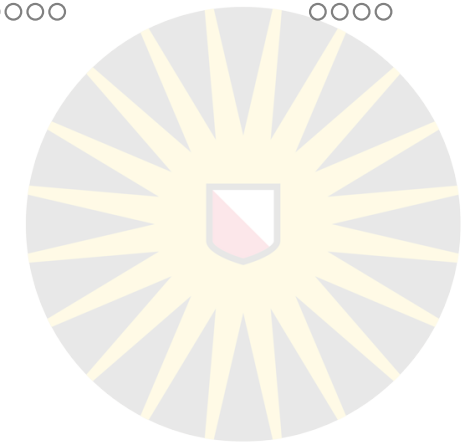
- The solution is given as:

$$X_J(T) = X_J(0) + \mu T + \int_0^T \sigma\ \mathrm{d}W^{\mathbb{P}}(t) + \sum_{k=1}^{X^{\mathbb{P}}_{\mathcal{P}}(T)} J_k.$$

- Numerical approximation: Monte Carlo simulation.

Path Simulation

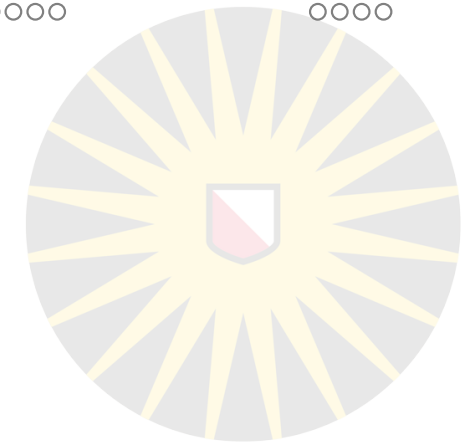
Jump-Diffusion Model



- $\mathrm{d}X_J(t) = \underbrace{\mu\ \mathrm{d}t + \sigma\ \mathrm{d}W^{\mathbb{P}}(t)}_{\text{The diffusion part } X(t)} + \underbrace{J\ X^{\mathbb{P}}_{\mathcal{P}}(t)}_{\text{The jump part } J(t)}$.

Path Simulation

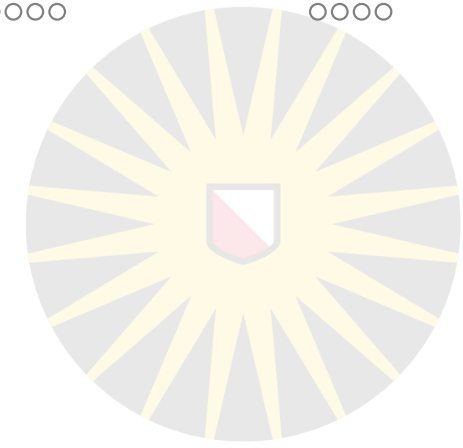
Jump-Diffusion Model



- $\mathrm{d}X_J(t) = \underbrace{\mu\ \mathrm{d}t + \sigma\ \mathrm{d}W^{\mathbb{P}}(t)}_{\text{The diffusion part } X(t)} + \underbrace{J\, X^{\mathbb{P}}_{\mathcal{P}}(t)}_{\text{The jump part } J(t)}$.
- Time discretization with Euler scheme:
  $X_J(t+\Delta t) \approx$
  $$X_J(t) + \mu\Delta t + \sigma\left(W^{\mathbb{P}}(t+\Delta t) - W^{\mathbb{P}}(t)\right) + \sum_{k=1}^{X^{\mathbb{P}}_{\mathcal{P}}(t+\Delta t) - X^{\mathbb{P}}_{\mathcal{P}}(t)} J_k$$
  - Wiener increment $W^{\mathbb{P}}(t+\Delta t) - W^{\mathbb{P}}(t) \sim \mathcal{N}(0, \Delta t)$;
  - Poisson increment $X^{\mathbb{P}}_{\mathcal{P}}(t+\Delta t) - X^{\mathbb{P}}_{\mathcal{P}}(t) \sim$ Pois $(\lambda_p \Delta t)$.

Path Simulation

Jump-Diffusion Model

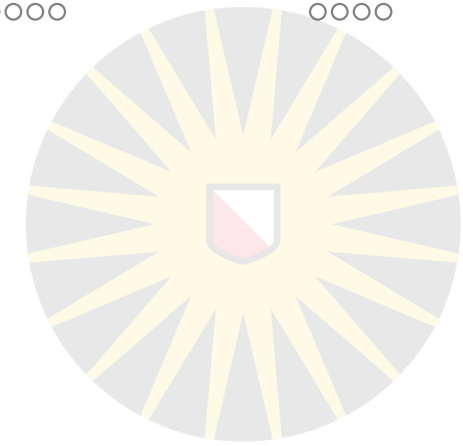


- The diffusion increment:

$$X(t+\Delta t) \overset{d}{=} X(t) + \mu\Delta t + \sigma\sqrt{\Delta t}Z, \text{ where } Z \sim \mathcal{N}(0,1).$$

Path Simulation

Jump-Diffusion Model



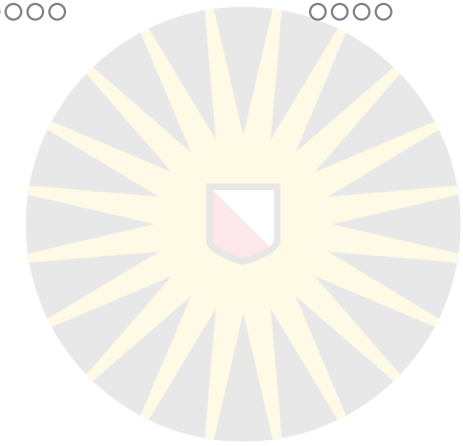
- The jump increment:

$$\mathbb{P}\left[X^{\mathbb{P}}_{\mathcal{P}}(t+\Delta t) - X^{\mathbb{P}}_{\mathcal{P}}(t) = k\right] = \frac{(\lambda_p \Delta t)^k e^{-\lambda_p \Delta t}}{k!}.$$

- $\mathbb{P}\left[X^{\mathbb{P}}_{\mathcal{P}}(t+\Delta t) - X^{\mathbb{P}}_{\mathcal{P}}(t) = 1\right] = \lambda_p \Delta t + o(\Delta t)$

Path Simulation

Jump-Diffusion Model



- The jump increment:
  $\Rightarrow$ Use a Bernoulli variable $B(t) \sim$ Bin $(1, \lambda_p \Delta t)$ to decide the jump occurrence at $t + \Delta t$:

$$\mathbb{P}\left[B(t) = 1\right] = \lambda_p \Delta t,$$
$$\mathbb{P}\left[B(t) = 0\right] = 1 - \lambda_p \Delta t$$

  $\Rightarrow J(t + \Delta t) \overset{d}{=} J(t) + \mathrm{J}\, B(t)$, where $\mathrm{J} \sim \mathcal{N}(\mu_J, \sigma_J^2)$ in Merton's model.

Path Simulation

Jump-Diffusion Model

- Bernoulli v.s. Poisson

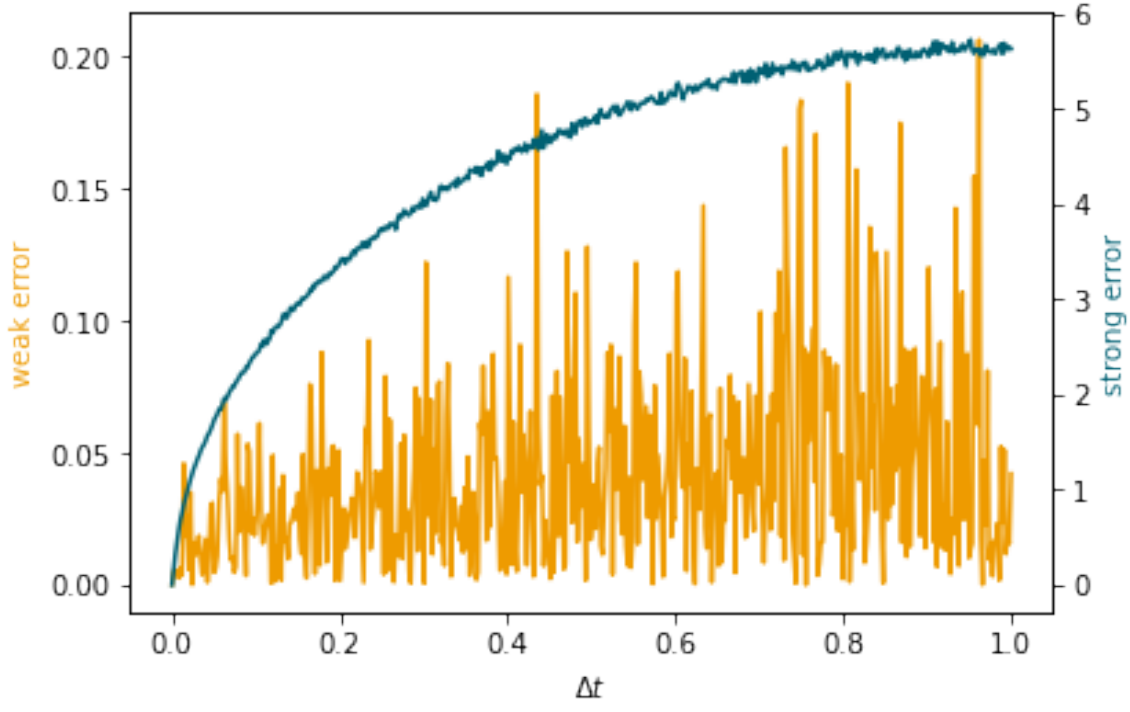
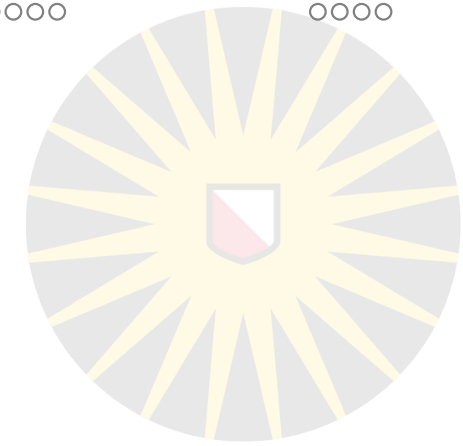


Figure: The error convergence plot for a Bernoulli process with $\lambda_p = 1$, compared to a Poisson process with the same parameters.

Path Simulation

Jump-Diffusion Model



- The diffusion increment:

$$X(t+\Delta t) \overset{d}{=} X(t) + \mu\Delta t + \sigma\sqrt{\Delta t}Z, \text{ where } Z \sim \mathcal{N}(0,1).$$

- The jump increment:

$$J(t+\Delta t) \overset{d}{=} J(t) + \mathrm{J}\, B(t).$$

- Jump-diffusion simulation:

$$X_J(t+\Delta t) \overset{d}{=} X_J(t) + \mu\Delta t + \sigma\sqrt{\Delta t}Z + \mathrm{J}B(t).$$

Path Simulation

Jump-Diffusion Model

- Jump-diffusion simulation:

$$X_J(t+\Delta t) \overset{d}{=} X_J(t) + \mu\Delta t + \sigma\sqrt{\Delta t}Z + \mathrm{J}B(t).$$

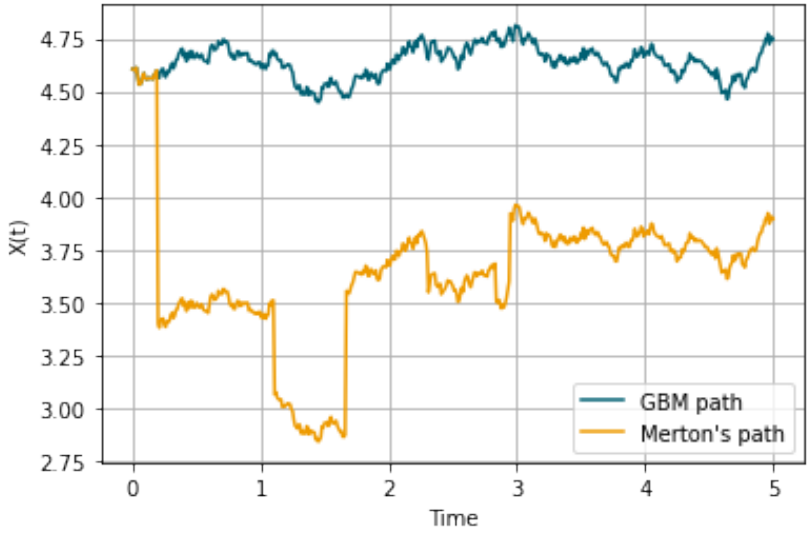
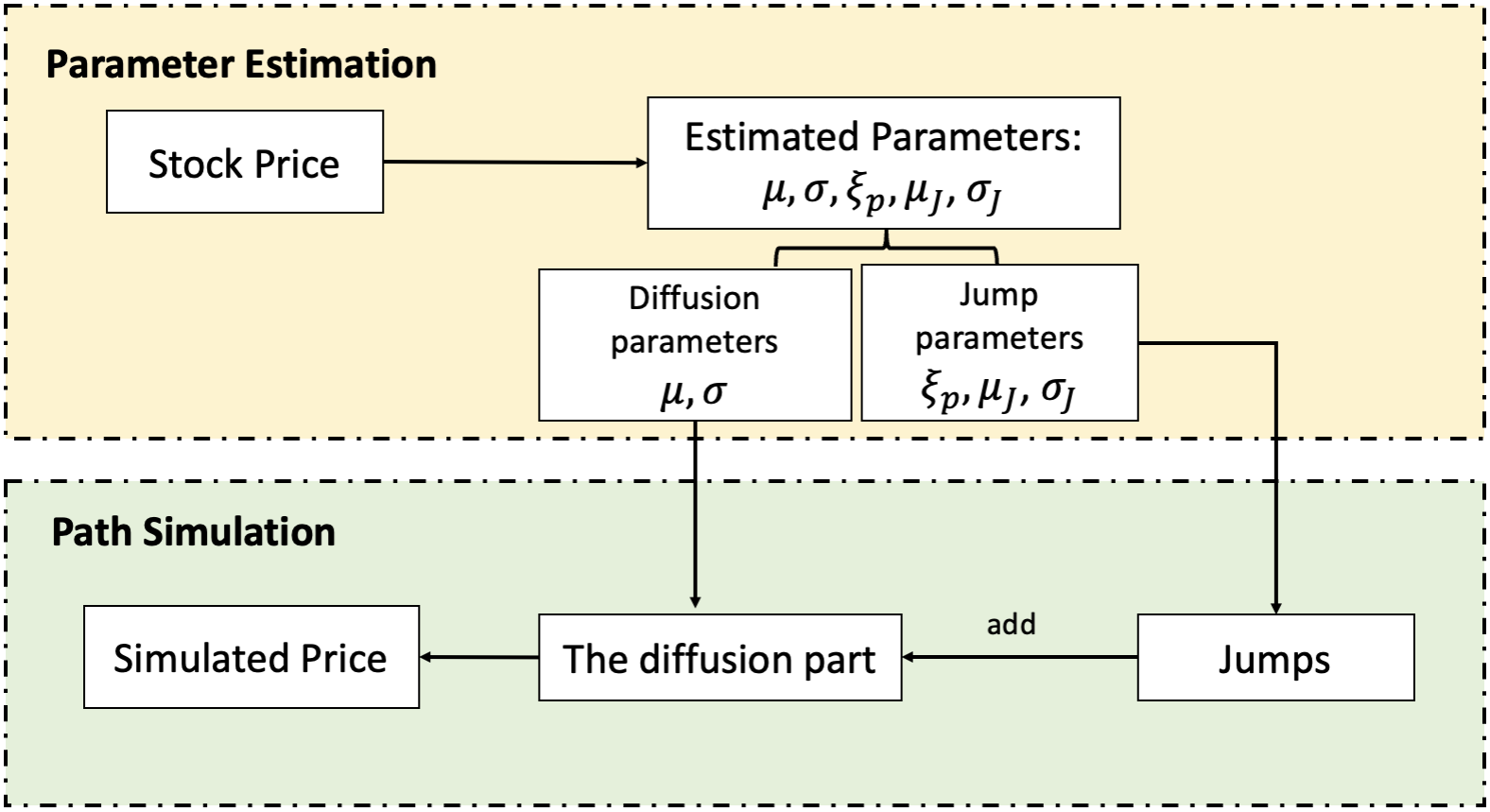


Figure: Paths of GBM and Merton's model, where $\mu = 0.05$, $\sigma = 0.2$, $S(0) = 100$, $\lambda_p = 1$, $J \sim \mathcal{N}(0, 0.5^2)$, $T = 5$.

Goal

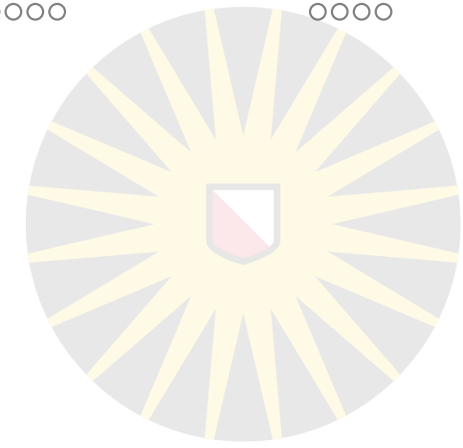
Jump-Diffusion Simulation

We aim to model empirical stock paths under the Merton's jump-diffusion dynamics: $\mathrm{d}X(t) = \mu\ \mathrm{d}t + \sigma\ \mathrm{d}W^{\mathbb{P}}(t) + \mathrm{J}X^{\mathbb{P}}_{\mathcal{P}}(t)$, here $\mathrm{J} \sim \mathcal{N}(\mu_J, \sigma_J^2)$.



Non-Machine Learning Method

Jump-Diffusion Simulation

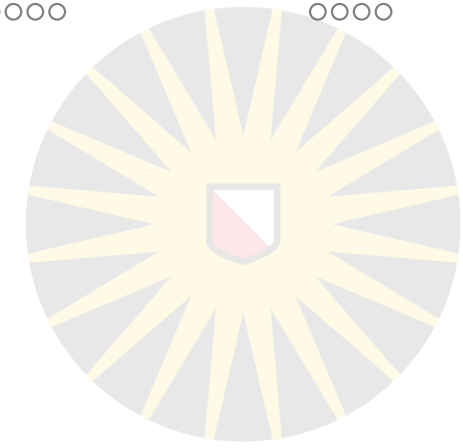


- Parameter estimation: Maximum Likelihood Estimation (MLE)
- Path simulation: Euler discretization

$$X_J(t+\Delta t) \overset{d}{=} X_J(t) + \mu\Delta t + \sigma\sqrt{\Delta t}Z + \mathrm{J}B(t)$$

Machine Learning-Based Approach

SDE-GAN



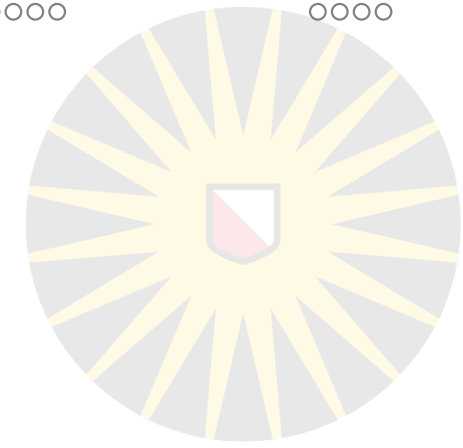
Monte Carlo Simulation of SDEs using GANs

Jorino van Rhijn[a,*], Cornelis W. Oosterlee[b], Lech A. Grzelak[c,d], Shuaiqiang Liu[a]

[a] *Centrum Wiskunde en Informatica (CWI), Amsterdam, Netherlands*
[b] *Mathematical Institute, Utrecht University, Utrecht, Netherlands*
[c] *Delft University of Technology, Delft, Netherlands*
[d] *Rabobank, Utrecht, Netherlands*

Machine Learning-Based Approach

SDE-GAN

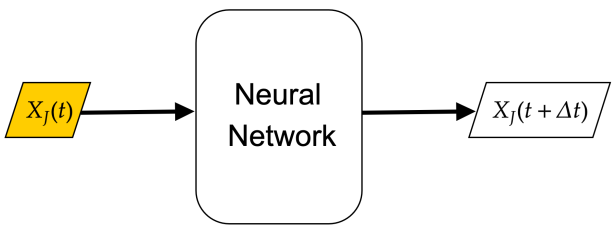


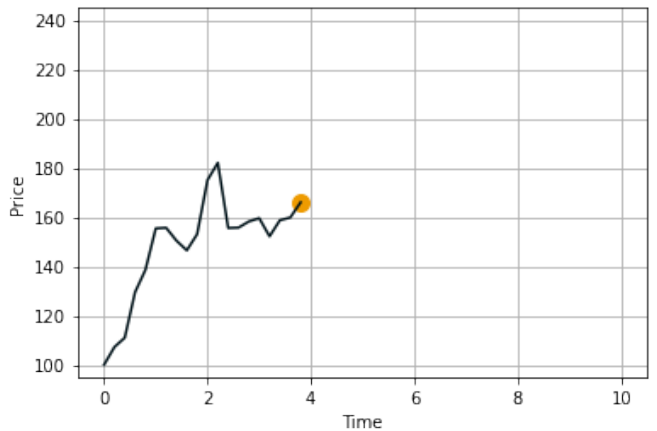
- It is a machine learning-based method for the approximation of the Itô SDEs.
- The jump-diffusion process is a Markov process:
  $\Rightarrow$ The future price $X_J(t + \Delta t)$ depends only on the current state $X_J(t)$ and does not rely on the past.

Machine Learning-Based Approach

SDE-GAN

- A neural network, especially a GAN, is designed to learn the pairing of $X_J(t)$ and $X_J(t+\Delta t)$, namely, the conditional distribution $\mathbb{P}_{X_J(t+\Delta t)|X_J(t)}$.
- A path is constructed by iteratively sampling the future state from the distribution $\mathbb{P}_{X_J(t+\Delta t)|X_J(t)}$.

