Deep-Learning Solution to Portfolio Selection with Serially-Dependent Returns

Zhiyi Da, Haochen Jiang

December 5, 2022

1 Introduction

Using deep learning and machine learning to solve real-world problems has been dramatically accelerated by the success of AlphaGo over humans in the game of Go (Tsang and Wong, 2020). Many mathematical finance problems have been recently solved using deep learning and machine learning, such as prediction with limit-order information (Sirignano, 2019), analysis of least-squares Monte Carlo method for American option pricing (Zanger, 2018), stochastic control problems (Bachouch et al., 2021) and Portfolio Optimization (Ban et al., 2018).

Tsang and Wong (2020) aims to propose a deep neutral network (DNN) architecture and numerical algorithm for multi-period portfolio optimization. Their deep learning solution is complementary to the literature as mentioned earlier by relaxing regularity conditions on the objective function and allowing for popular econometric models. Practical portfolio selection has to overcome the challenges arising from high-dimensionality, flexible modeling for risky asset returns and investment constraints. The literature has great success in developing theoretical results for models with serially-independent asset returns (Li and Ng, 2000).

Tsang and Wong (2020) focuse on two classes of well-known time series models: autoregressive (AR) and the generalized autoregressive conditional heteroskedasticity (GARCH) models. The former describes serial-dependence through the drift terms of risky asset returns while the latter through the innovation terms. Their goal is to solve the high-dimensional multi-period portfolio problem in discrete-time subject to the bounding constraints on the control.

This paper is divided into the following sections. Section 2 introduces the problem formulation and notations of feedforward neural network. Section 3 details of the deep neutral network architecture for portfolio optimization. Section 4 provides the numerical and empirical examples with their deep neutral network framework for a high-dimensional portfolio. In Section 5, a conclusion is provided.

2 Problem Formulation

Tsang and Wong (2020) claimed that the classic Markov decision process (MDP) can be defined as follows in a filtered probability space $\left(\Omega, P, \{\mathcal{F}_k\}_{k\geq 0}\right)$, where k belongs to the integer set:

$$\begin{aligned} V_0(s_0) &= \min_{g_0,\ldots,g_{K-1}\in\mathcal{U}(\cdot)} \mathbb{E}\left[\Psi(s_K) \mid \mathcal{F}_0\right], \\ \text{s.t.} \quad s_{k+1} &= h\left(s_k, g_k(s_k), \eta_k\right), k = 0, 1, \ldots, K-1, \end{aligned} \tag{1}$$

where

$h(\cdot,\cdot,\cdot) : \mathcal{S} \times D \times \Omega \to \mathcal{S}$ is the transition function;
$\Psi(\cdot) : \mathcal{S} \to \mathbb{R}$ is the objective function;
The $\mathcal{F}_k$-measurable s_k is defined in the Markovian state space $\mathcal{S}$;
$\eta_k \in \Omega$ is a $\mathcal{F}_{k+1}$-measurable random variable;
$g_k : \mathcal{S} \to D$ is a feedback control function;
$\mathcal{U}(\cdot)$ is the set of admissible control functions satisfying some regularity conditions.

2.1 Portfolio Optimization

Tsang and Wong (2020) mentioned that Multi-period portfolio optimization problems can be grouped under the MDP problem (1). In particular, An interesting MDP problem is the expected utility maximization (EUM) in discrete-time settings. The problem can be expressed as follows:

$$\begin{aligned} V_0(\cdot) &= \max_{\{g_k\}_{k=0}^{K-1}} E\left[U\left(W_K\right) \mid \mathcal{F}_0\right], \\ \text{s.t.} \quad W_{k+1} &= W_k\left(g_k^T R_k + R_f\right), k = 0, 1, \ldots, K-1, \end{aligned} \tag{2}$$

where

$U(\cdot) : \mathbb{R} \to \mathbb{R}$ is the utility function on the terminal wealth;
The $\mathcal{F}_k$-measurable $g_k \in D \subseteq \mathbb{R}^p$ is the investment policy on p risky assets at time k while D is the constrained investment opportunity set;
$R_k \in \mathbb{R}^p$ is the excess returns of risky assets between the time points k and $k+1$;
R_f is the constant risk-free return for each period.
When $\{R_k\}$ is serially-independent, W_k represents the state vector in (1).
When $\{R_k\}$ has serially-dependence, W_k is no longer Markovian in general and the state vector s_k in (1) contains elements beyond W_k.
In both cases, we still have $\Psi\left(s_K\right) = -U\left(W_K\right)$.

