Deep Generative AI for Portfolio Management

Jeongyeon Park[1], **Young Shin Kim**[2]

[1]Department of Applied Mathematics and Statistics, Stony Brook University
[2]College of Business, Stony Brook University

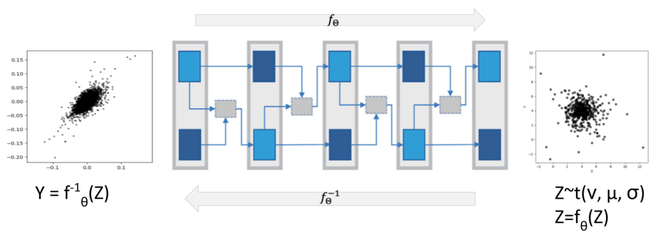
Abstract

- Variance treats big gains and big losses the same. That means it doesn't tell us much about downside risk. CVaR, on the other hand, looks at the average loss in the worst-case scenarios, so it gives a clearer picture of extreme risks. It also works well with skewed data while variance assumes everything is balanced and symmetric, which is not true for most financial returns. Since Gaussian distributions fail to capture heavy tails and extreme risk, we use deep generative models to learn flexible return distributions. These models enable more accurate CVaR-based portfolio optimization by better modeling the tail behavior of asset returns.

Models

Student-t Glow

- We construct a generative model following the Glow architecture (Kingma and Dhariwal, 2018). We replace the standard Gaussian prior with a base distribution composed of independent Student-*t* distributions that have learnable parameters: degrees of freedom ν, location μ, and scale σ to enhance the flexibility of model's output distribution. The overall transformation from latent variables to observed returns is built by stacking multiple flow steps, where each step consists of three invertible components: an ActNorm layer, an invertible linear transformation, and an affine coupling layer.



Flow Matching

- The Flow Matching model (Lipman *et al.*, 2023) generates a path of probability distributions that moves from a starting distribution p_0, which is a simple one like a Gaussian, to a target distribution p_1, which is more complex and comes from real data. This model introduces two key components: A time-varying probability density, $p : [0,1] \times \mathbb{R}^d \to \mathbb{R}^+$, which gives the likelihood of each point x at time t. A time-dependent vector field, $v : [0,1] \times \mathbb{R}^d \to \mathbb{R}^d$, which tells us how each point in space moves over time. This vector field is used to build a time-varying transformation function called a flow, written as $\phi : [0,1] \times \mathbb{R}^d \to \mathbb{R}^d$. The flow is defined by an ordinary differential equation (ODE).

$$\frac{d}{dt}\phi_t(x) = v_t(\phi_t(x))$$
$$\phi_0(x) = x$$

For any probability path p_t, we define the vector field v_t generates the probability path p_t if its flow ϕ_t satisfies

$$X_t := \phi_t(X_0) \sim p_t$$

where $X_0 \sim p_0$. The goal of Flow Matching is to learn a vector field v_t so that its flow ϕ_t generates the sequence of probability distributions, starting from $p_0 = p$ and ending at $p_1 = q$. Given a target probability density path $p_t(x)$ and a vector field $v_t(x)$ that produces $p_t(x)$, we represent the vector field v_t using a neural network, $v_t(x;\theta)$. The Flow Matching(FM) loss function is then defined as follows:

$$\mathcal{L}_{FM}(\theta) = \mathbb{E}_{t,p_t(x)}\|v_t(x;\theta) - u_t(x)\|^2,$$

where $t \sim \mathcal{U}[0,1]$(uniform distribution), and $x \sim p_t(x)$. After training, we can get a sample from $X_1 \sim q$ by picking a sample from the starting distribution, $X_0 \sim p$, and solving the Ordinary Differential Equation given by the vector field v_t. Since it is hard to calculate $\mathcal{L}_{FM}$, we use the Conditional Flow Matching(CFM) objective,

$$\mathcal{L}_{CFM}(\theta) = \mathbb{E}_{t,x_1\sim q,x_t\sim p_t(\cdot|x_1)}\|v_t(x;\theta) - u_{t|1}(x|x_1)\|^2.$$