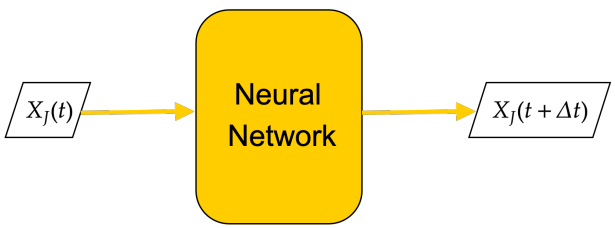


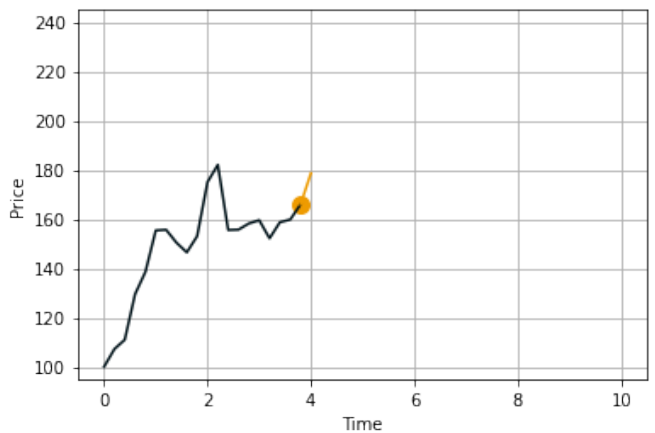


Machine Learning-Based Approach

SDE-GAN

- A neural network, especially a GAN, is designed to learn the pairing of $X_J(t)$ and $X_J(t+\Delta t)$, namely, the conditional distribution $\mathbb{P}_{X_J(t+\Delta t)|X_J(t)}$.
- A path is constructed by iteratively sampling the future state from the distribution $\mathbb{P}_{X_J(t+\Delta t)|X_J(t)}$.

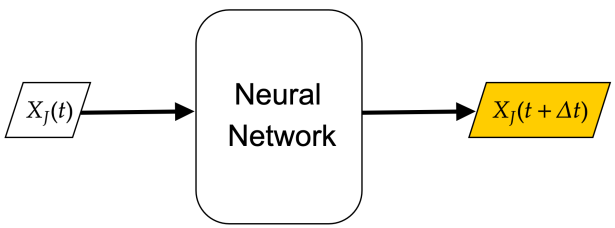


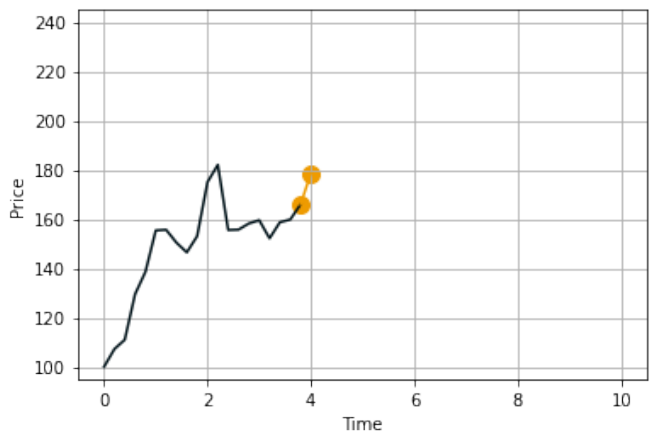


Machine Learning-Based Approach

SDE-GAN

- A neural network, especially a GAN, is designed to learn the pairing of $X_J(t)$ and $X_J(t+\Delta t)$, namely, the conditional distribution $\mathbb{P}_{X_J(t+\Delta t)|X_J(t)}$.
- A path is constructed by iteratively sampling the future state from the distribution $\mathbb{P}_{X_J(t+\Delta t)|X_J(t)}$.

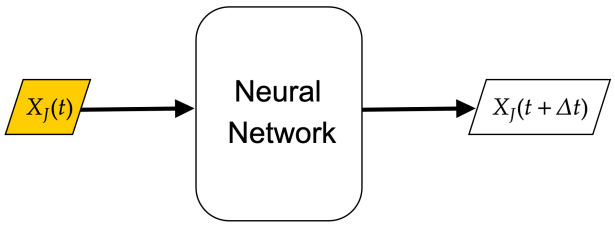


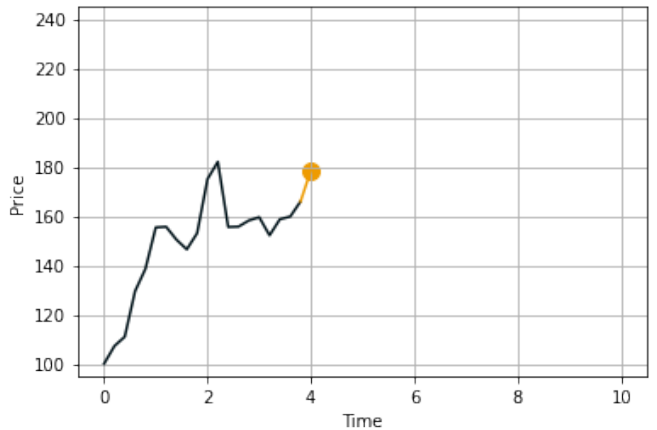


Machine Learning-Based Approach

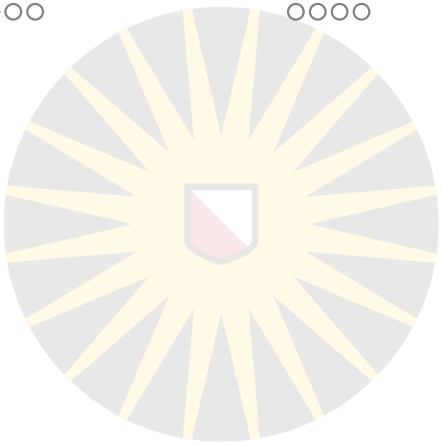
SDE-GAN

- A neural network, especially a GAN, is designed to learn the pairing of $X_J(t)$ and $X_J(t+\Delta t)$, namely, the conditional distribution $\mathbb{P}_{X_J(t+\Delta t)|X_J(t)}$.
- A path is constructed by iteratively sampling the future state from the distribution $\mathbb{P}_{X_J(t+\Delta t)|X_J(t)}$.





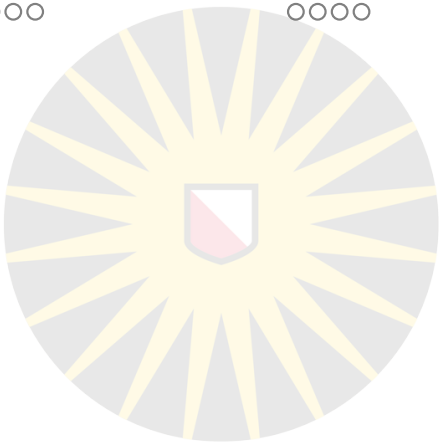
Machine Learning-Based Approach



The advantages of using the SDE-GAN:

- It gives a general pattern of simulating Itô SDEs;
- Only the prices are required for simulation;
- It offers potential benefits when dealing with higher-dimensional problems.

Machine Learning-Based Approach



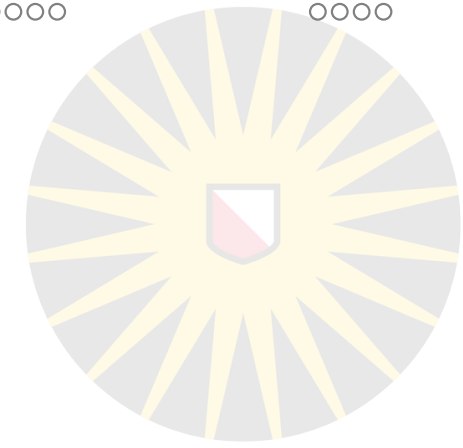
The advantages of using the SDE-GAN:

- It gives a general pattern of simulating Itô SDEs;
- Only the prices are required for simulation;
- It offers potential benefits when dealing with higher-dimensional problems.

Application: The SDE-GAN can be used to simulate the diffusion part.

Question: Is it possible to use a machine learning-based method for the jump part?

Machine Learning-Based Approach



The advantages of using the SDE-GAN:

- It gives a general pattern of simulating Itô SDEs;
- Only the prices are required for simulation;
- It offers potential benefits when dealing with higher-dimensional problems.

Application: The SDE-GAN can be used to simulate the diffusion part.

Question: Is it possible to use a machine learning-based method for the jump part?

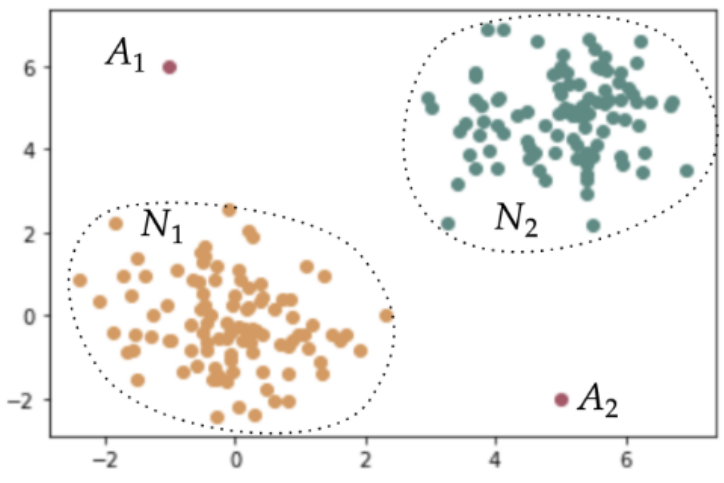
$\Rightarrow$ **Idea:** Use anomaly detection techniques.

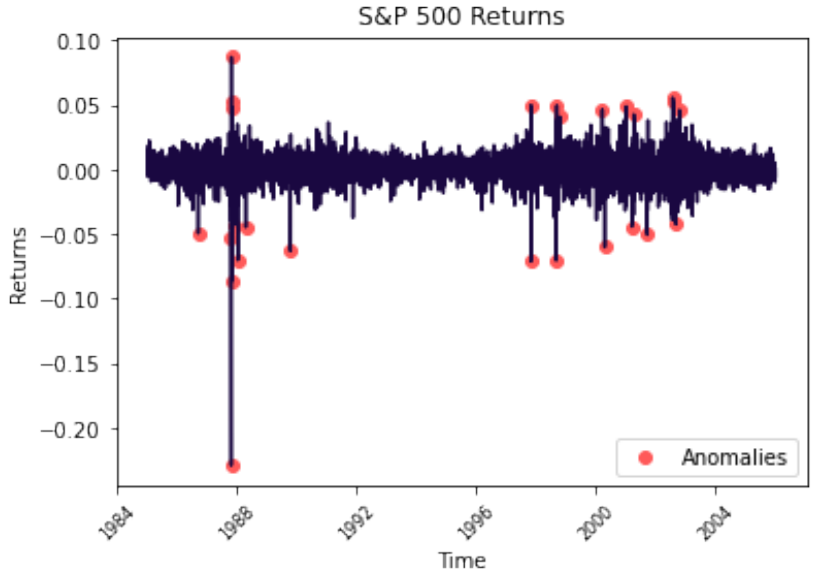
Anomaly Detection

What are anomalies?

Anomalies or Rare events

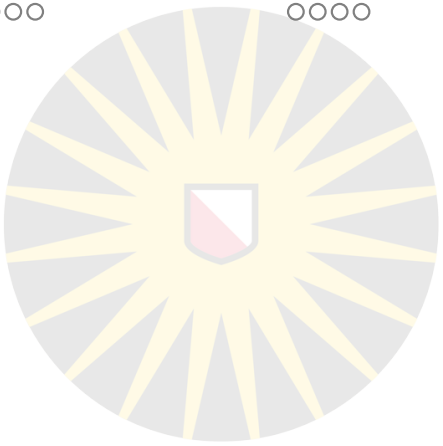
Anomalies are patterns in data that do not conform to a well defined notion of normal behavior.





Anomaly Detection

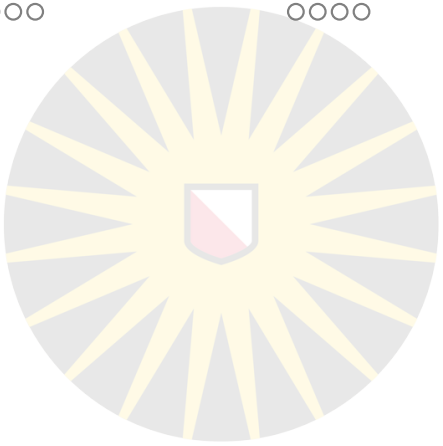
Methodology



- Anomaly detection techniques:
  - Classification-based;
  - Distance-based;
  - Clustering-based;
  - Reconstruction-based;
  - ...

Anomaly Detection

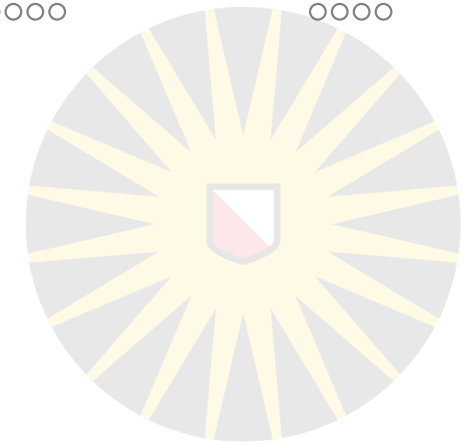
Methodology



- Two main tasks of anomaly detection:
  - "Learn" the normal pattern;
  - Measure the anomalousness or calculate the anomaly score.

Anomaly Detection

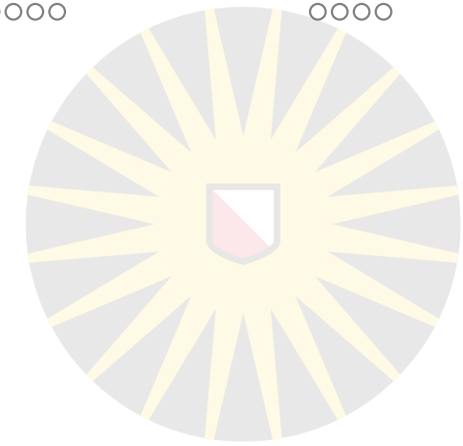
Methodology



- Two main tasks of anomaly detection:
  - "Learn" the normal pattern;
  - Measure the anomalousness or calculate the anomaly score.
- Example: reconstruction-based anomaly detection
  - Usually a neural network is built to reconstruct the data;
  - The neural network is trained on the normal data
    $\Rightarrow$ It can only approximate the normal pattern.
  - The anomalousness is evaluated based on the difference between the reconstructed pattern and its actual pattern.
  - One sample is abnormal if its anomaly score is larger than the threshold.

Anomaly Detection

Methodology



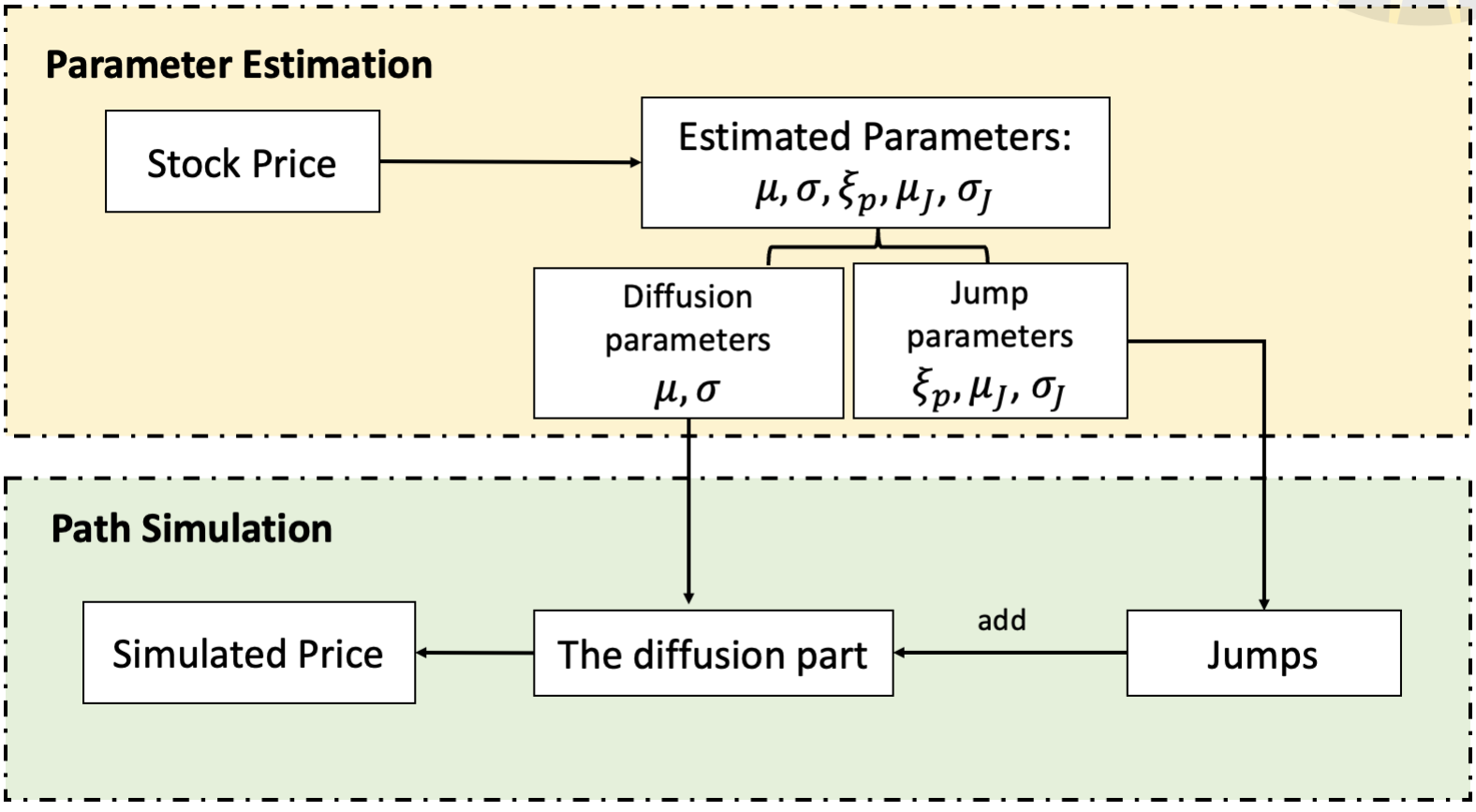
- Two main tasks of anomaly detection:
  - "Learn" the normal pattern;
  - Measure the anomalousness or calculate the anomaly score.

Ideas:

- We can view the jumps as anomalies, while the non-jump prices are the normal data.
- The jump instances of a jump-diffusion path can be detected via a machine learning-based anomaly detection method.
- The jump parameters can be further estimated based on the detected jump instances.

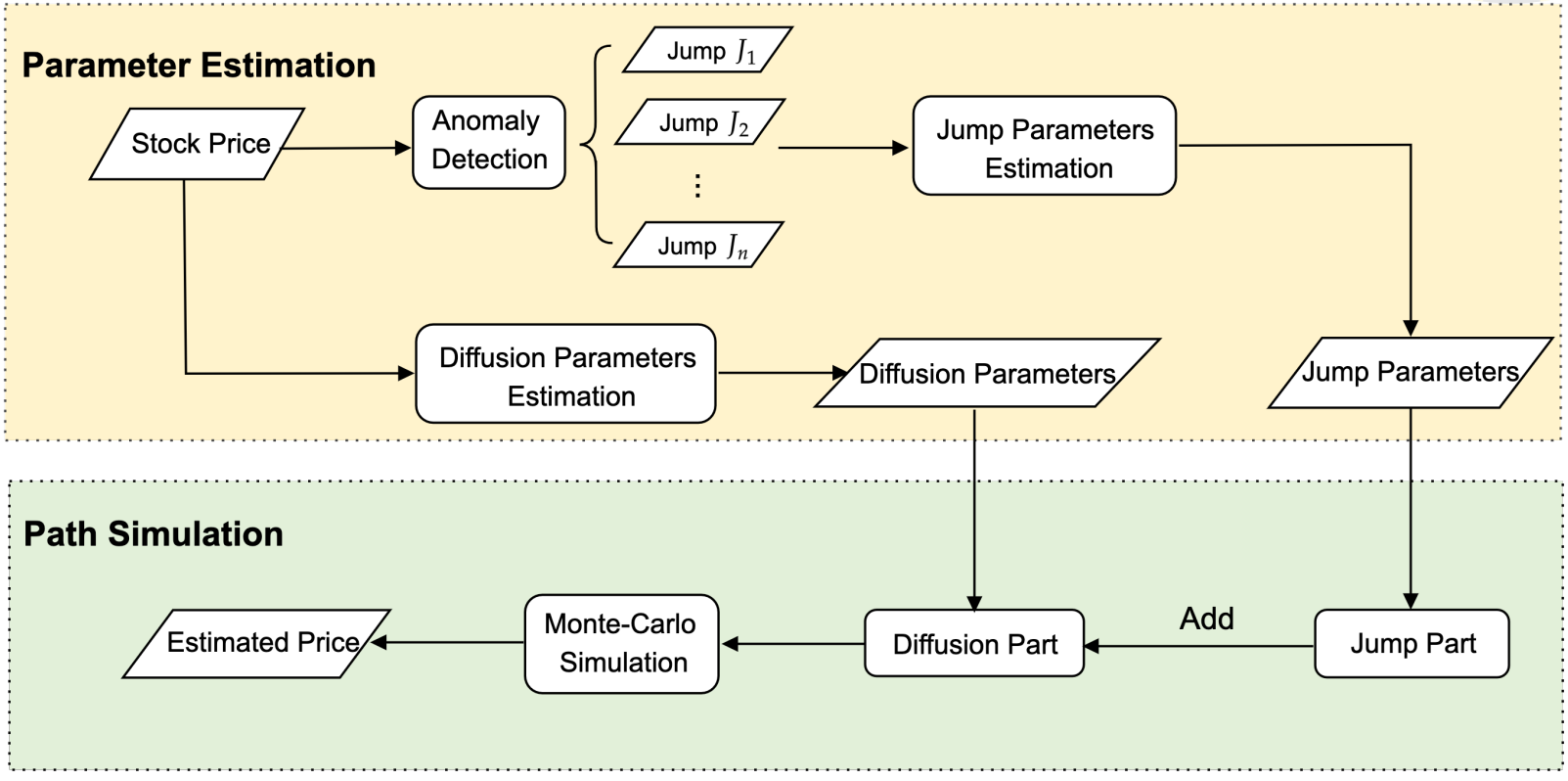
Idea

A Machine Learning-Based Jump-Diffusion Simulation



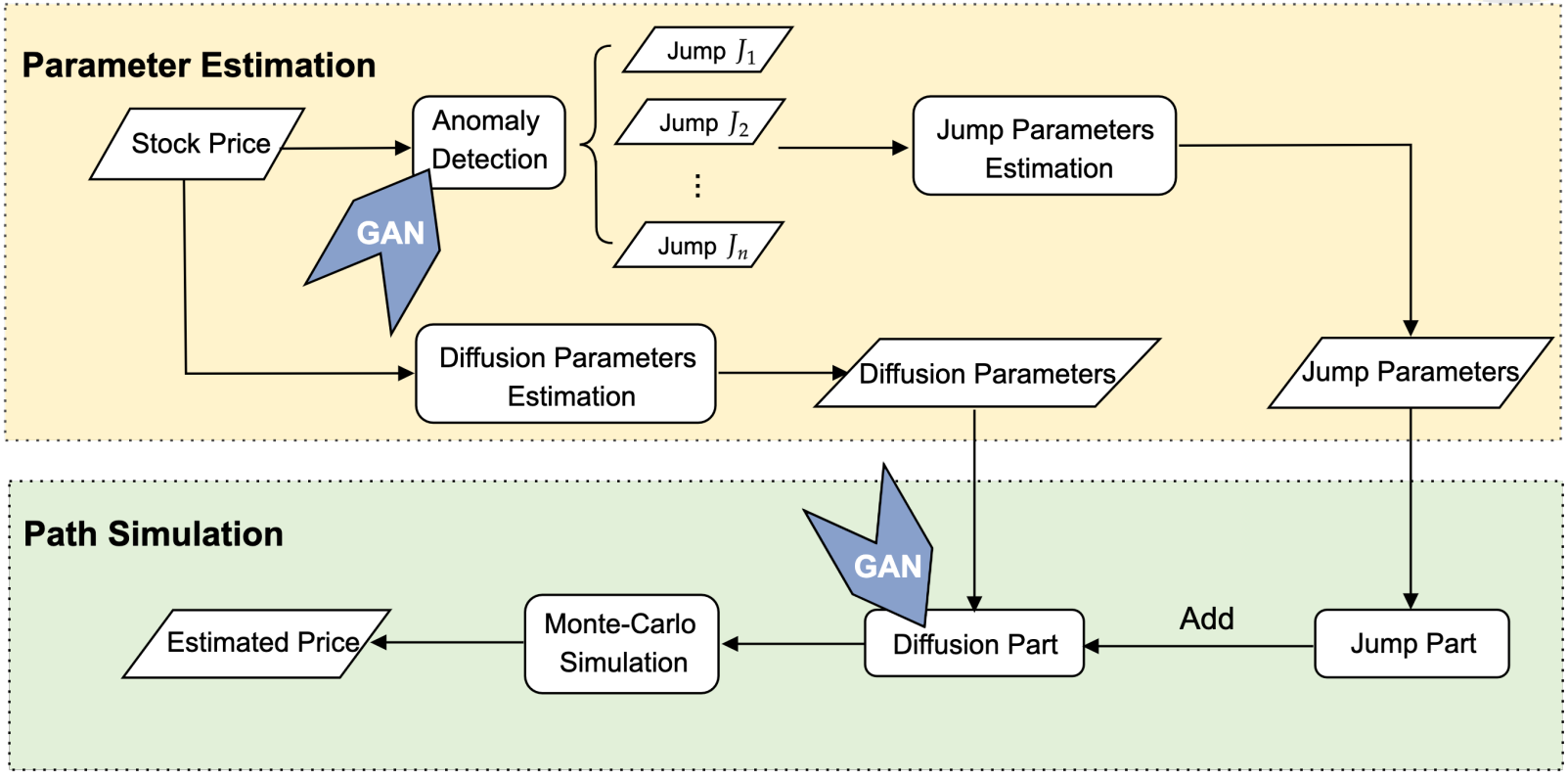
Idea

A Machine Learning-Based Jump-Diffusion Simulation



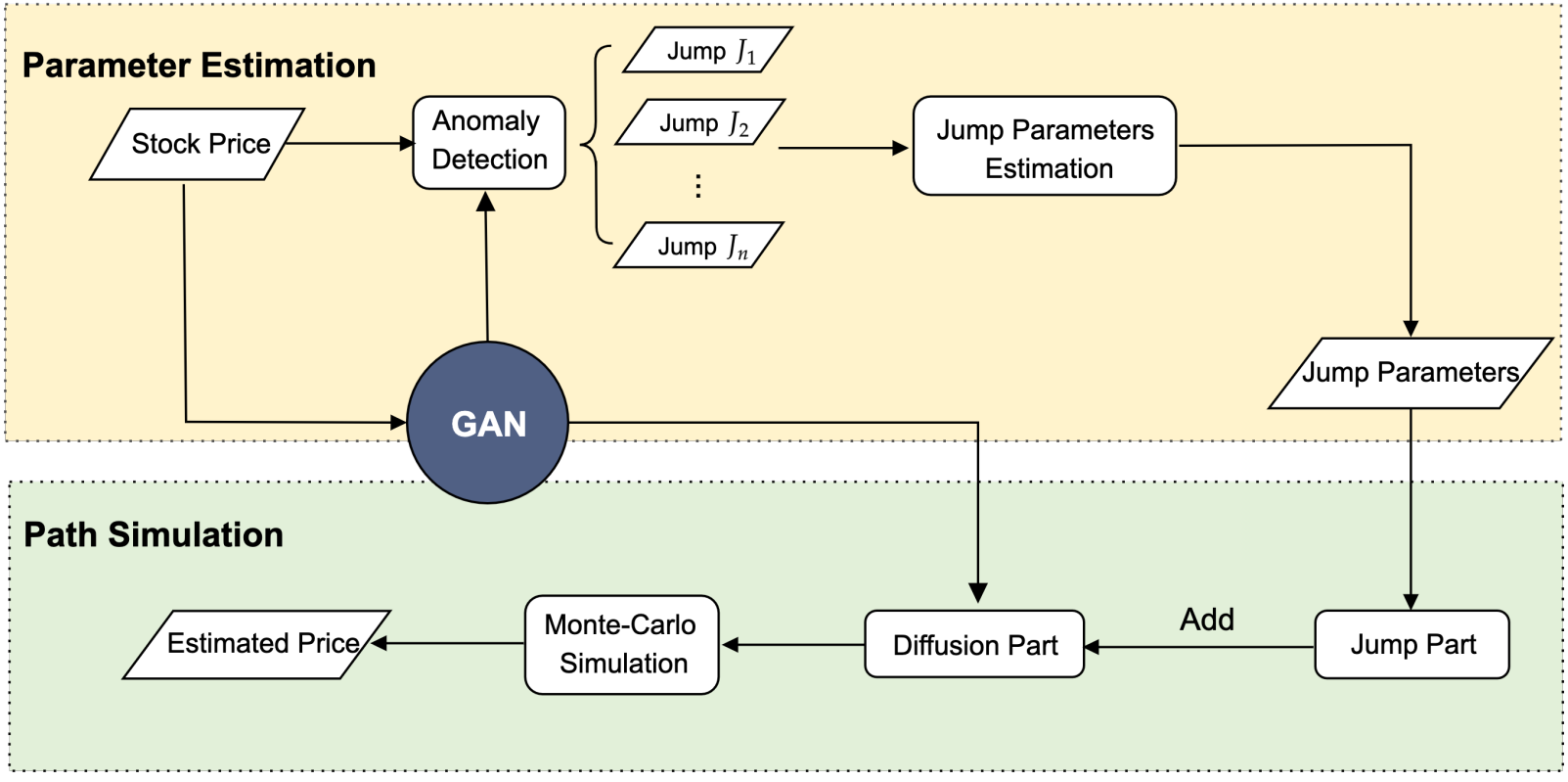
Idea

A Machine Learning-Based Jump-Diffusion Simulation



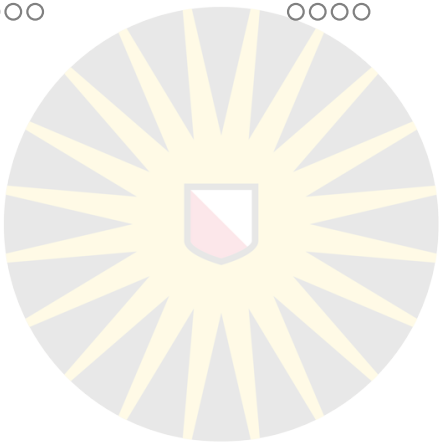
Idea

A Machine Learning-Based Jump-Diffusion Simulation



Generative Adversarial Networks

Introduction

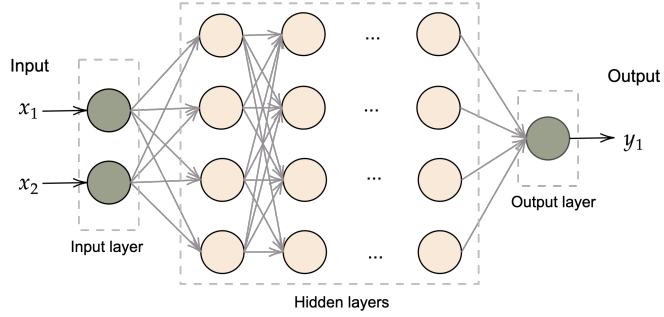


- GANs are a special kind of artificial intelligence algorithm proposed by Ian Goodfellow and his colleagues in 2014.

Generative Adversarial Networks

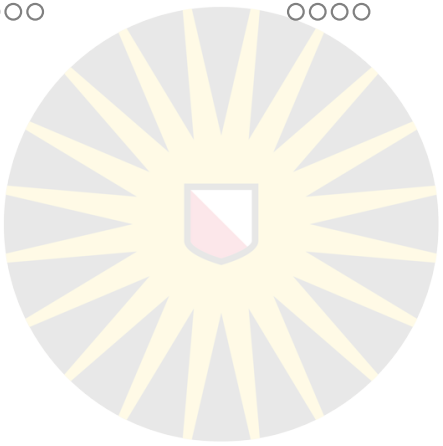
Introduction

- GANs are a special kind of artificial intelligence algorithm proposed by Ian Goodfellow and his colleagues in 2014.
- A GAN consists of two neural networks, which compete with each other forming a zero-sum game.

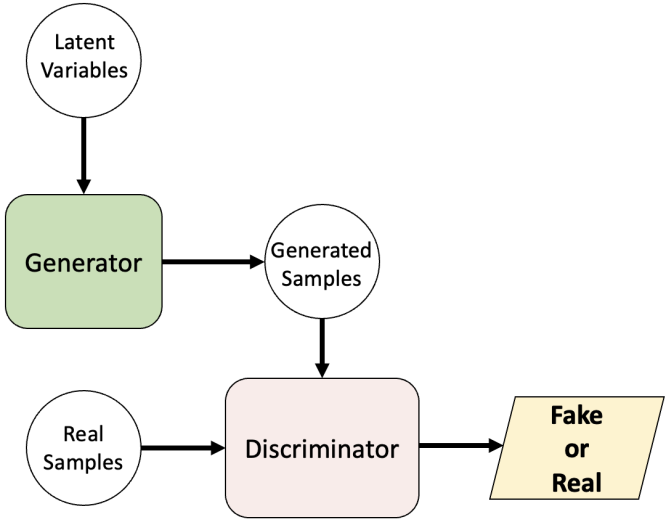


Generative Adversarial Networks

Introduction



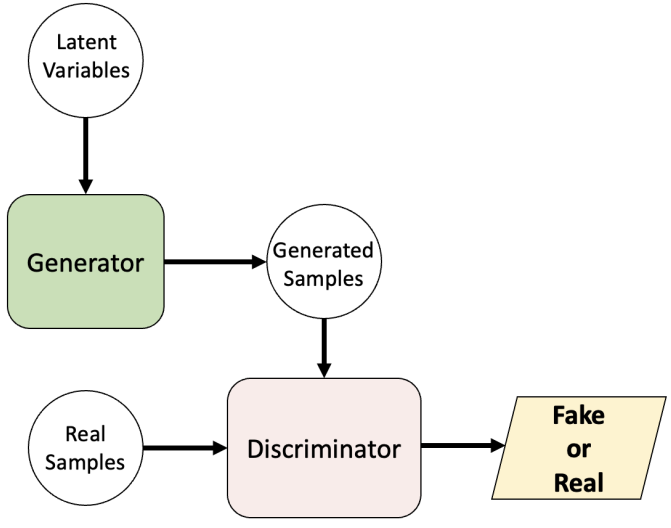
- A GAN consists of two neural networks, which compete with each other forming a zero-sum game.
- The generator G aims to make realistic patterns; The discriminator D tells the generated patterns and the real (authentic) patterns apart.



Generative Adversarial Networks

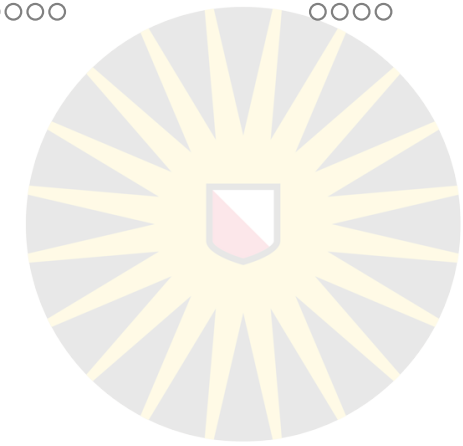
Introduction

- A GAN consists of two neural networks, which compete with each other forming a zero-sum game.
- The generator G aims to make realistic patterns; The discriminator D tells the generated patterns and the real (authentic) patterns apart.
- G and D are trained simultaneously, and in competition with each other.



Generative Adversarial Networks

From a Mathematical Point of View

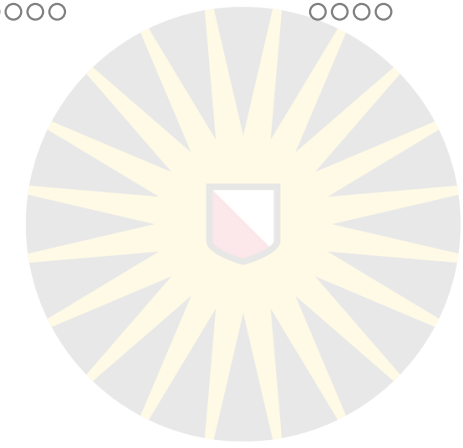


$$
\begin{aligned}
G_\eta: \quad & \mathcal{Z} \subseteq \mathbb{R}^m && \to && G(\mathcal{Z};\eta) \subseteq \mathcal{X} \subseteq \mathbb{R}^n \\
& \mathbf{z} \sim \mathbb{P}_z && \mapsto && \hat{\mathbf{x}} \sim \mathbb{P}_G, \\
D_\theta: \quad & \mathcal{X} \subseteq \mathbb{R}^n && \to && D(\mathcal{X};\theta) \subseteq [0,1] \\
& \mathbf{x} && \mapsto && D(\mathbf{x};\theta),
\end{aligned}
$$

where G and D are essentially differentiable functions with parameters η and θ respectively; the latent variable $\mathbf{z}$ is usually normally distributed; $\hat{\mathbf{x}} = G(\mathbf{z};\eta)$ is a generated sample; $\mathbf{x}$ is a sample that is either from the real data or generated by G; $D(\mathbf{x};\theta)$ measures the possibility of $\mathbf{x}$ sampling from $\mathbb{P}_{\text{data}}$ rather than $\mathbb{P}_G$.

Generative Adversarial Networks

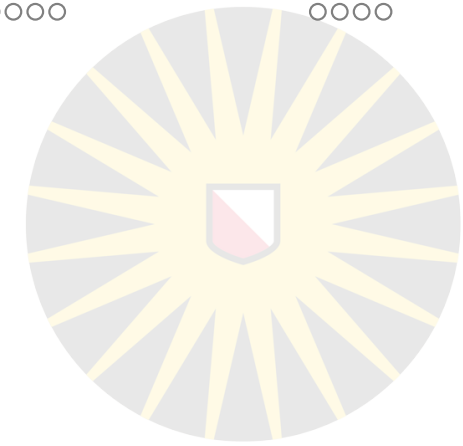
Training GANs



- The training of GANs aims to find the optimal parameters for both G and D:
  - For D: maximize the accuracy of D about assigning the correct label to the inputs;
  - For G: maximize the probability of G about confusing D.

Generative Adversarial Networks

Training GANs



- The training of GANs aims to find the optimal parameters for both G and D:
  - For D: maximize the accuracy of D about assigning the correct label to the inputs;
  - For G: maximize the probability of G about confusing D.
- Use binary cross-entropy as loss functions:

$$L_{D_\theta} = -\mathbb{E}_{\mathbf{x}_d \sim p_{\text{data}}} \left[\log D_\theta \left(\mathbf{x}_d\right)\right] - \mathbb{E}_{\mathbf{z} \sim p_z} \left[\log \left(1 - D_\theta \left(G_\eta \left(\mathbf{z}\right)\right)\right)\right]$$

$$L_{G_\eta} = \mathbb{E}_{\mathbf{z} \sim p_z} \left[\log \left(1 - D_\theta \left(G_\eta \left(\mathbf{z}\right)\right)\right)\right]$$

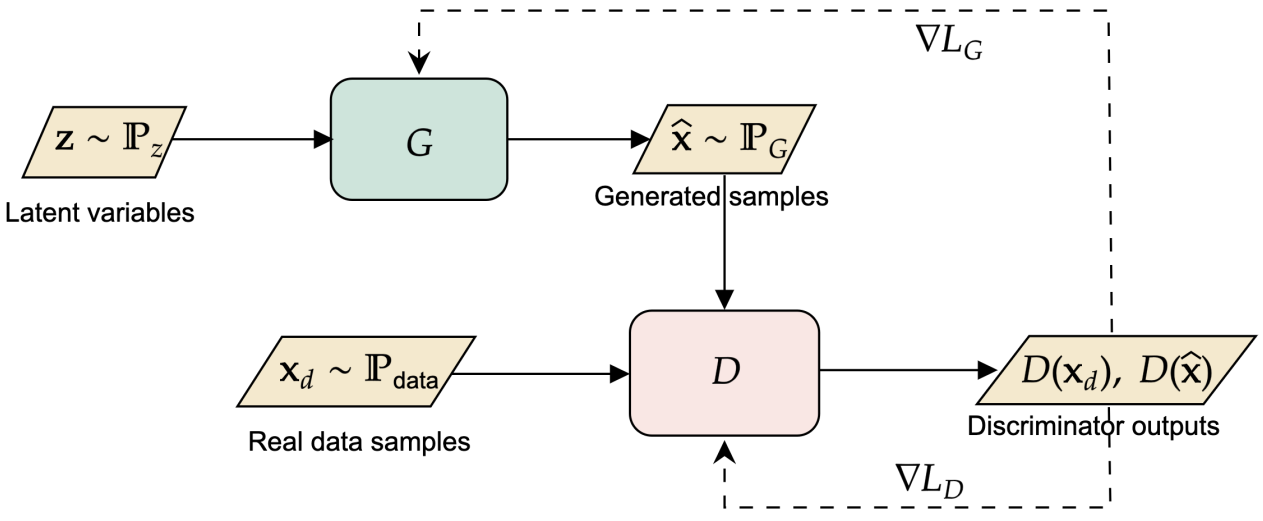
Generative Adversarial Networks

Training GANs

- The GAN is trained to solve the minimax problem:

$$\min_{G_\eta} \max_{D_\theta} V\left(G_\eta, D_\theta\right),$$

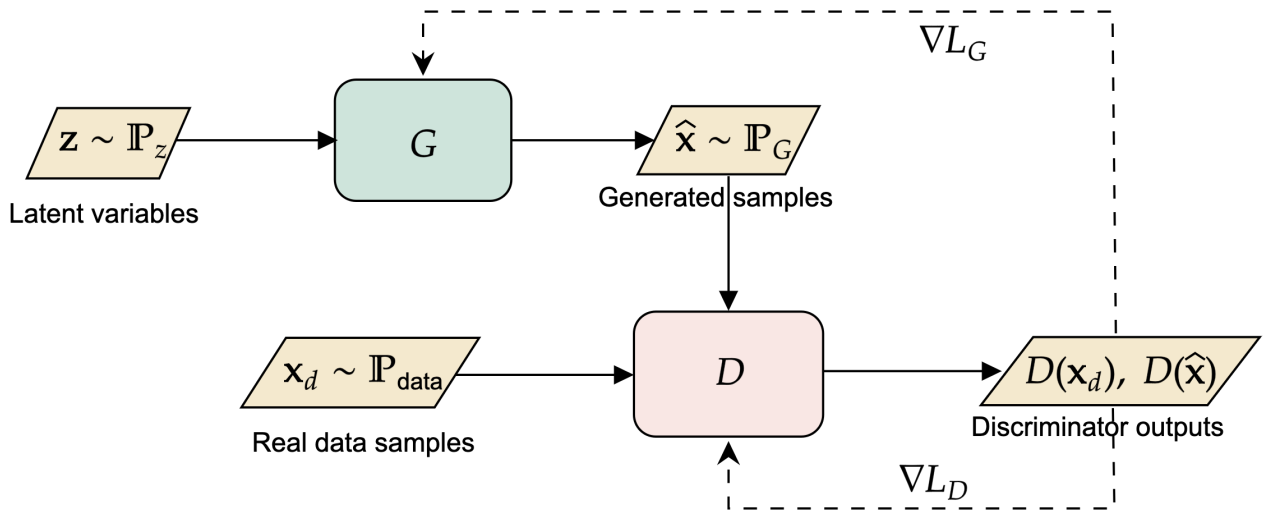
$$V\left(G_\eta, D_\theta\right) = \mathbb{E}_{\mathbf{x}_d \sim p_{\text{data}}}\left[\log D_\theta\left(\mathbf{x}_d\right)\right] + \mathbb{E}_{\mathbf{z} \sim p_z}\left[\log\left(1 - D_\theta\left(G_\eta\left(\mathbf{z}\right)\right)\right)\right].$$



Generative Adversarial Networks

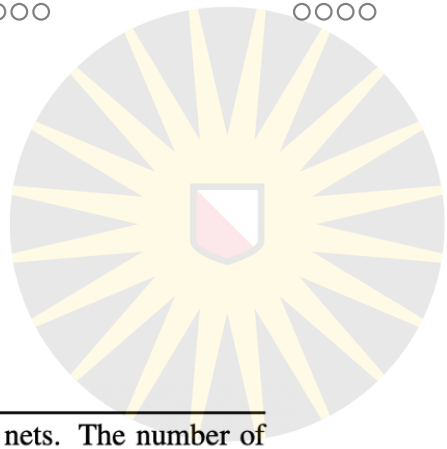
Training GANs

- Use gradient descent methods to find the optimum:
  - The gradients are calculated by the backpropagation algorithm;
  - The parameters θ and η are updated by the Adam algorithm.