Score-Based Model

- We propose a Score-Based Generative Model (De Bortoli *et al.*, 2023). Let $p_0 = p_{data}$ be the data distribution and p_{prior} be a prior density. We consider a Markov chain that evolves according to transition densities $p_{k+1|k}$ for each $k \in 0,\dots,N-1$. Then, for a given sequence $x_{0:N} = \{x_k\}_{k=0}^N \in (\mathbb{R}^d)^{N+1}$, the joint density can be written as

$$p(x_{0:N}) = p_0(x_0)\prod_{k=0}^{N-1} p_{k+1|k}(x_{k+1}|x_k).$$

The joint density admits the backward decomposition

$$p(x_{0:N}) = p_N(x_N)\prod_{k=0}^{N-1} p_{k|k+1}(x_k|x_{k+1}),$$
$$p_{k|k+1}(x_k|x_{k+1}) = \frac{p_k(x_k)p_{k+1|k}(x_{k+1}|x_k)}{p_{k+1}(x_{k+1})}$$

We define the forward transition density as:

$$p_{k+1|k}(x_{k+1}|x_k) = \mathcal{N}(x_{k+1}; x_k + \gamma_{k+1}f(x_k), 2\gamma_{k+1}I)$$

with drift $f : \mathbb{R}^d \to \mathbb{R}^d$ and stepsize $\gamma_{k+1} > 0$. Assuming $p_k \approx p_{k+1}$, applying a Taylor expansion to $\log\ p_{k+1}$ around x_{k+1}, and using the approximation $f(x_k) \approx f(x_{k+1})$, we arrive at the following approximation:

$$\begin{aligned} p_{k|k+1}(x_k|x_{k+1}) &= p_{k+1|k}(x_{k+1}|x_k)\exp\big[\log p_k(x_k) \\ &\quad - \log p_{k+1}(x_{k+1})\big] \\ &\approx \mathcal{N}\big(x_k; x_{k+1} - \gamma_{k+1}f(x_{k+1}) \\ &\quad + 2\gamma_{k+1}\nabla\log p_{k+1}(x_{k+1}), 2\gamma_{k+1}I\big). \end{aligned}$$

In practice, this approximation is valid when $\|x_{k+1} - x_k\|$ is small, which can be achieved by choosing a small enough step size γ_{k+1}. We also have the relation $p_{k+1}(x_{k+1}) = \int p_k(x_k)p_{k+1|k}(x_{k+1}|x_k)dx_k$ and from this, we can show that $\nabla\log\ p_{k+1}(x_{k+1}) = \mathbb{E}_{p_{k|k+1}}[\nabla_{x_{k+1}}\log\ p_{k+1|k}(x_{k+1}|X_k)]$. Using this idea, we can estimate the score function with a neural network by learning an approximation $s_\theta(k,x_k) \approx \nabla\log p_k(x_k)$. To do this, we use the following loss function

$$\mathcal{L}_{score} = \sum_{k=1}^{N}\mathbb{E}_{p_{k-1,k}}[\|s_\theta(k,X_k) - \nabla_{x_k}\log\ p_{k|k-1}(X_k|X_{k-1})\|^2].$$

In addition, we use another loss function used to minimize the negative log-likelihood of x_N:

$$\mathcal{L}_{nll} = -\mathbb{E}_{x_N\sim p_N}[\log p_N(x_N)].$$

where p_N is a standard normal distribution. Then we train the neural networks on parameters that minimizes $\mathcal{L}_{score} + \mathcal{L}_{nll}$. After that, we generate a sample X_0 by starting from $X_N \sim p_{prior}$ and applying the reverse update based on the approximation in equation

$$\begin{aligned} X_k &= X_{k+1} - \gamma_{k+1}f(X_{k+1}) + 2\gamma_{k+1}s_{\theta^*}(k+1, X_{k+1}) \\ &\quad + \sqrt{2\gamma_{k+1}}Z_{k+1}, \\ Z_{k+1} &\overset{i.i.d}{\sim} \mathcal{N}(0,I). \end{aligned}$$

Portfolio Optimization

Data

- Train models with 6367 daily log-returns standardized to zero mean and unit variance, consisting of 9 sector ETFs: XLB (Materials), XLE (Energy), XLF (Financials), XLI (Industrials), XLK (Technology), XLP (Consumer Staples), XLU (Utilities), XLV (Health Care), and XLY(Consumer Discretionary).