Generative Adversarial Networks

Training GANs



Algorithm 1 Minibatch stochastic gradient descent training of generative adversarial nets. The number of steps to apply to the discriminator, k, is a hyperparameter. We used $k = 1$, the least expensive option, in our experiments.

for number of training iterations **do**

Discriminator Training

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

Generator Training

- 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:

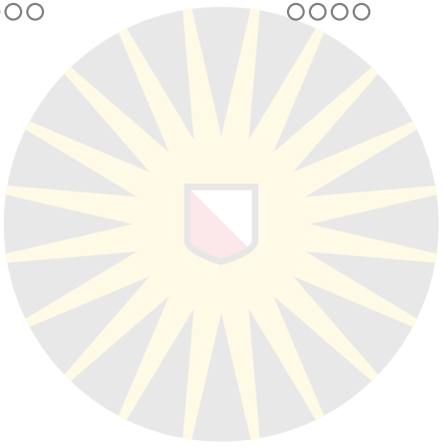
$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right).$$

end for

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

Generative Adversarial Networks

Theoretical Results



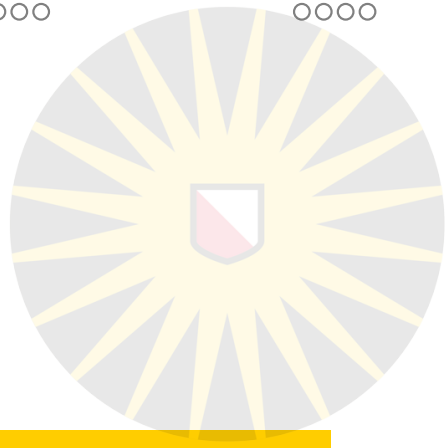
Theorem (Optimal discriminator)

When the generator G is fixed, the optimal discriminator D_θ^* is given by

$$D_\theta^*(\mathbf{x}) = \frac{p_{data}(\mathbf{x})}{p_{data}(\mathbf{x}) + p_G(\mathbf{x})}.$$

Generative Adversarial Networks

Theoretical Results



Theorem (Optimal discriminator)

When the generator G is fixed, the optimal discriminator D_θ^* is given by

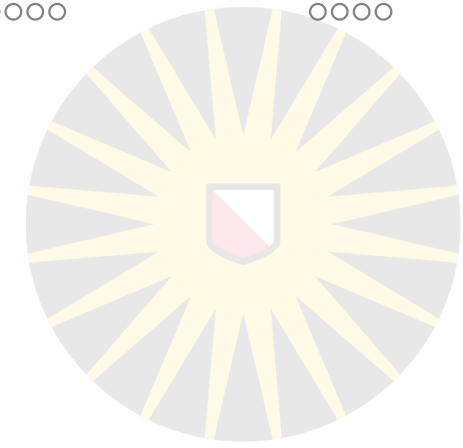
$$D_\theta^*(\mathbf{x}) = \frac{p_{data}(\mathbf{x})}{p_{data}(\mathbf{x}) + p_G(\mathbf{x})}.$$

- Related to JS divergence:

$$V\left(G_\eta, D_\theta^*\right) = \mathbb{E}_{\mathbf{x}_d \sim \mathbb{P}_{\text{data}}}\left[\log D_\theta^*\left(\mathbf{x}_d\right)\right] + \mathbb{E}_{\hat{\mathbf{x}} \sim \mathbb{P}_G}\left[\log\left(1 - D_\theta^*(\hat{\mathbf{x}})\right)\right]$$

Generative Adversarial Networks

Theoretical Results



- Related to JS divergence:

$$
\begin{aligned}
V\left(G_\eta, D_\theta^*\right) &= \mathbb{E}_{\mathbf{x}_d \sim \mathbb{P}_{\text{data}}}\left[\log D_\theta^*\left(\mathbf{x}_d\right)\right] + \mathbb{E}_{\hat{\mathbf{x}} \sim \mathbb{P}_G}\left[\log\left(1 - D_\theta^*(\hat{\mathbf{x}})\right]\right. \\
&= \mathbb{E}_{\mathbf{x}_d \sim \mathbb{P}_{\text{data}}}\left[\log \frac{p_{\text{data}}\left(\mathbf{x}_d\right)}{p_{\text{data}}\left(\mathbf{x}_d\right) + p_G\left(\mathbf{x}_d\right)}\right] \\
&+ \mathbb{E}_{\hat{\mathbf{x}} \sim \mathbb{P}_G}\left[\log \frac{p_G\left(\hat{\mathbf{x}}\right)}{p_{\text{data}}\left(\hat{\mathbf{x}}\right) + p_G\left(\hat{\mathbf{x}}\right)}\right],
\end{aligned}
$$

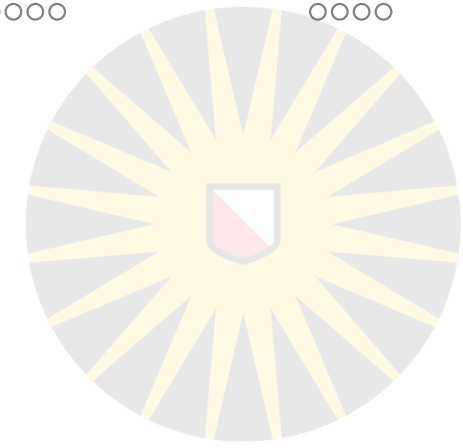
Jensen-Shannon (JS) divergence

$$
\mathrm{JS}(P\|Q) = \frac{1}{2}\mathrm{KL}\left(P\|\frac{P+Q}{2}\right) + \frac{1}{2}\mathrm{KL}\left(Q\|\frac{P+Q}{2}\right),
$$

where $\mathrm{KL}(P\|Q) = \int p(x) \log \frac{p(x)}{q(x)}\, \mathrm{d}x$ is the Kullback-Leibler (KL) divergence.

Generative Adversarial Networks

Theoretical Results



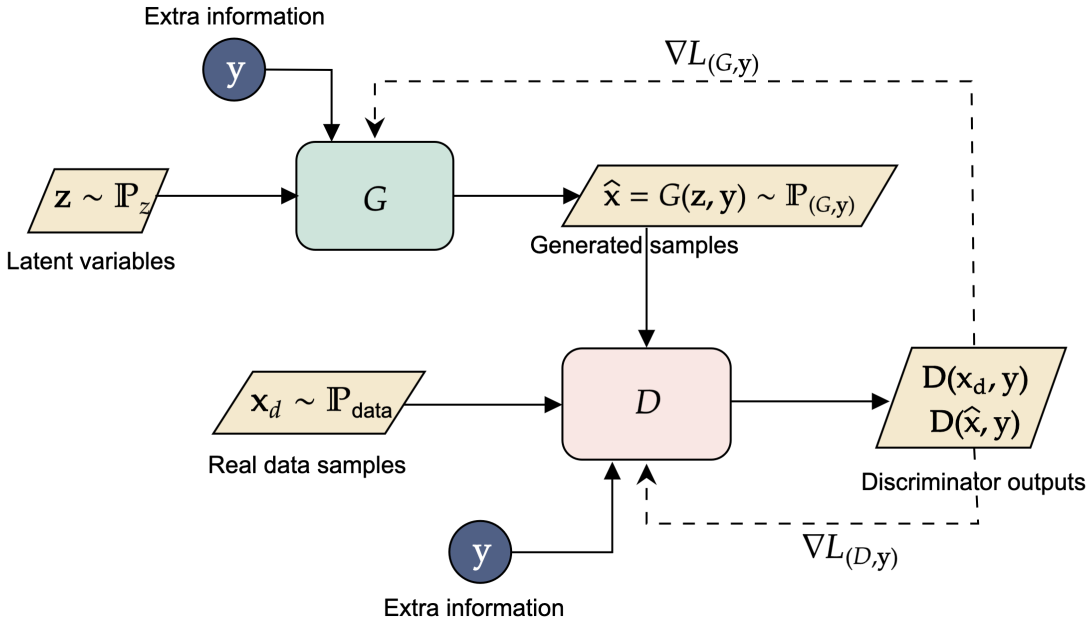
Theorem (Global optimality)

$V(G_\eta, D_\theta^*)$ is globally minimized if and only if $\mathbb{P}_{data} = \mathbb{P}_G$, and the minimum value is $-2\log 2$.

$$V(G_\eta, D_\theta^*) = \mathbb{E}_{\mathbf{x}_d \sim \mathbb{P}_{\text{data}}} \left[\log \frac{p_{\text{data}}(\mathbf{x}_d)}{p_{\text{data}}(\mathbf{x}_d) + p_G(\mathbf{x}_d)} \right] + \mathbb{E}_{\hat{\mathbf{x}} \sim \mathbb{P}_G} \left[\log \frac{p_G(\hat{\mathbf{x}})}{p_{\text{data}}(\hat{\mathbf{x}}) + p_G(\hat{\mathbf{x}})} \right],$$

Conditional GANs

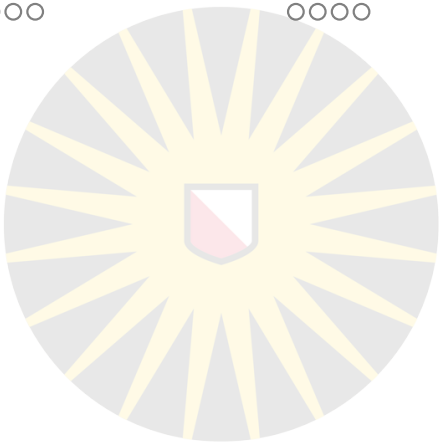
A Direct Extension of Vanilla GAN



- The minimax problem with respect to the conditional GAN:

$$\min_{G_\eta} \max_{D_\theta} V(G_\eta, D_\theta; \mathbf{y}) = \mathbb{E}_{\mathbf{x}_d \sim \mathbb{P}_{\text{data}}} [\log D_\theta(\mathbf{x}_d, \mathbf{y})] + \mathbb{E}_{\mathbf{z} \sim \mathbb{P}_z} [\log(1 - D_\theta(G_\eta(\mathbf{z}, \mathbf{y})))].$$

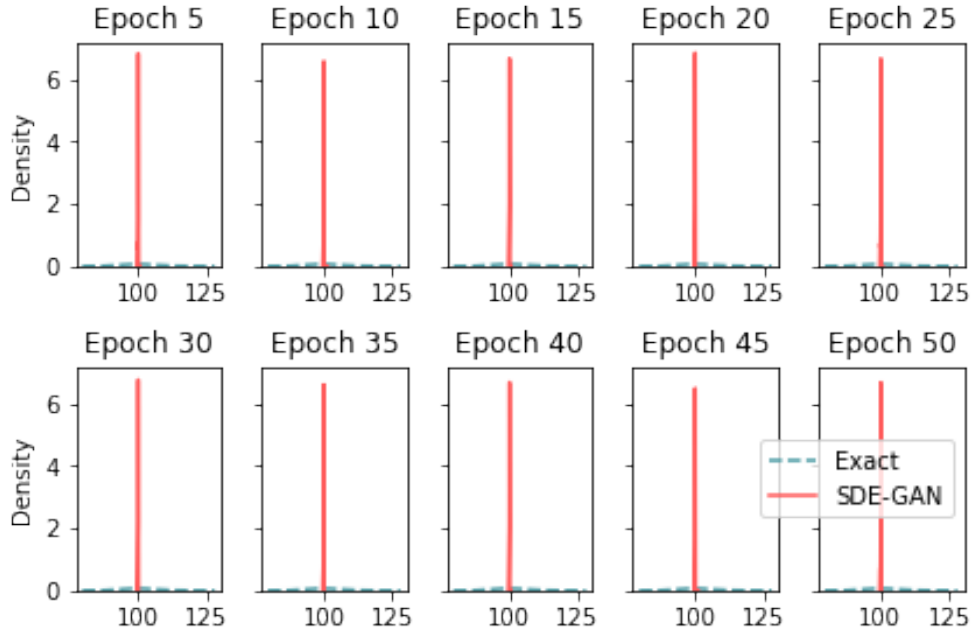
GAN Failure Modes



- Mode collapse
- Vanishing gradients
- Non-convergence

GAN Failure Modes

- Mode collapse
  - The generator can only produce a small set of outputs.
  - The generator learns an especially plausible distribution and only produce that output, then the best strategy for the discriminator is to reject.



GAN Failure Modes

- Vanishing gradients
  - It is usually caused by the discriminator.
  - When a discriminator performs too well, the generator then cannot receive useful information to make progress.

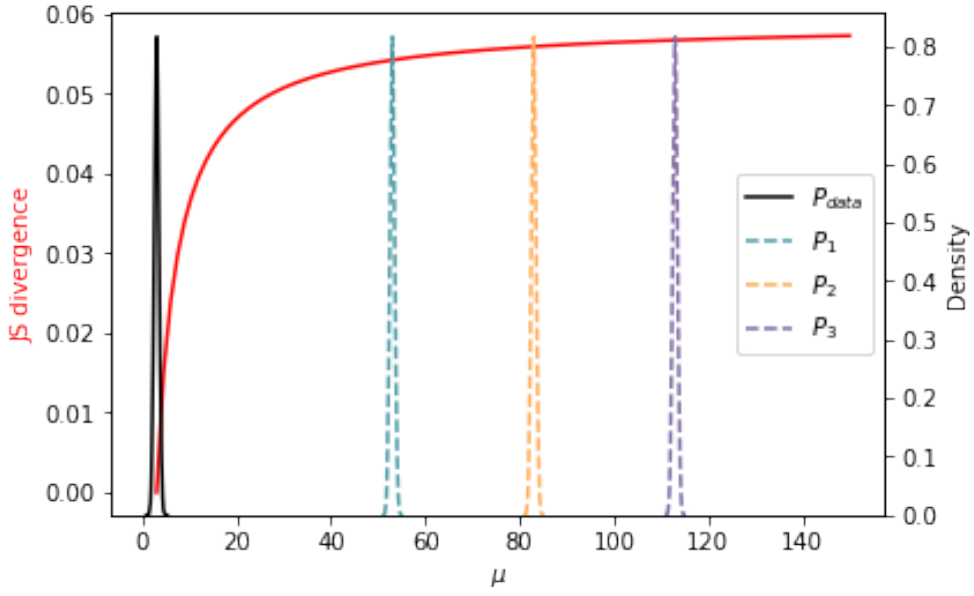
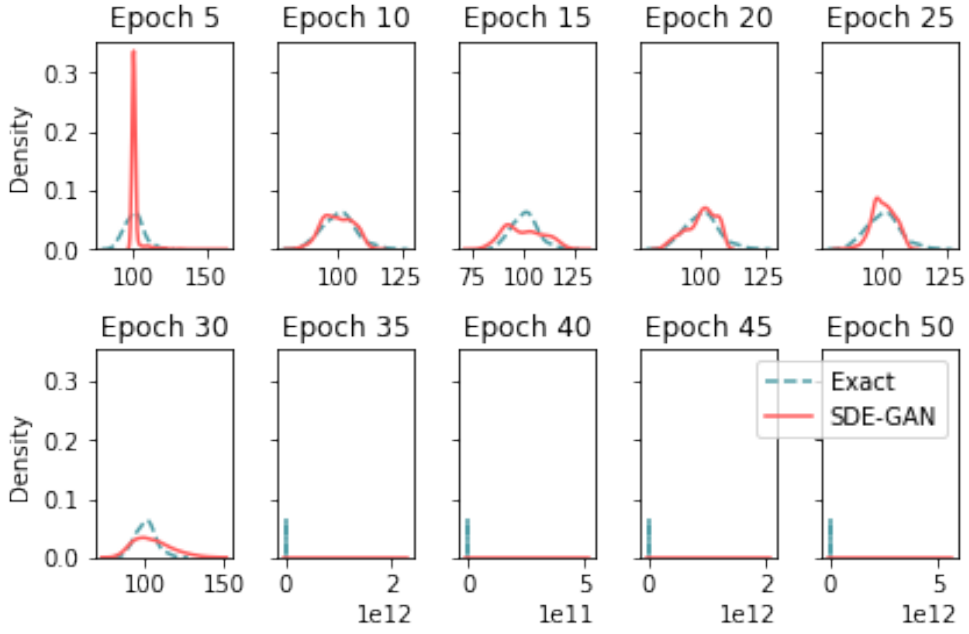


Figure: JS divergence between distributions $\mathbb{P}_n$ and $\mathbb{P}_{\text{data}}$, where $\mathbb{P}_{\text{data}} \sim \mathcal{N}(3, 0.5^2)$, $\mathbb{P}_n \sim \mathcal{N}(\mu, 0.5^2)$ for $\mu \in [3, 150]$, in particular, $\mathbb{P}_1 \sim \mathcal{N}(50, 0.5^2)$, $\mathbb{P}_2 \sim \mathcal{N}(80, 0.5^2)$ and $\mathbb{P}_3 \sim \mathcal{N}(110, 0.5^2)$.

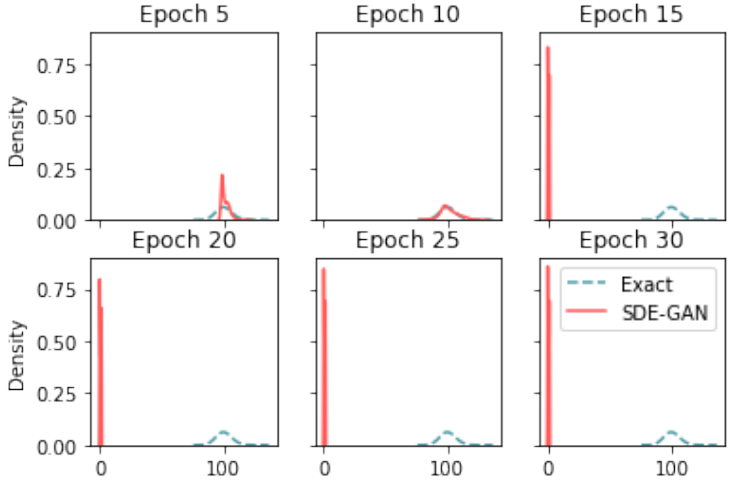
GAN Failure Modes

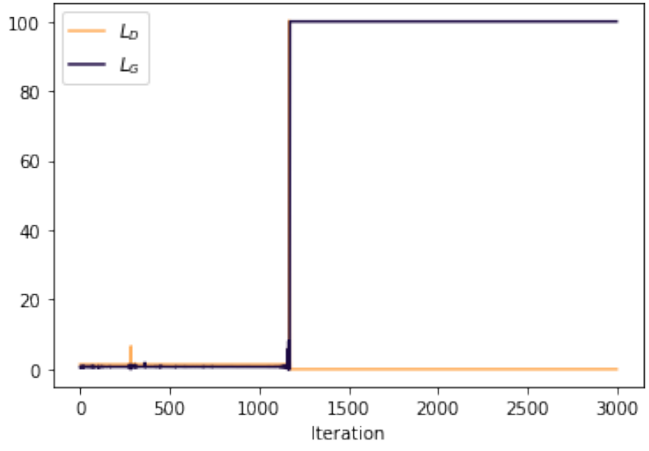
- Vanishing gradients
  - It is usually caused by the discriminator.
  - When a discriminator performs too well, the generator then cannot receive useful information to make progress.



GAN Failure Modes

- Non-Convergence
  - No equilibrium can be found between the discriminator and generator.
  - One signal to identify GANs non-convergence failure is the discriminator loss decreasing to zero or close to zero.
  - The generator only provide low quality outputs which are easily be recognized as fake samples by the discriminator.





Wasserstein GAN

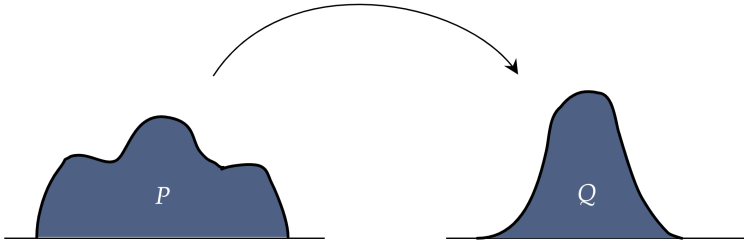
Wasserstein Loss

- Wasserstein GAN involves the so-called 1-Wasserstein distance.

Definition (1-Wasserstein distance)

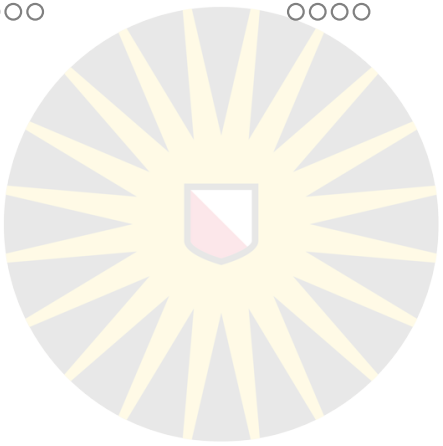
$$W(\mathbb{P}_r, \mathbb{P}_g) = \inf_{(X,Y)} \left\{ \mathbb{E}_{(X,Y)}[\|X - Y\|] : X \sim \mathbb{P}_r, Y \sim \mathbb{P}_g \right\}.$$

- 1-Wassersein distance can be interpreted as the minimum energy cost of moving a pile of material from one form to another form.

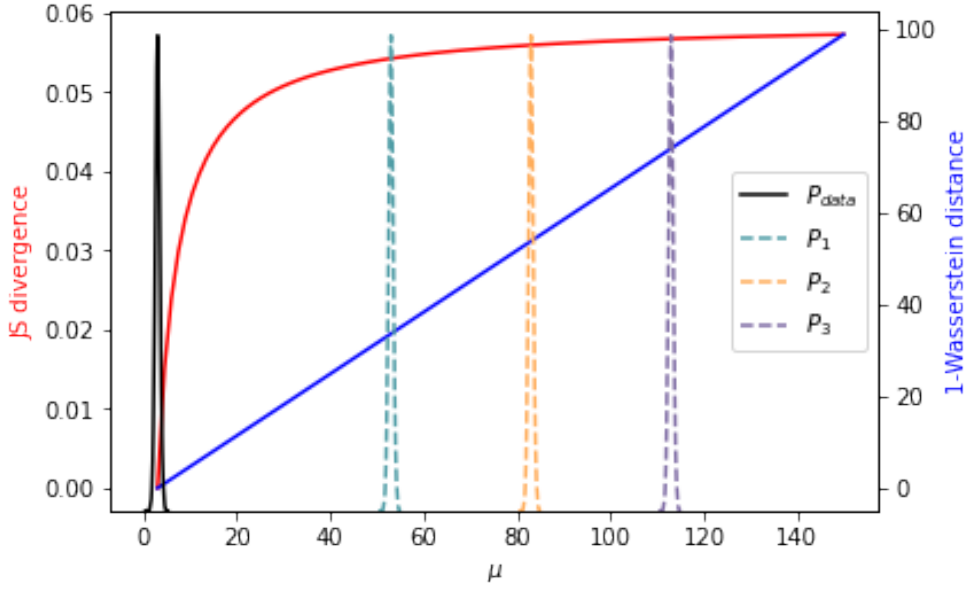


Wasserstein GAN

Wasserstein Loss

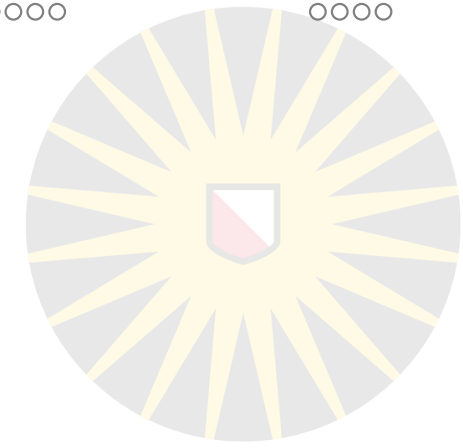


- 1-Wasserstein distance is more sensible than the JS divergence



Wasserstein GAN

Wasserstein Loss

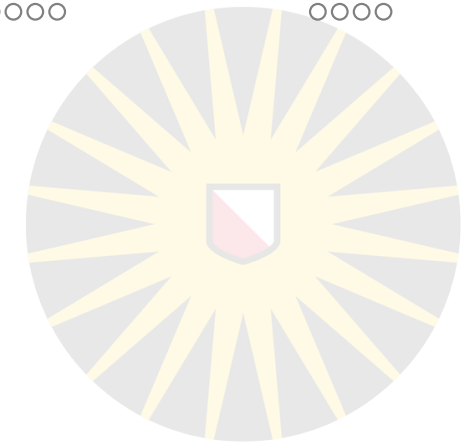


Definition (The Kantorovich-Rubinstein duality)

$$W(\mathbb{P}_r, \mathbb{P}_g) = \sup_{\|\nabla f\|_\infty \leq 1} \mathbb{E}_{x \sim \mathbb{P}_r}[f(x)] - \mathbb{E}_{x \sim \mathbb{P}_g}[f(x)].$$

Wasserstein GAN

Wasserstein Loss



Definition (The Kantorovich-Rubinstein duality)

$$W(\mathbb{P}_r, \mathbb{P}_g) = \sup_{\|\nabla f\|_\infty \leq 1} \mathbb{E}_{x\sim\mathbb{P}_r}[f(x)] - \mathbb{E}_{x\sim\mathbb{P}_g}[f(x)].$$

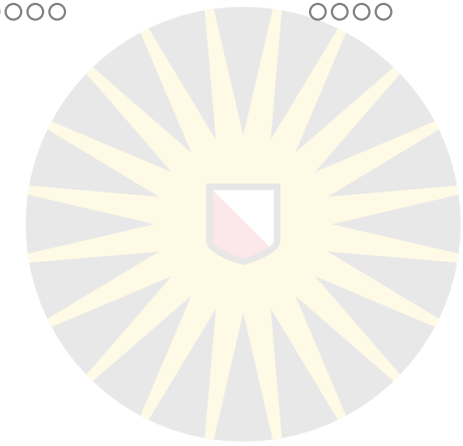
- The minimax problem of the Wasserstein GAN:

$$\min_{G_\eta} \max_{D\in\mathcal{D}} \mathbb{E}_{\mathbf{x}\sim\mathbb{P}_{\text{data}}}[D(\mathbf{x})] - \mathbb{E}_{\mathbf{z}\sim\mathbb{P}_z}[D(G_\eta(z))],$$

where $\mathcal{D}$ is the set of 1-Lipschitz functions.

Wasserstein GAN

Wasserstein Loss



- The minimax problem of the Wasserstein GAN:

$$\min_{G_\eta} \max_{D \in \mathcal{D}} \mathbb{E}_{\mathbf{x} \sim \mathbb{P}_{\text{data}}}[D(\mathbf{x})] - \mathbb{E}_{\mathbf{z} \sim \mathbb{P}_z}[D(G_\eta(z))],$$

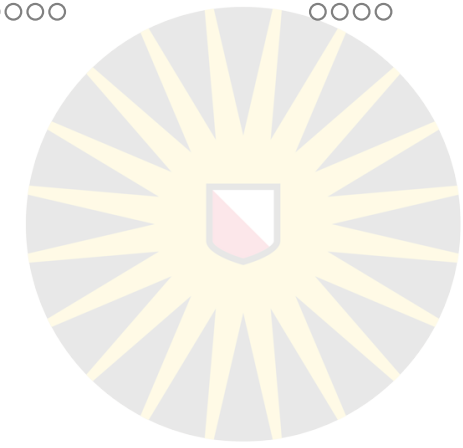
  where $\mathcal{D}$ is the set of 1-Lipschitz functions.

- Note that the discriminator output $D(\mathbf{x})$ here does not signify the probability of real samples, and its value is not bounded by $[0, 1]$.

  $\Rightarrow$ The discriminator is called a critic in Wasserstein GAN, and it is denoted by f_w.

Wasserstein GAN

Wasserstein Loss



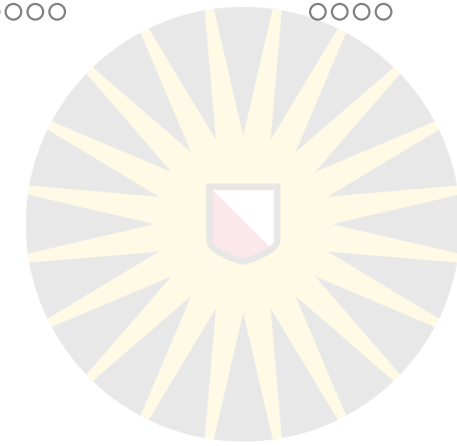
- The minimax problem of the Wasserstein GAN:

$$\min_{G_\eta} \max_{w \in \mathcal{W}} \mathbb{E}_{\mathbf{x} \sim \mathbb{P}_{\text{data}}} [f_w(\mathbf{x})] - \mathbb{E}_{\mathbf{z} \sim \mathbb{P}_z} [f_w(G_\eta(\mathbf{z}))],$$

where $\{f_w\}_{w \in \mathcal{W}}$ is a parameterized family of 1-Lipschitz functions.

Wasserstein GAN

Wasserstein Loss



- The minimax problem of the Wasserstein GAN:

$$\min_{G_\eta} \max_{w \in \mathcal{W}} \mathbb{E}_{\mathbf{x} \sim \mathbb{P}_{\text{data}}} [f_w(\mathbf{x})] - \mathbb{E}_{\mathbf{z} \sim \mathbb{P}_z} [f_w(G_\eta(\mathbf{z}))] ,$$

  where $\{f_w\}_{w \in \mathcal{W}}$ is a parameterized family of 1-Lipschitz functions.

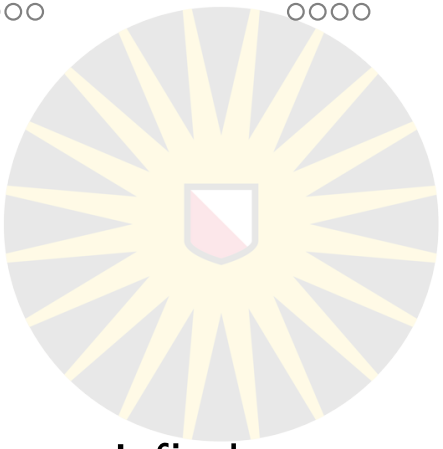
- The loss functions:

$$L_{f_w} = - \left(\mathbb{E}_{\mathbf{x} \sim \mathbb{P}_{\text{data}}} [f_w(\mathbf{x})] - \mathbb{E}_{\mathbf{z} \sim \mathbb{P}_z} [f_w(G_\eta(\mathbf{z}))] \right)$$

$$L_G = -\mathbb{E}_{\mathbf{z} \sim \mathbb{P}_z} [f_w(G_\eta(\mathbf{z}))]$$

Wasserstein GAN

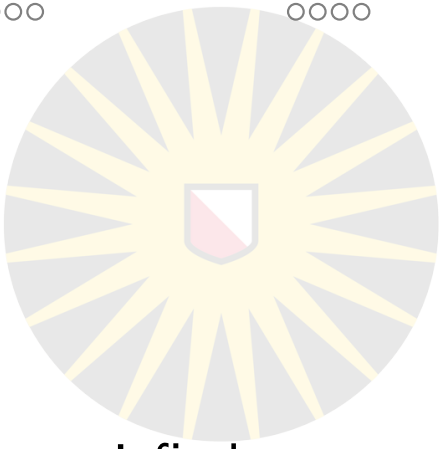
Gradient Penalty



- The 1-Lipschitz constraint $\|\nabla f\|_\infty \leq 1$ should be satisfied during the training.

Wasserstein GAN

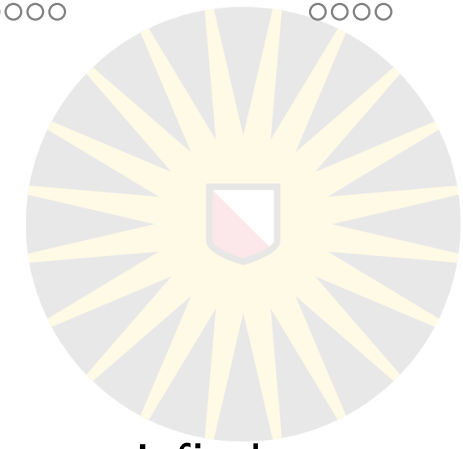
Gradient Penalty



- The 1-Lipschitz constraint $\|\nabla f\|_\infty \leq 1$ should be satisfied during the training.
- Weight clipping method: Clamp the parameters w to a fixed box (say $\mathcal{W} = [0.01, 0.01]^l$) after each gradient update.

Wasserstein GAN

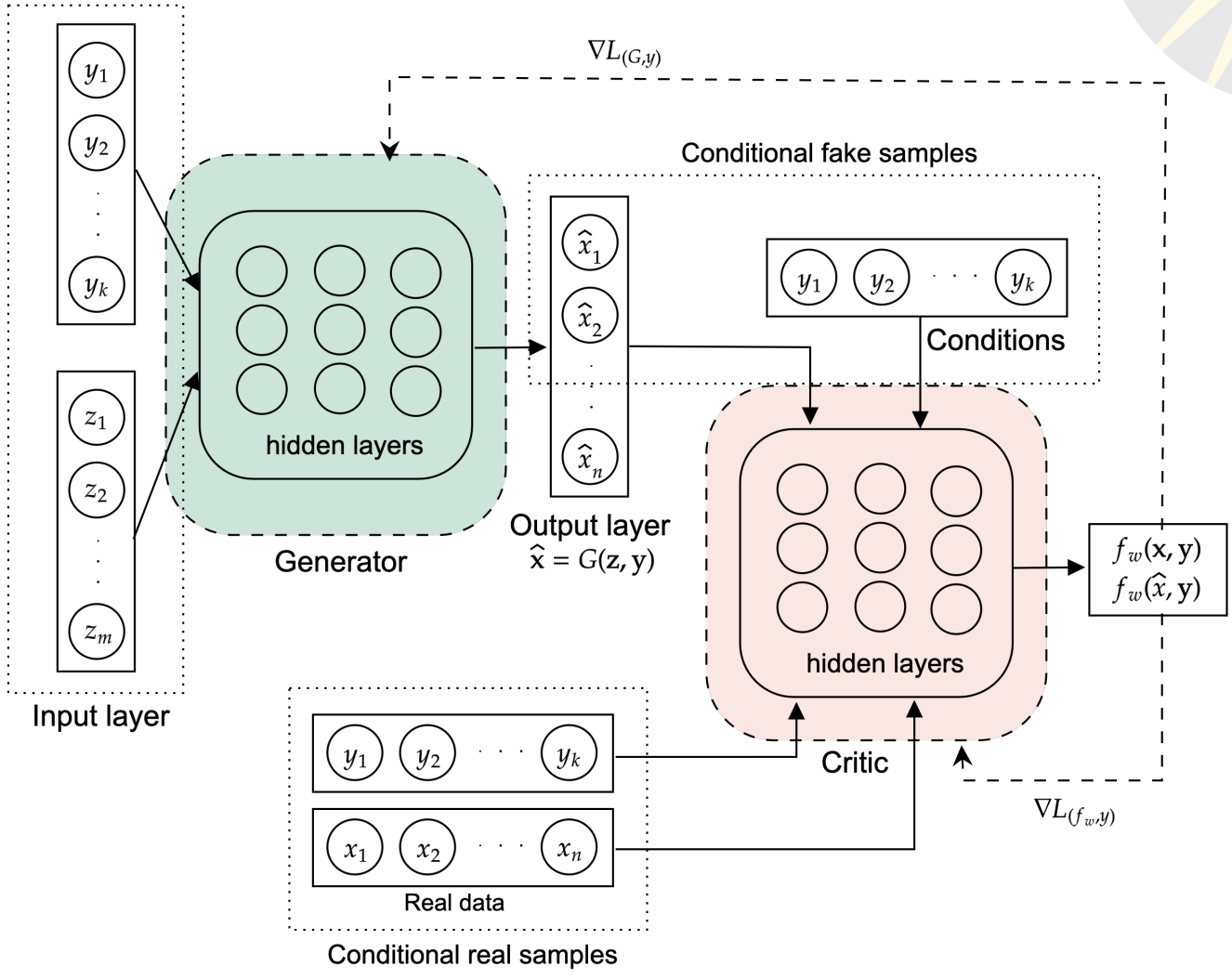
Gradient Penalty



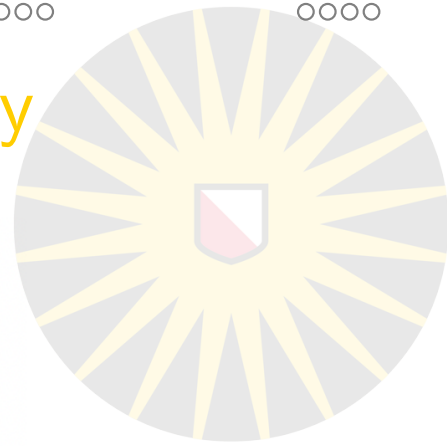
- The 1-Lipschitz constraint $\|\nabla f\|_\infty \leq 1$ should be satisfied during the training.
- Weight clipping method: Clamp the parameters w to a fixed box (say $\mathcal{W} = [0.01, 0.01]^l$) after each gradient update.
- Gradient penalty method: Add an additional term to the critic loss to enforce the 1-Lipschitz constraint.
- $L_{f_w} =$
$\underbrace{\mathbb{E}_{\hat{\mathbf{x}}\sim\mathbb{P}_G}[f_w(\hat{\mathbf{x}})] - \mathbb{E}_{\mathbf{x}\sim\mathbb{P}_{\text{data}}}[f_w(\mathbf{x})]}_{\text{Original critic loss}} + \underbrace{\lambda\mathbb{E}_{\tilde{x}\sim\ \mathbb{P}_{\tilde{x}}}[(\|\nabla_{\tilde{x}} f_w(\tilde{x})\|_2 - 1)^2]}_{\text{gradient penalty}},$
where, $\mathbb{P}_{\tilde{x}}$ samples uniformly along the straight lines between pairs (x, y) with any $x \sim \mathbb{P}_{\text{data}}$ and $y \sim \mathbb{P}_G$.

Conditional Wasserstein with Gradient Penalty

Architecture



Conditional Wasserstein with Gradient Penalty



Algorithm 8: Training algorithm of cWGAN-GP.

Input : Training epochs N, mini-batch size L, mini-batch number M, initial critic parameters w_0, initial generator parameters η_0, critic iteration number N_c, gradient penalty coefficient λ, extra information $\mathbf{y}$.

Output: Critic parameters w, generator parameters η.

for *Training epoch* $n = 1, \ldots, N$ **do**

  for *Mini-batch* $m = 1, \ldots, M$ **do**

    for *Critic iteration* $n_c = 1, \ldots, N_C$ **do**

      Generate L random latent variables $\left\{\mathbf{z}^{(1)}, \ldots, \mathbf{z}^{(L)}\right\}$ from distribution P_z.

      Choose L random samples $\left\{\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(L)}\right\}$ from the training data.

      /* Gradient penalty construction. */

      Sample L random numbers $\left\{\epsilon^{(1)}, \ldots, \epsilon^{(L)}\right\}$ from uniform distribution $\mathcal{U}[0,1]$.

      $\hat{\mathbf{x}}^{(l)} \leftarrow G_\eta\left(\mathbf{z}^{(l)}, \mathbf{y}\right)$, for $l = 1, \ldots, L$

      $\tilde{\mathbf{x}}^{(l)} \leftarrow \epsilon \mathbf{x}^{(l)} + (1-\epsilon)\hat{\mathbf{x}}^{(l)}$, for $l = 1, \ldots, L$

      /* Critic loss. */

      $L_{f_w}^m \leftarrow$

      $$-\left[\frac{1}{L}\sum_{l=1}^{L} f_w\left(\mathbf{x}^{(l)}, \mathbf{y}\right) - \frac{1}{L}\sum_{l=1}^{L} f_w\left(\hat{\mathbf{x}}^{(l)}, \mathbf{y}\right)\right] + \lambda\left(\left\|\nabla_{(\tilde{\mathbf{x}},\mathbf{y})} f_w(\tilde{\mathbf{x}}^{(l)}, \mathbf{y})\right\| - 1\right)^2$$

      /* Update critic parameters. */

      $w \leftarrow \text{RMSProp}\left(\nabla_w \sum_{m=1}^{M} L_{f_w}^m\right)$

    end

    Generate L random latent variables $\left\{\mathbf{z}^{(1)}, \ldots, \mathbf{z}^{(L)}\right\}$ from distribution P_z.