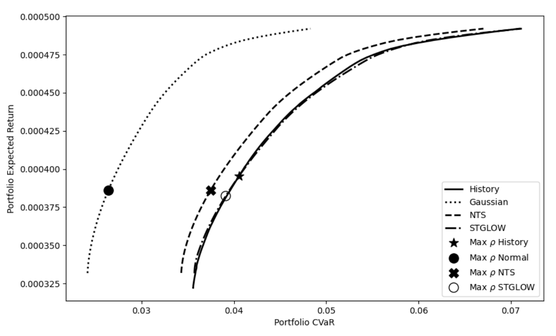
Training

- **Student-t Based Glow model:** We use batch size of 512, 9 stacks of overall transformation, 5500 training steps, and 6 hidden layers with 64 hidden nodes and $\nu = 4$, $\mu = (0,\dots,0) \in \mathbb{R}^9$, and $\sigma = (1,\dots 1) \in \mathbb{R}^9$ for initial parameters of base independent Student-t distributions. Adam optimizer with an initial learning rate of 10^{-4}, and the Cosine Annealing Warm Restarts learning rate scheduler are used.
- **Flow Matching model:** We use a batch size of 512, 2500 training steps, 6 hidden layers with 256 hidden nodes. For p_0, we choose $\nu = 4$, $\mu = (0,\dots,0) \in \mathbb{R}^9$, and $\sigma = (1,\dots 1) \in \mathbb{R}^9$ for initial parameters of independent Student-t distributions, for p_0. Adam optimizer with a starting learning rate of 10^{-4}, and the Cosine Annealing Warm Restarts scheduler are used.
- **Score-Based Generative model:** We use a 9×9 matrix with every entry being learnable and initialized with diagonal matrices whose every diagonal entry is 0.01, instead of γ_{k+1} and a batch size of 512, 500 training steps, $N = 8$, and 6 hidden layers with 256 hidden nodes. For p_N, we use a standard normal distribution. Adam optimizer with an initial learning rate of 10^{-4}, and the StepLR scheduler are used.

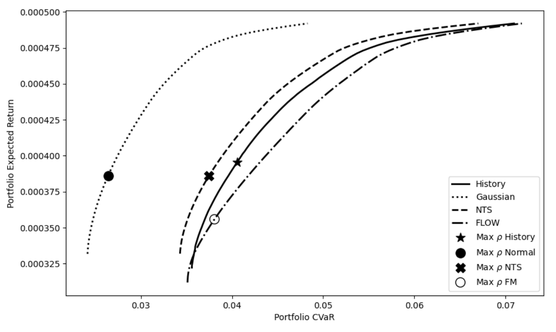
Simulation and Optimzation

$$\begin{aligned} &\min_w \text{CVaR}_\alpha(R(w)) \\ &\text{subject to } \mathbb{E}[R(w)] \geq \mu \\ &\sum_{n=1}^{9} w_n = 1 \\ &w_n \geq 0 \text{ for all } n \in \{1,2,\dots,9\} \end{aligned}$$

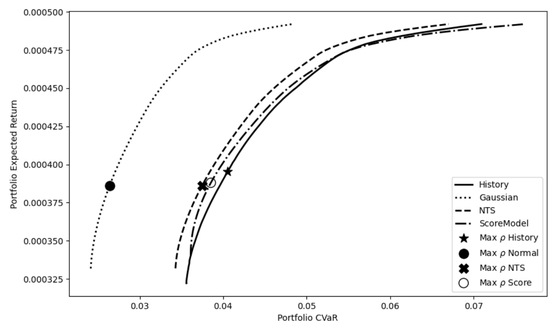
- $R(w) = \sum_{n=1}^9 w_n R_n$, R is a return vector.
- Solve for 200,000 synthetic return samples
- Models : Historical, Gaussian, NTS, Generative AI
- $\rho = \langle w, \mathbb{E}[R]\rangle / \text{CVaR}_\alpha(R(w))$ where we assume $\mathbb{E}[R]$ is equal to historical mean vector.



(a) Efficient frontier from Student-t-based Glow model.



(b) Efficient frontier from Flow Matching model.



(c) Efficient frontier from Score-Based Generative Model.

References

De Bortoli, V., Thornton, J., Heng, J., and Doucet, A. (2023). Diffusion schrödinger bridge with applications to score-based generative modeling. *arXiv preprint arXiv:2302.05150*. URL https://arxiv.org/abs/2302.05150.

Kingma, D. P. and Dhariwal, P. (2018). Glow: Generative flow with invertible 1x1 convolutions. In *Advances in Neural Information Processing Systems*. Curran Associates, Inc., vol. 31.

Lipman, Y., Albergo, M., Tallec, C., Musso, T., Kyle, E., Cohen, T., Cotter, A., Swaminathan, A., Vemuri, S., Pope, P., *et al.* (2023). Flow matching for generative modeling. In *International Conference on Machine Learning*. PMLR.