    /* Generator loss. */

    $L_G^m \leftarrow -\frac{1}{L}\sum_{l=1}^{L} f_w\left(G_\eta\left(\mathbf{z}^{(l)}, \mathbf{y}\right), \mathbf{y}\right)$

    /* Update generator parameters. */

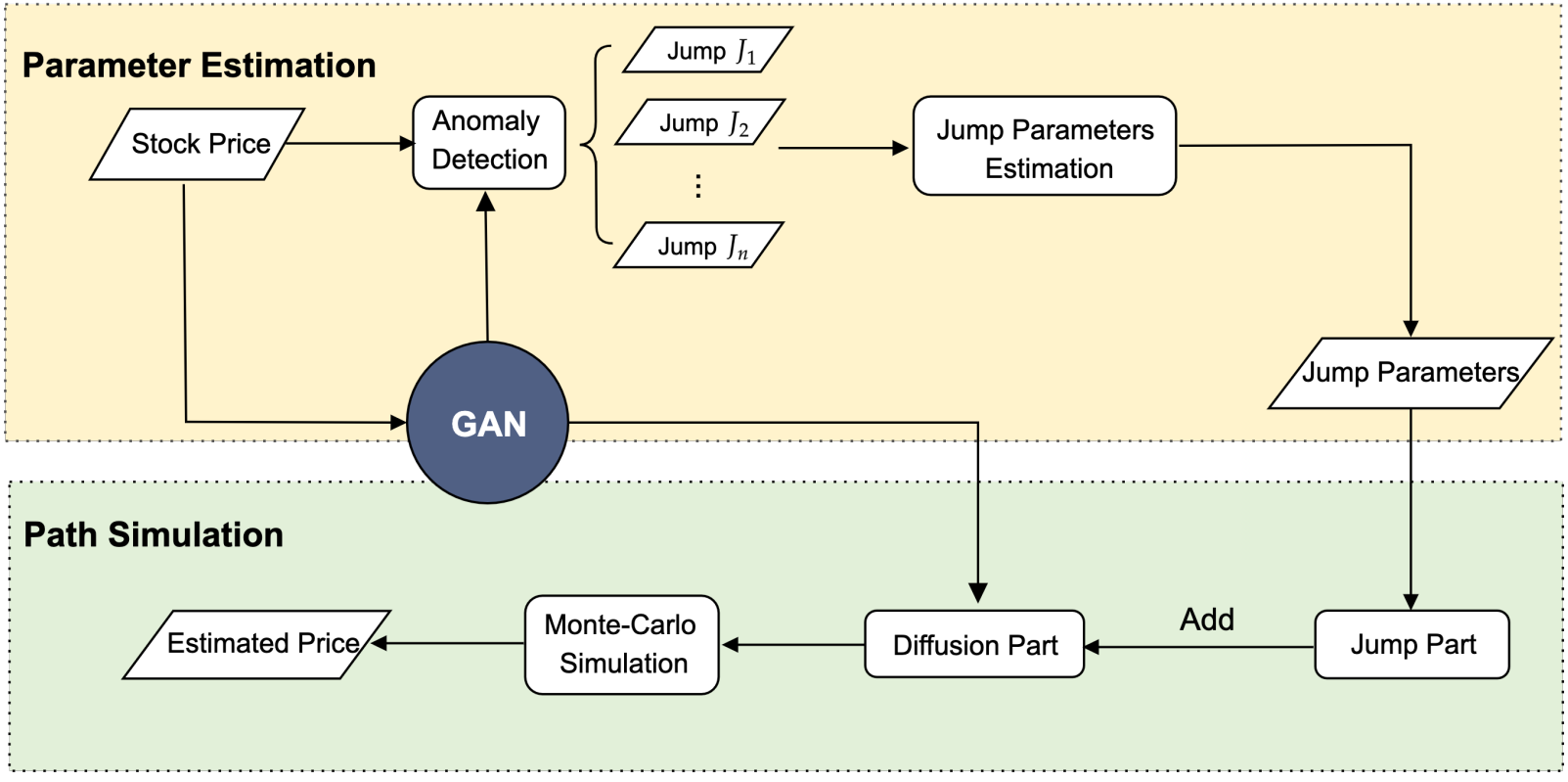
    $\eta \leftarrow \text{RMSProp}\left(\nabla_\eta \sum_{m=1}^{M} L_G^m\right)$

  end

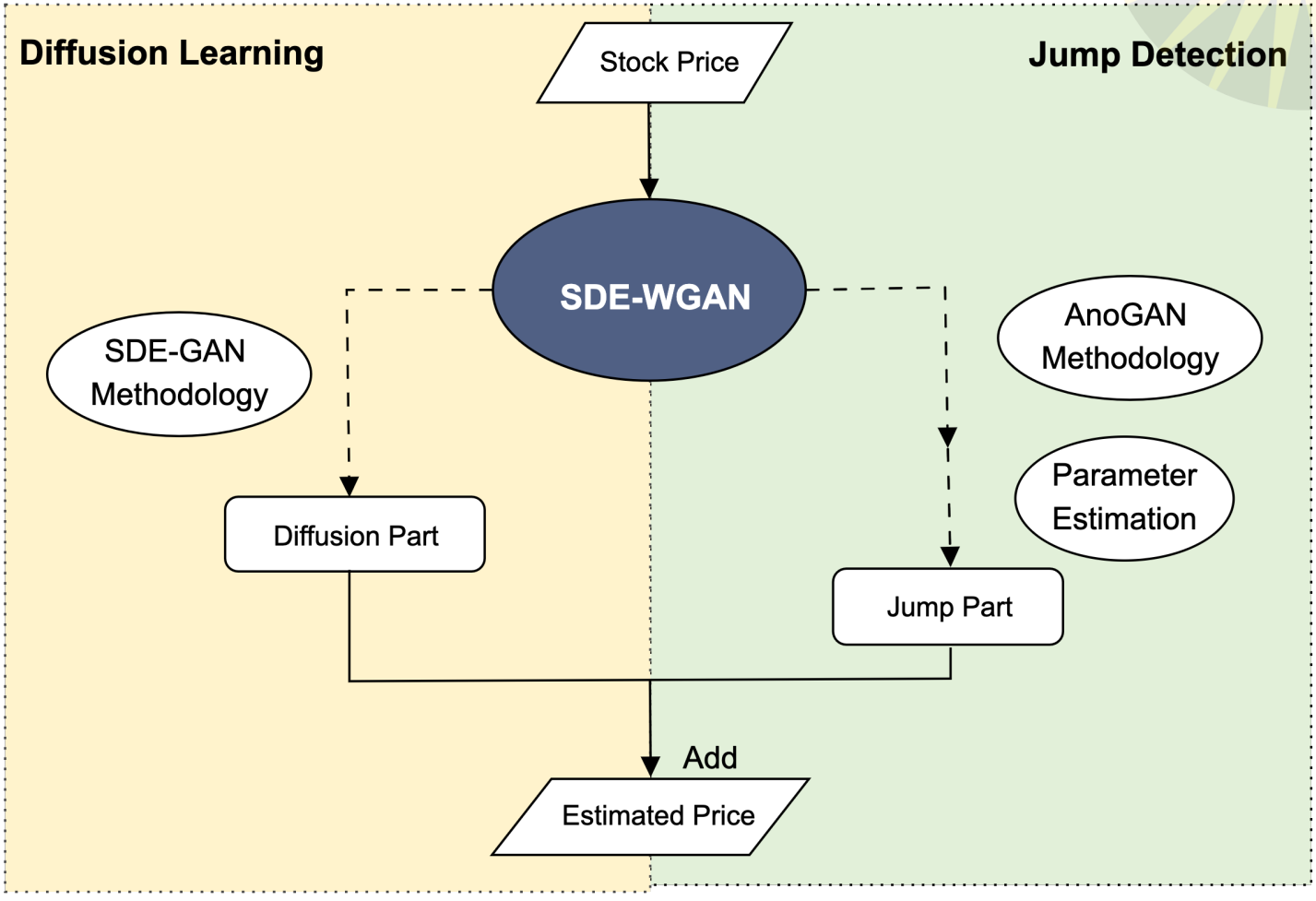
end

Detailed Proposed Framework

Recall

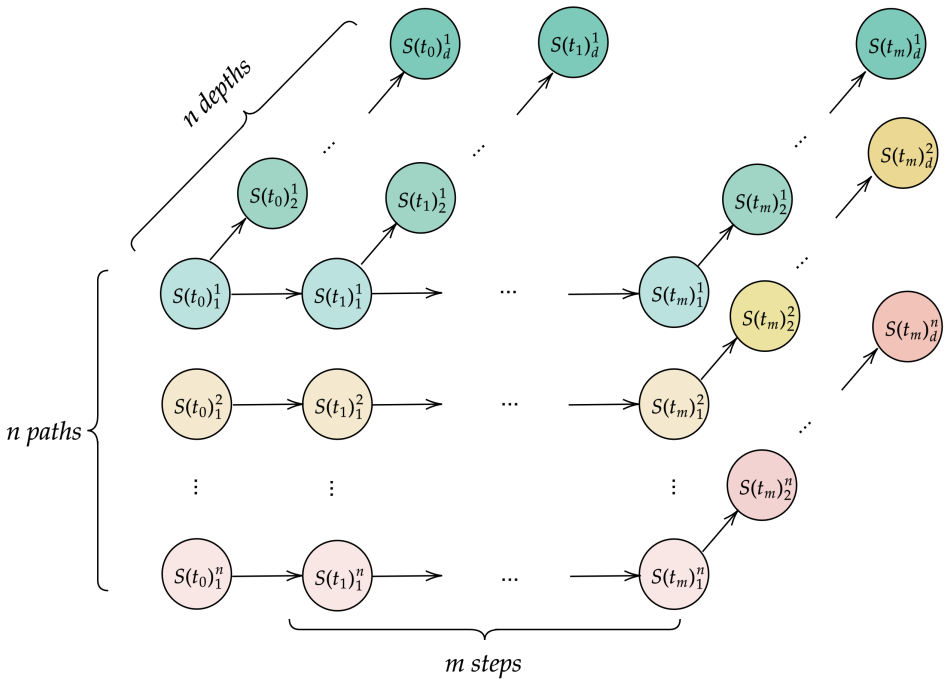


Detailed Proposed Framework



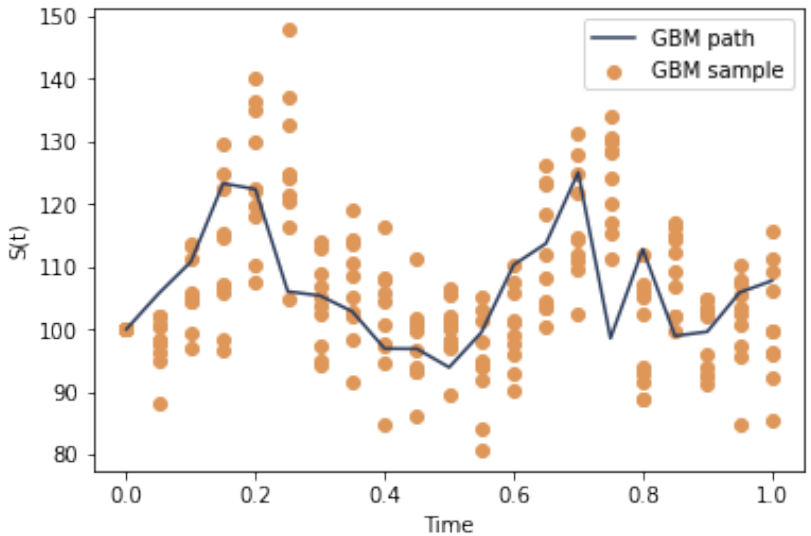
3D Dataset

- Nested Monte Carlo Simulation: two layers of simulation
  - The outer layer:
    $$S(t_i)_1^j = f(S(t_{i-1})_1^j), \quad i = 1, \ldots, m \text{ and } j = 1, \ldots, n.$$
  - The inner layer:
    $$S(t_i)_k^j = f(S(t_{i-1})_1^j), \quad k = 1, \ldots, d.$$



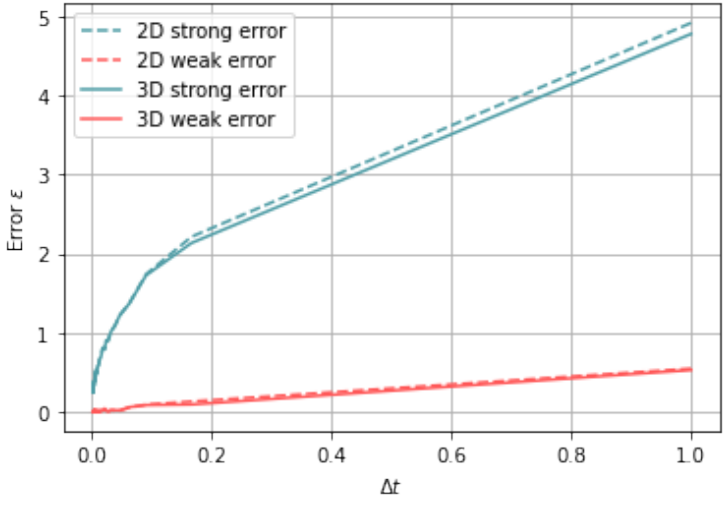
3D Dataset

- Nested Monte Carlo Simulation: two layers of simulation
  - The outer layer:
    $$S(t_i)_1^j = f(S(t_{i-1})_1^j), \quad i = 1, \ldots, m \text{ and } j = 1, \ldots, n.$$
  - The inner layer:
    $$S(t_i)_k^j = f(S(t_{i-1})_1^j), \quad k = 1, \ldots, d.$$
- For example, the GBM samples under the Euler discretization,
$$f(s_i) := s_{i-1} + \mu s_{i-1}\Delta t + \sigma s_{i-1}\Delta t Z.$$

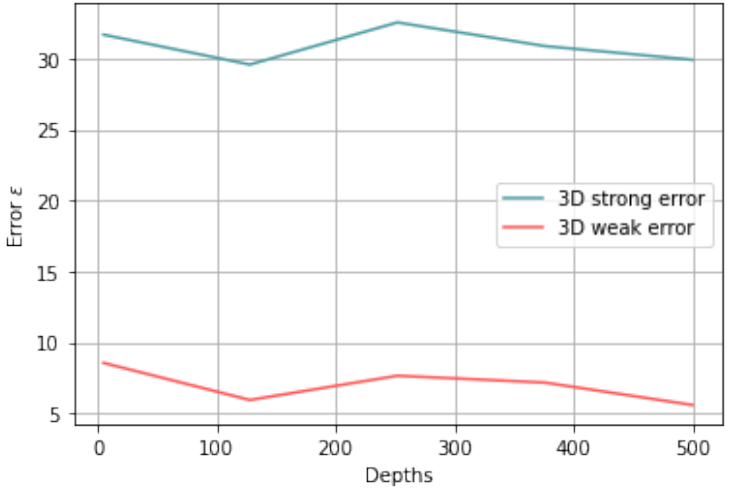


3D Dataset

- Comparison: 3D dataset and 2D dataset



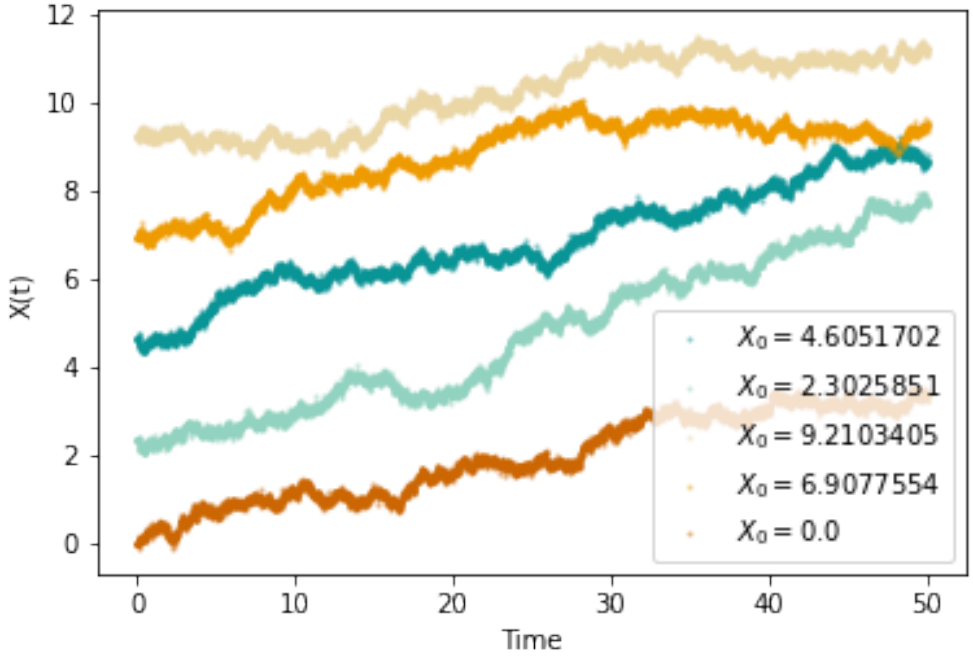
Fixed $T = 1$ *and depth* $= 10$.



Fixed $T = 1$ *and* $\Delta t = 0.5$.

3D Dataset

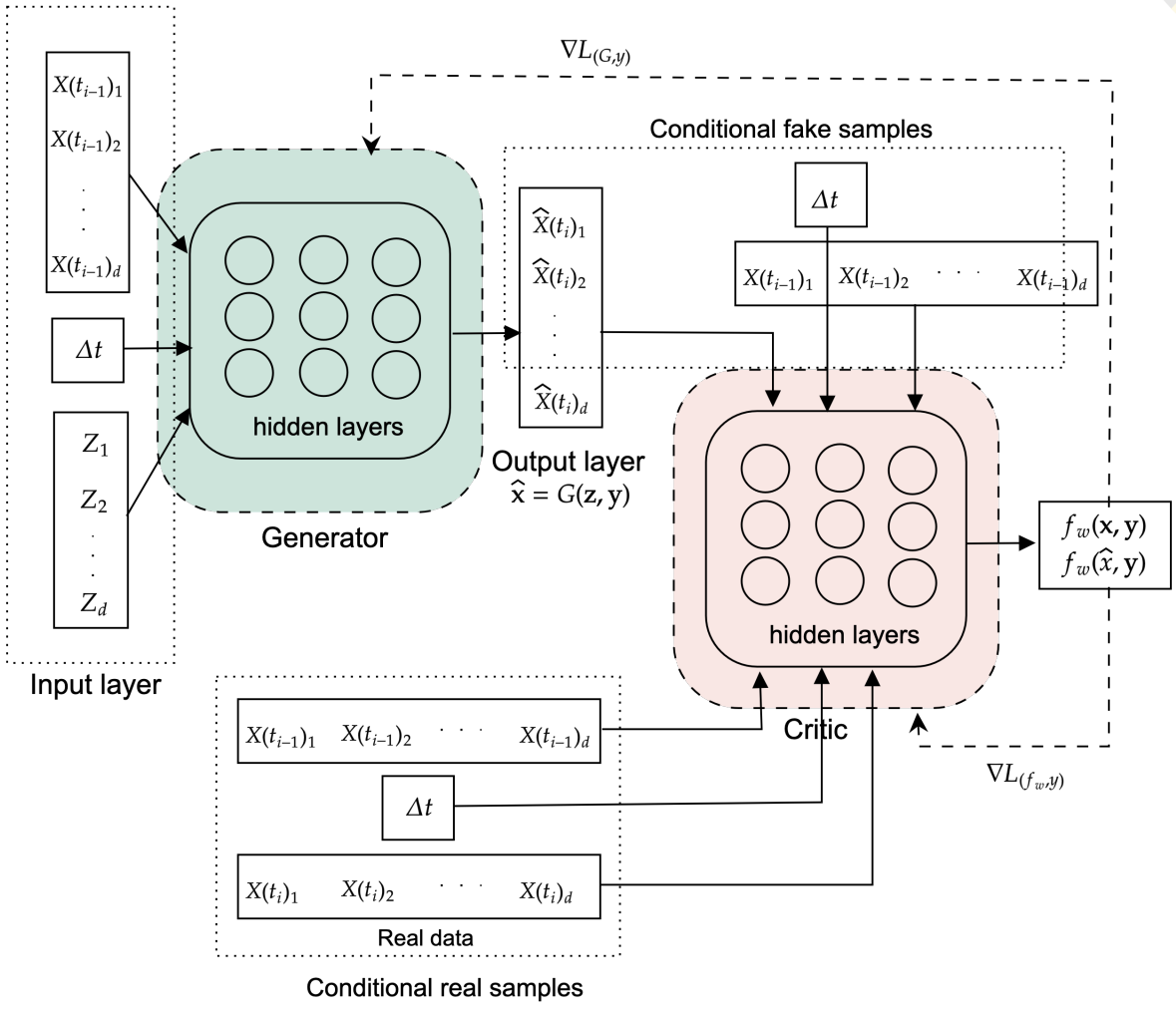
- Training dataset:
  A 3D dataset with 10000 paths, 500 steps and 10 depths sampling from the GBM log-price dynamics.



The initial states are 0, $\log 10$, $\log 100$, $\log 1000$ and $\log 10000$.

SDE-WGAN

Adaptation of SDE-GAN



SDE-WGAN

Architecture

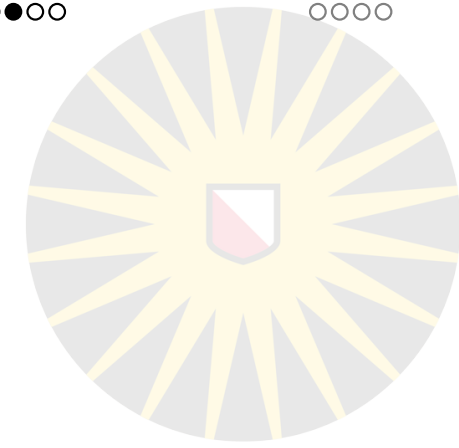


Table B.1: Generator architecture

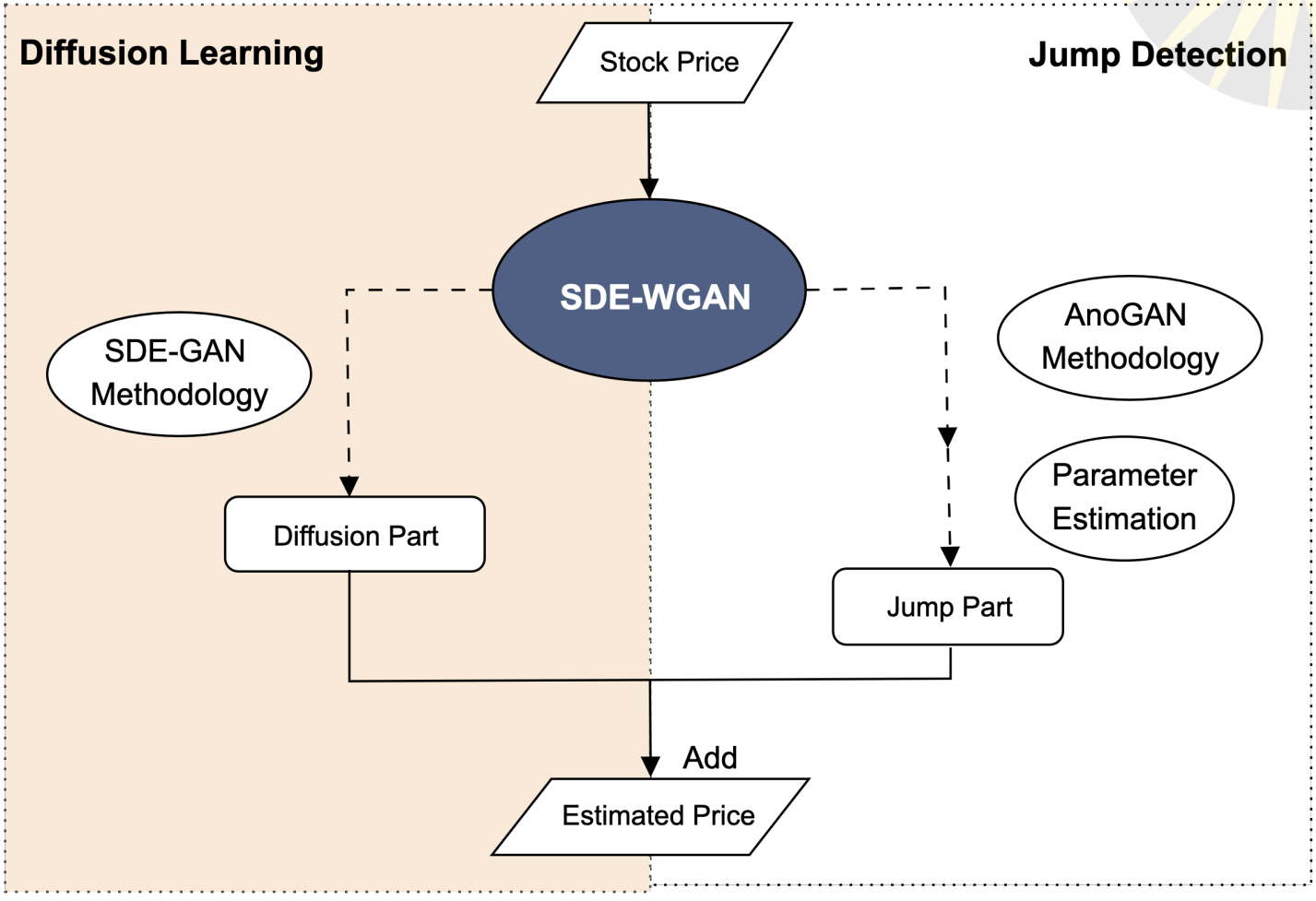
Optimiser	RMSProp $lr = 0.0003$, $\beta = 0.99$	
Layer	Nodes	Activation
Input layer	$3d$	LeakyReLU, negative slope=0.1
Hidden 1	100	LeakyReLU, negative slope=0.1
Hidden 2	100	LeakyReLU, negative slope=0.1
Hidden 3	100	LeakyReLU, negative slope=0.1
Hidden 4	100	LeakyReLU, negative slope=0.1
Output layer	d	None

Table B.2: Critic architecture

Optimiser	RMSProp lr = 0.0003, $\beta = 0.99$	
Layer	Nodes	Activation
Input layer	$3d$	LeakyReLU, negative slope=0.1
Hidden 1	100	LeakyReLU, negative slope=0.1
Hidden 2	100	LeakyReLU, negative slope=0.1
Hidden 3	100	LeakyReLU, negative slope=0.1
Hidden 4	100	LeakyReLU, negative slope=0.1
Output layer	1	None

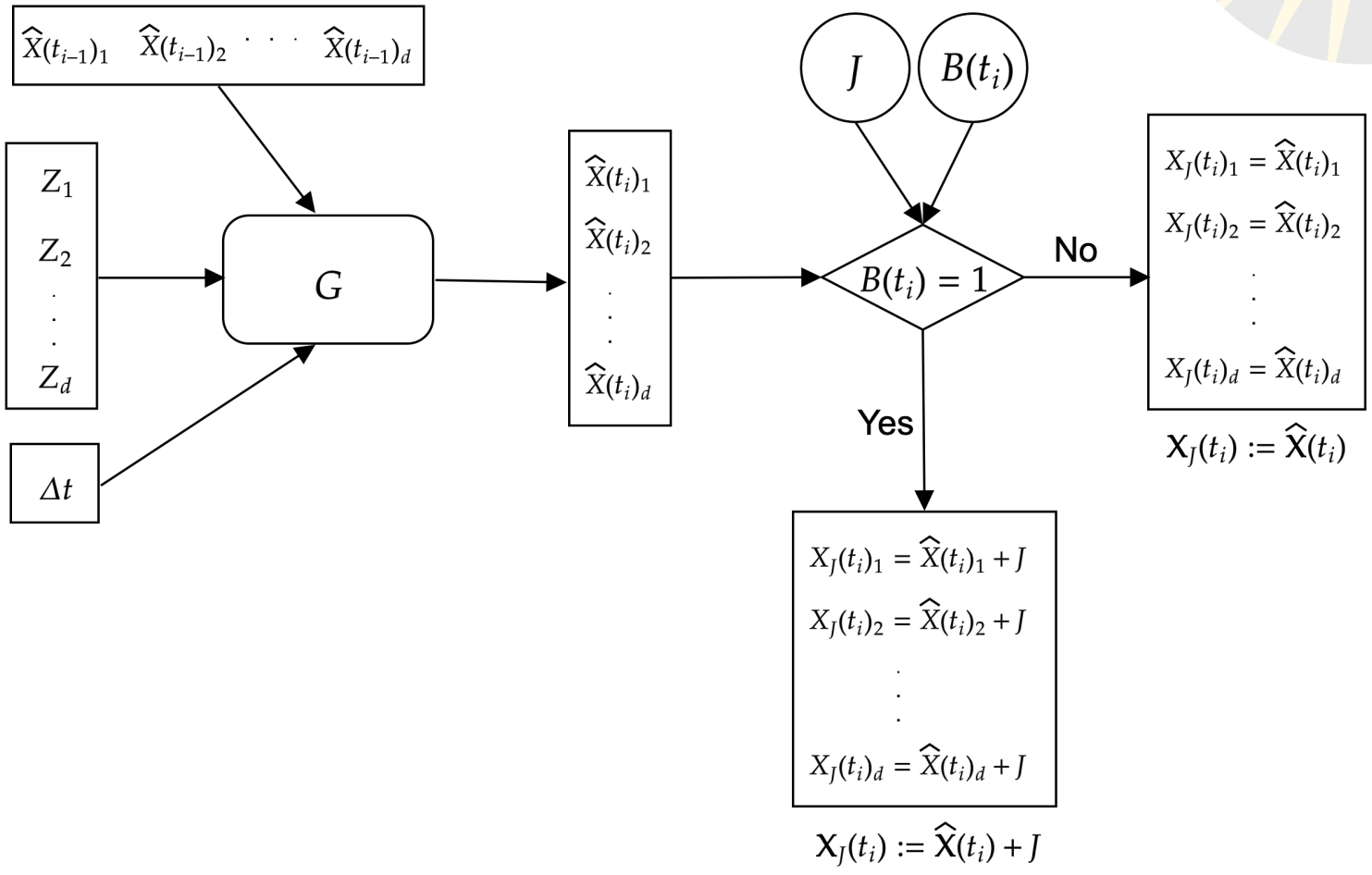
- Generator: 100-100-100-100
- Critic: 100-100-100-100
- Epochs: 100
- Batch size: 100
- Critic iteration: 9
- Gradient penalty λ: 60

Diffusion Learning



Diffusion Learning

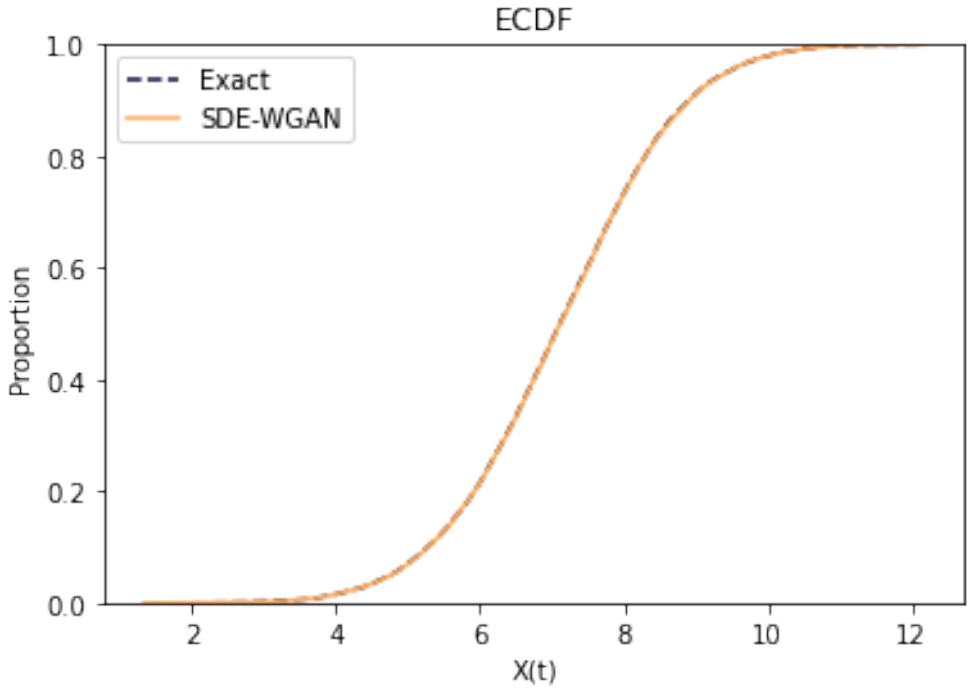
Jump-Diffusion Path Simulation



Diffusion Learning

Results

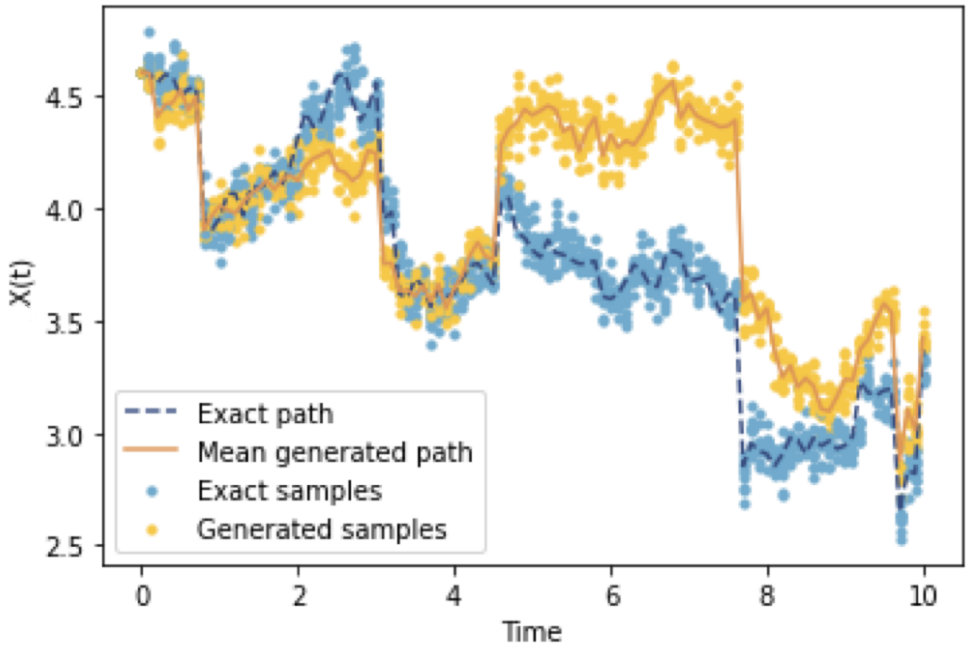
- Distribution approximation
- KS metric $= 0.003110$ with p-value 0.72, $W_1 = 0.006066$.



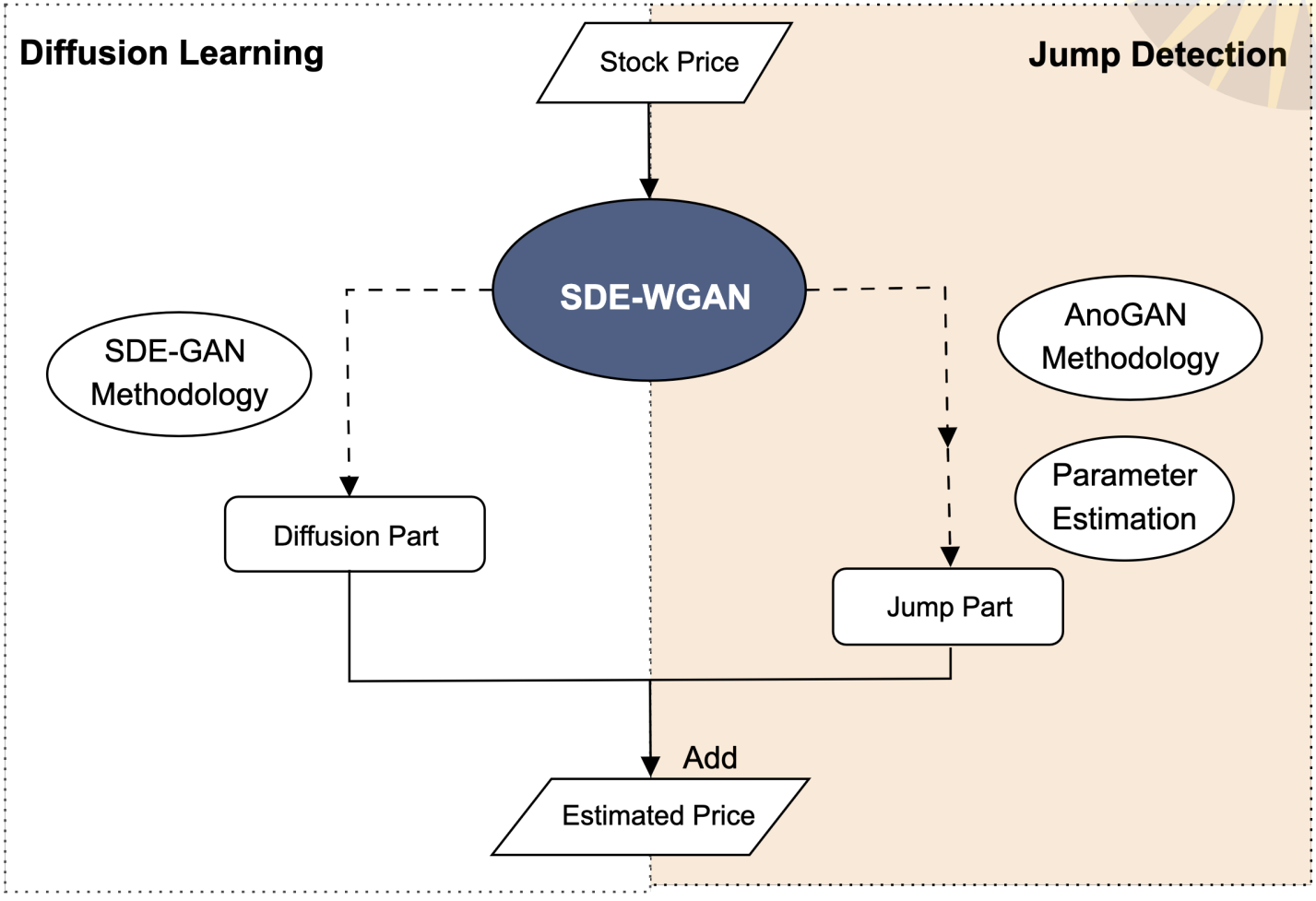
Diffusion Learning

Results

- Jump-diffusion path simulation

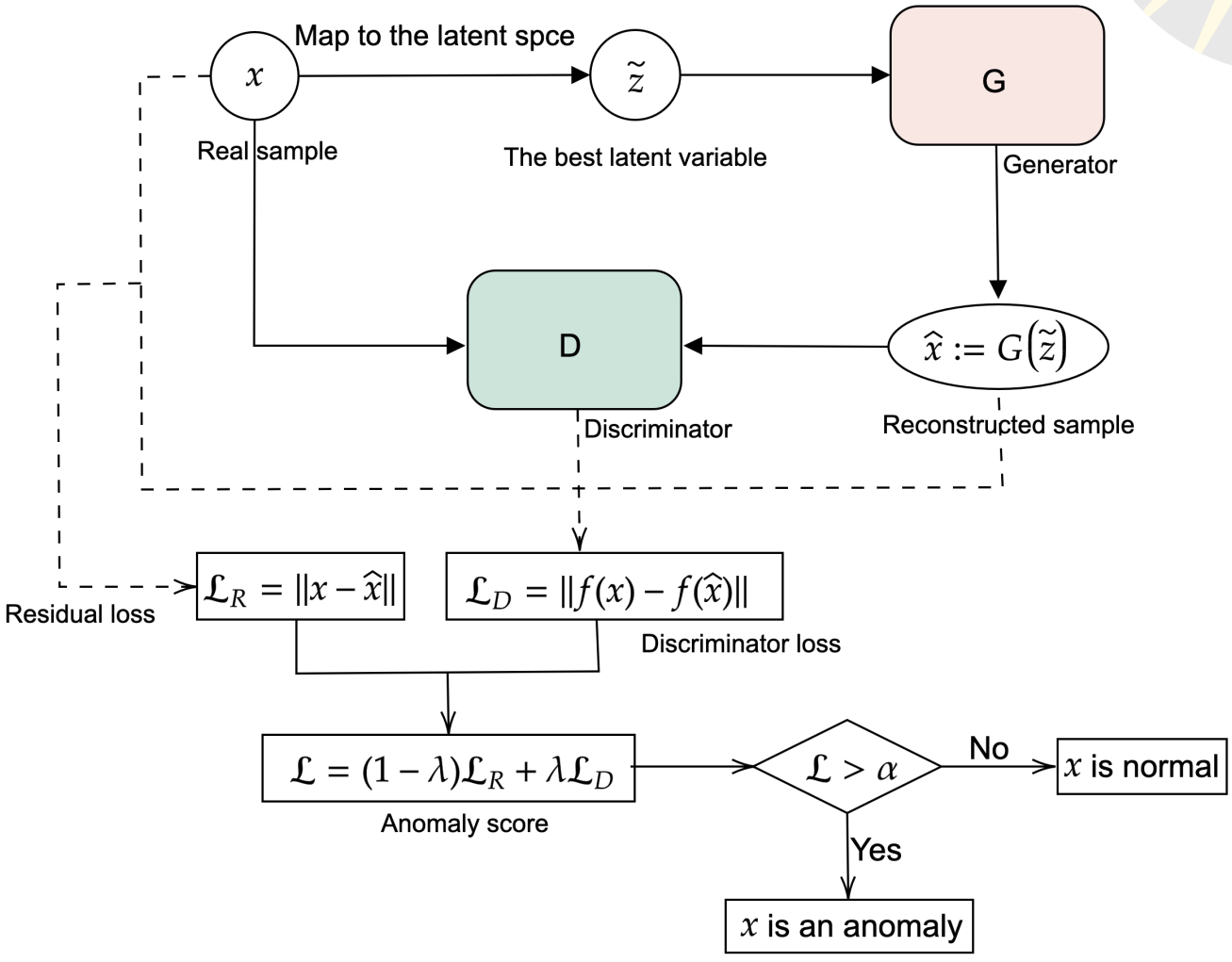


Jump Detection



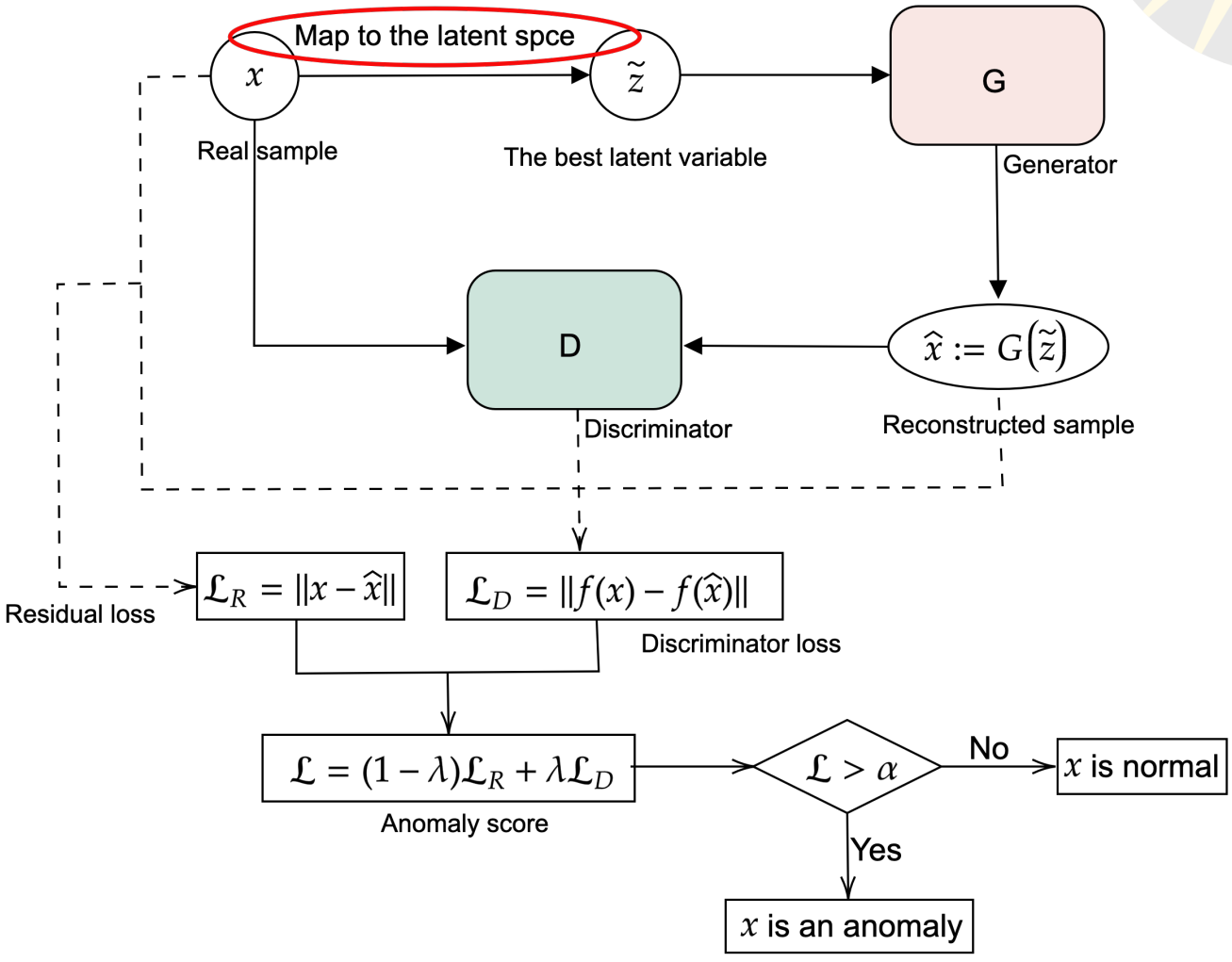
Jump Detection

AnoGAN Methodology



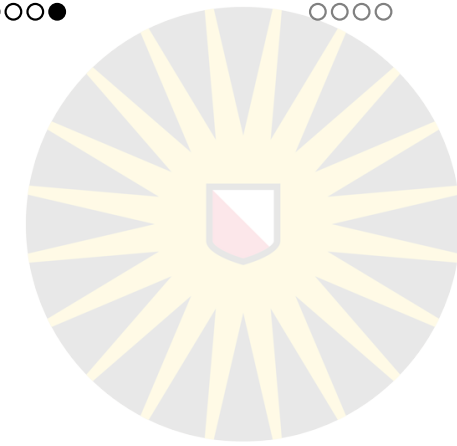
Jump Detection

AnoGAN Methodology



Jump Detection

AnoGAN Methodology: Mapping to the Latent Space



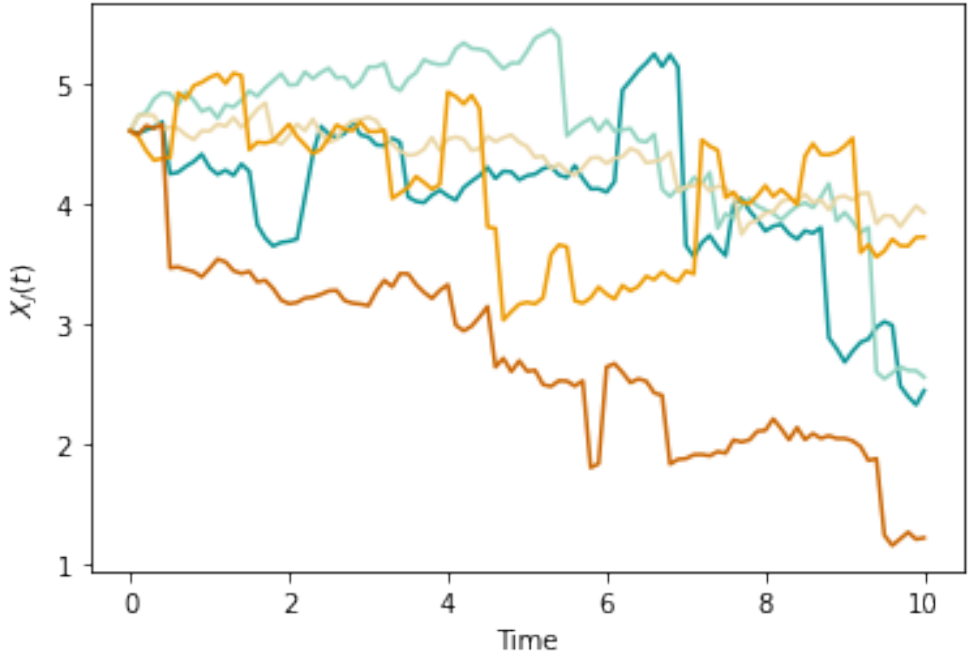
- The GAN do not have the inverse map: $G^{-1}(\mathbf{x}) : \mathbf{x} \mapsto \mathbf{z}$.
- Find a point $\mathbf{z}$ in the latent space such that $G(\mathbf{z})$ is visually most similar to $\mathbf{x}$.
- Use a gradient descent method, for example the Adam, to minimise the loss function $\mathcal{L}$:

$$\mathcal{L} = \mathcal{L}_R + \mathcal{L}_D$$

Jump Detection

Detecting Jump Instances

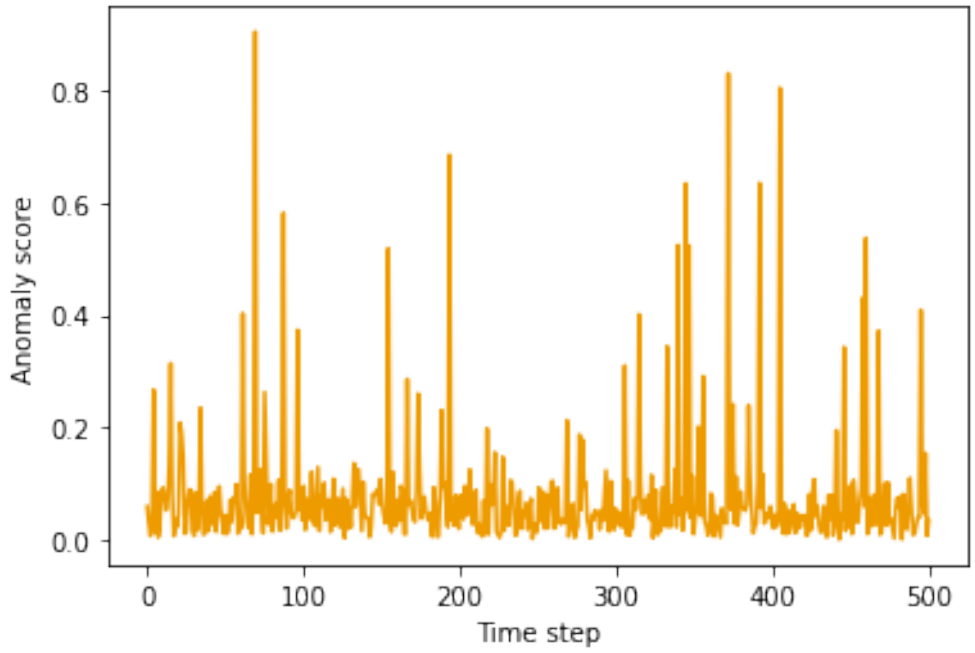
- Assumption: No jump occurs at the initial state $X(0)$.
- Test path: A path consists of five Merton's jump-diffusion paths, with parameters $x(0) = \log 100$, $\mu = 0.02$, $\sigma = 0.5$, $\Delta t = 0.1$, $\lambda_p = 1$, $\mu_J = -0.2$, $\sigma_J = 0.5$.



Jump Detection

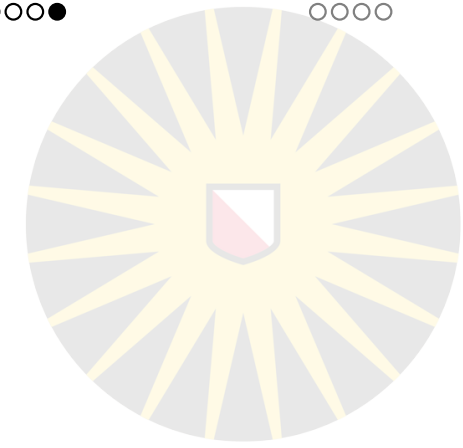
Results

- The anomaly scores of 500 non-initial states, with anomaly score coefficient $\lambda = 0.4$.
- We set the threshold $\alpha = 0.2$.



Jump Detection

Results



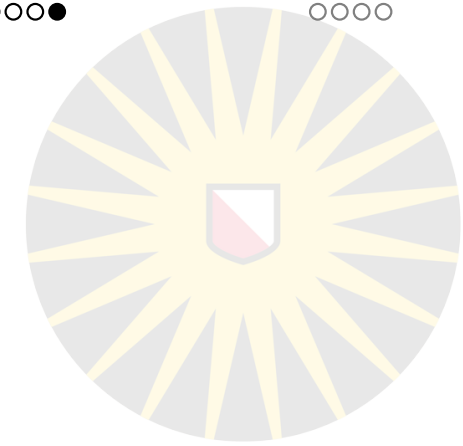
- The detection results:

	Normal data	Jumps
Normal data	454	0
Jumps	13	44

- Accuracy: 97.40%
- Recall: 97.22%
- Precision: 100%
- F1 score: 98.59

Jump Detection

Parameters Estimation

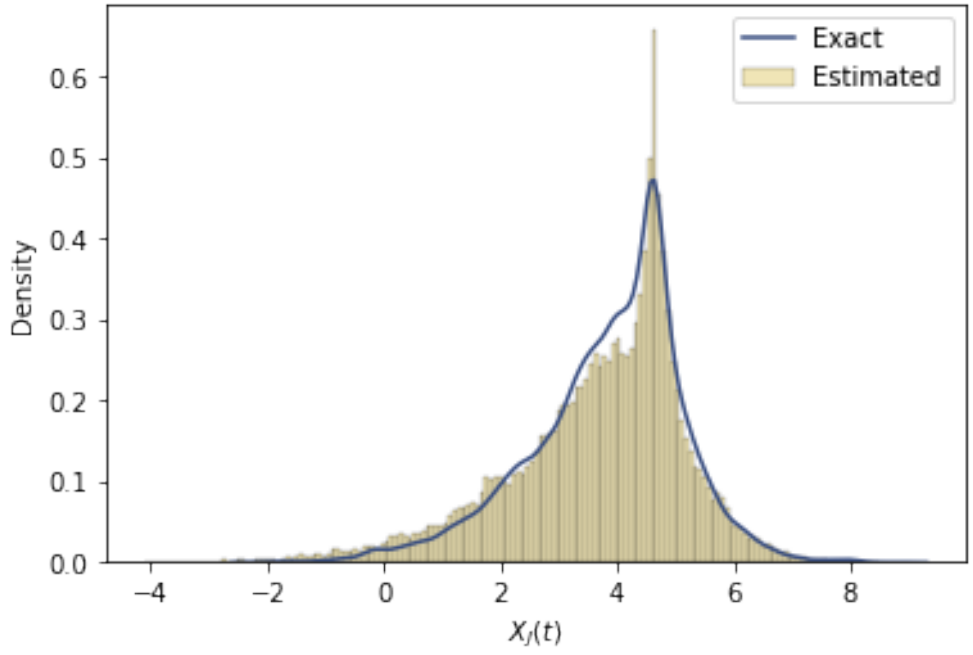


- The detected jump magnitude is the difference between the reconstructed state and the actual state.
- Use the MLE to estimate λ_p, μ_J and σ_J, based on the detected jump instances.
- $\hat{\lambda}_p = \frac{K}{t_K}$
- $\hat{\mu}_J = \frac{1}{K} \sum_{k=1}^{K} J_k$
- $\hat{\sigma}_J^2 = \frac{1}{K} \sum_{k=1}^{K} (J_k - \hat{\mu}_J)^2$

Jump Detection

Results

- $\hat{\lambda}_p = 0.715834 \qquad (\lambda_p = 1.0)$
- $\hat{\mu}_J = -0.324300 \qquad (\mu_J = -0.2)$
- $\hat{\sigma}_J^2 = 0.638275 \qquad (\sigma_J = 0.5)$



Discussion

Robustness

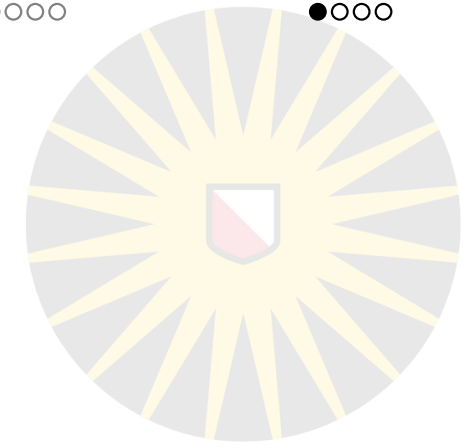
- The robustness of the diffusion learning part

Δt	μ	σ	KS	KS p-value	W_1
1.0	0.05	0.2	0.002700	0.858293	0.014771
0.5	0.05	0.2	0.002310	0.951809	0.010555
0.01	0.05	0.2	0.005040	0.157102	0.004024
0.001	0.05	0.2	0.003090	0.725023	0.000633
0.1	0.1	0.2	0.003510	0.567733	0.004405
0.1	-1.0	0.2	0.003190	0.687738	0.005754
0.1	0.05	1.0	0.002830	0.816995	0.025889

- The SDE-WGAN is robust to various diffusion parameter settings.

Discussion

Robustness



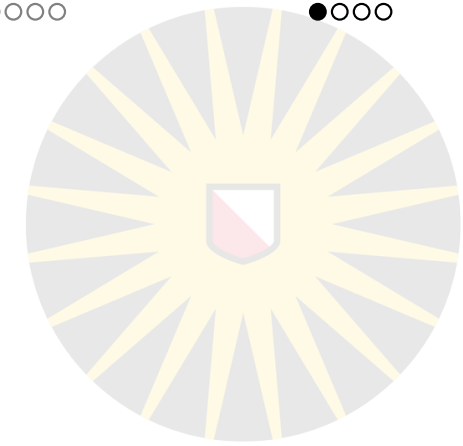
- The robustness of the jump detection part

Table 6.7: The jump detection results with respect to different jump parameters.

	λ_p	μ_J	σ_J	Confusion matrix	F1 score
Exact	1	-1.0	0.5	$\begin{pmatrix} 447 & 3 \\ 3 & 47 \end{pmatrix}$	99.44
Estimated	1.099436	-0.979969	0.527999		
Exact	1	0.0	0.5	$\begin{pmatrix} 444 & 0 \\ 21 & 35 \end{pmatrix}$	97.69
Estimated	0.824069	0.009215	0.651517		
Exact	1	0.0	0.2	$\begin{pmatrix} 432 & 3 \\ 44 & 21 \end{pmatrix}$	94.84
Estimated	0.546952	-0.034642	0.328174		
Exact	1	1.0	0.5	$\begin{pmatrix} 440 & 0 \\ 6 & 54 \end{pmatrix}$	99.32
Estimated	1.328758	1.192697	0.426563		

Discussion

Robustness



- The robustness of the jump detection part

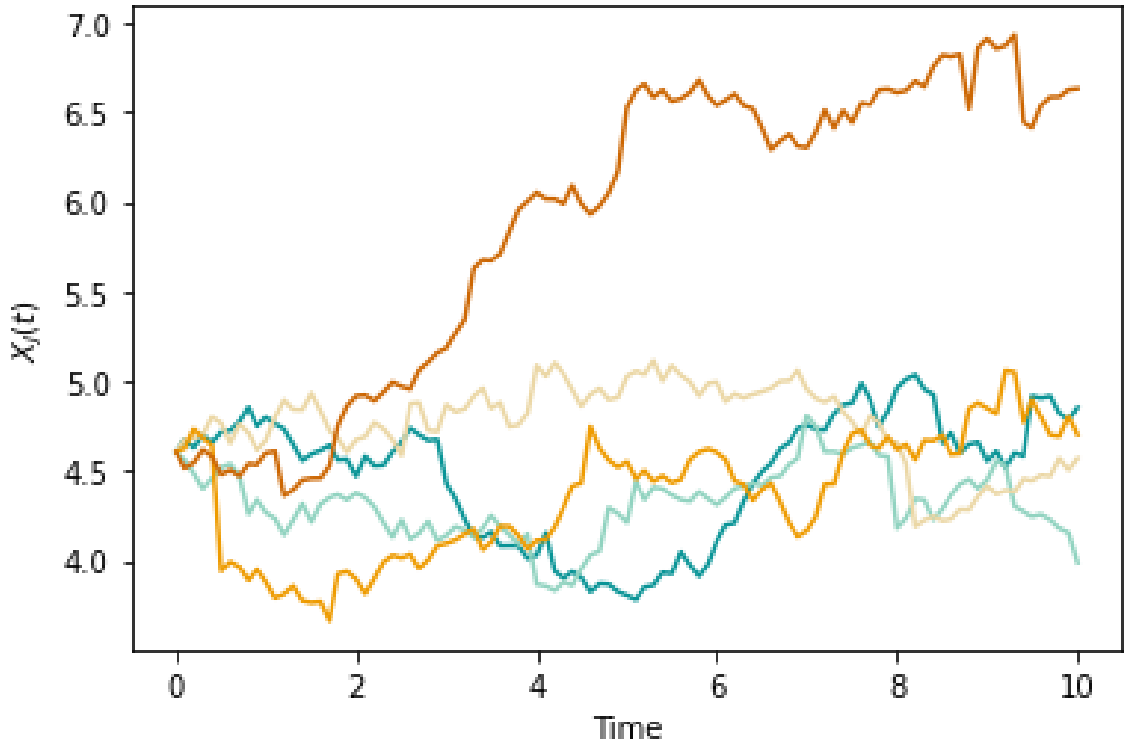
Table 6.7: The jump detection results with respect to different jump parameters.

	λ_p	μ_J	σ_J	Confusion matrix	F1 score
Exact	1	-1.0	0.5	$\begin{pmatrix} 447 & 3 \\ 3 & 47 \end{pmatrix}$	99.44
Estimated	1.099436	-0.979969	0.527999		
Exact	1	0.0	0.5	$\begin{pmatrix} 444 & 0 \\ 21 & 35 \end{pmatrix}$	97.69
Estimated	0.824069	0.009215	0.651517		
Exact	1	0.0	0.2	$\begin{pmatrix} 432 & 3 \\ 44 & 21 \end{pmatrix}$	94.84
Estimated	0.546952	-0.034642	0.328174		
Exact	1	1.0	0.5	$\begin{pmatrix} 440 & 0 \\ 6 & 54 \end{pmatrix}$	99.32
Estimated	1.328758	1.192697	0.426563		

Discussion

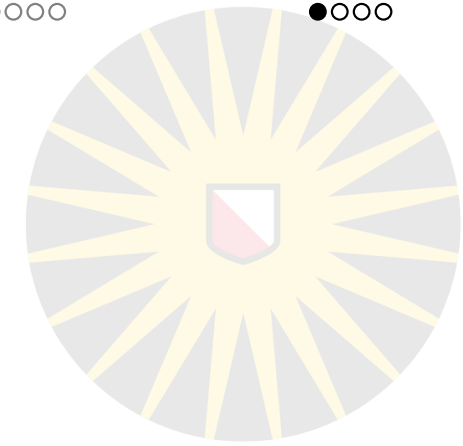
Robustness

- The jump-diffusion paths to test:



Discussion

Robustness



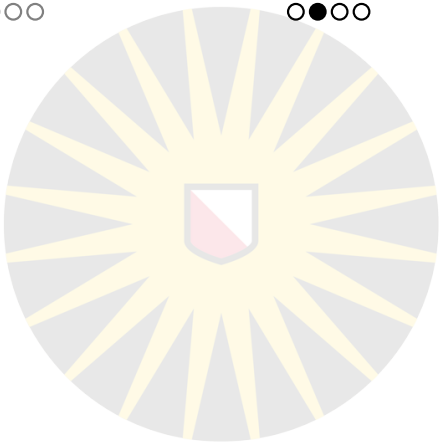
- The robustness of the jump detection part

Table 6.7: The jump detection results with respect to different jump parameters.

	λ_p	μ_J	σ_J	Confusion matrix	F1 score
Exact	1	-1.0	0.5	$\begin{pmatrix} 447 & 3 \\ 3 & 47 \end{pmatrix}$	99.44
Estimated	1.099436	-0.979969	0.527999		
Exact	1	0.0	0.5	$\begin{pmatrix} 444 & 0 \\ 21 & 35 \end{pmatrix}$	97.69
Estimated	0.824069	0.009215	0.651517		
Exact	1	0.0	0.2	$\begin{pmatrix} 432 & 3 \\ 44 & 21 \end{pmatrix}$	94.84
Estimated	0.546952	-0.034642	0.328174		
Exact	1	1.0	0.5	$\begin{pmatrix} 440 & 0 \\ 6 & 54 \end{pmatrix}$	99.32
Estimated	1.328758	1.192697	0.426563		

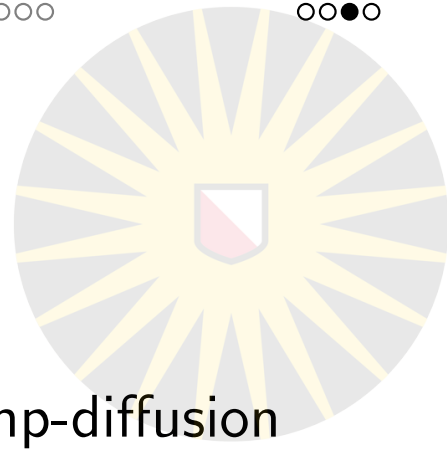
- The jump detection model has excellent performance when the jump magnitude is obvious.

Future Research



- BiGAN: automatic "inverse map";
- Automatic anomaly score threshold;
- Application on real market data;
- Advanced stochastic models, e.g. Hawkes processes.

Conclusion



- We propose a general framework to simulate jump-diffusion paths.
- The framework is GAN-based and combines anomaly detection methods.
- The framework can be divided into the diffusion learning part and the jump detection part.
- The SDE-WGAN is used to simulate the diffusion part.
- The anomaly detection method is applied to recognize the jumps. We can estimate the jump parameters and then simulate the jump part based on the detected jump instances.

References

[1] Goodfellow, Ian, et al. *Generative adversarial networks*. Communications of the ACM 63.11 (2020): 139-144.

[2] Kou, Steve G. *Jump-diffusion models for asset pricing in financial engineering*. Handbooks in operations research and management science 15 (2007): 73-116.

[3] van Rhijn, Jorino, et al. *Monte carlo simulation of sdes using gans*. Japan Journal of Industrial and Applied Mathematics (2022): 1-32.

[4] Schlegl, Thomas, et al. *Unsupervised anomaly detection with generative adversarial networks to guide marker discovery*. International conference on information processing in medical imaging. Springer, Cham, 2017.