Tsang and Wong (2020) aims to solve the EUM problem (2) with bounding constraints and serially-dependent returns. For purposes of more clearly illustrating their results, Tsang and Wong (2020) focus on the quadratic utility function $U\left(W_K\right) = -\left(W_K - \frac{\gamma}{2}\right)^2$ with AR(1) and/or the CCC-GARCH $(1,1)$ models for the asset returns. Furthermore, Tsang and Wong (2020) emphasize that there is the potential for their algorithm to be applied to the general utility functions as well as to more general return dynamics.

The AR(1) model specifies that

$$R_k = \alpha + AR_{k-1} + \epsilon_k, k = 0, 1, \ldots, K-1 \tag{3}$$

where $\alpha \in \mathbb{R}^p, A \in \mathbb{R}^{p \times p}$ and ϵ_k is identically and independently distributed (i.i.d.) p-dimensional random variables.

The CCC-GARCH(1,1) model is represented as follows.

$$\begin{aligned} R_{k,i} &= \mu_i + z_{k,i}, \\ z_{k,i} &= \Sigma_{k,i}\epsilon_{k,i}, \\ \Sigma_{k,i}^2 &= \omega_i + \alpha_i \Sigma_{k-1,i}^2 + \beta_i z_{k-1,i}^2, i = 1, \ldots, p, k = 0, 1, \ldots, K-1, \end{aligned} \tag{4}$$

where $\omega_i, \alpha_i, \beta_i \geq 0$ for $i = 1, \ldots, p$. $\mu = (\mu_1, \ldots, \mu_p) \in \mathbb{R}^p, z_k = (z_{k,1}, \ldots, z_{k,p}) \in \mathbb{R}^p$ and $\Sigma_k^2 = \left(\Sigma_{k,1}^2, \ldots, \Sigma_{k,p}^2\right)$. $\epsilon_k = (\epsilon_{k,1}, \ldots, \epsilon_{k,p})$ are *i.i.d.* random variables with constant correlation matrix Γ

which is positive definite and satisfies $\Gamma_{ii} = 1, \Gamma_{ij} = \Gamma_{ji} = \rho_{ij}$ and $\rho_{ij} \in [-1, 1]$. In fact, the CCC-GARCH(1,1) model can be viewed as p-univariate GARCH(1,1) model connected through the stationary correlation ρ_{ij}. As a result of its simplicity and simple parameter estimation process, CCC-GARCH(1,1) is one of the most widely used multivariate GARCH models.

In addition, Tsang and Wong (2020) investigate the relationship between EUM (2) and MDP (1) for different return dynamics. In Section 3, Tsang and Wong (2020) describe how a deep-learning solution is constructed for a general MDP in (1) to solve the EUM. A serially-independent return can be formulated as an equivalent MDP problem by defining $\Psi(s_K) = -U(W_K), s_k = W_k, \eta_k = R_k$ and $h(s_k, g_k(s_k), \eta_k) = W_k(g_k(s_k)^T R_k + R_f)$ in equation (1).

As an equivalent to the EUM (2) with serially-dependent returns, Tsang and Wong (2020) define two MDP problems. Assuming that the return follows the AR(1) in (3), the equivalent MDP problem can be formulated as follows:

$$\begin{aligned} V_0(s_0) &= \min_{g_0,\ldots,g_{K-1}\in\mathcal{U}(\cdot)} \mathbb{E}\left[-U(W_K) \mid \mathcal{F}_0\right] \\ \text{s.t.} \quad s_{k+1} &= \left(W_k\left(g_k(s_k)^T R_k + R_f\right), R_k\right), k = 0, 1, \ldots, K-1, \end{aligned} \tag{5}$$

where

$s_k \equiv (W_k, R_{k-1})$;
$\eta_k \equiv \epsilon_k$;
$R_k = \alpha + AR_{k-1} + \eta_k$.

Assuming that the return follows the CCC-GARCH(1,1) in (4), the equivalent MDP problem can be formulated as follows:

$$\begin{aligned} V_0(s_0) &= \min_{g_0,\ldots,g_{K-1}\in\mathcal{U}(\cdot)} \mathbb{E}\left[-U(W_K) \mid \mathcal{F}_0\right] \\ \text{s.t.} \quad s_{k+1} &= \left(W_k\left(g_k(s_k)^T R_k + R_f\right), \Sigma^2_{k+1}\right), k = 0, 1, \ldots, K-1, \end{aligned} \tag{6}$$

where

$s_k \equiv \left(W_k, \Sigma^2_k\right)$;
$\eta_k \equiv z_k$;
$R_k = \mu + \eta_k$;
$\Sigma^2_{k+1,i} = \omega_i + \alpha_i \Sigma^2_{k,i} + \beta_i \eta^2_{k,i}$ for $i = 1, \ldots, p$.

2.2 Basics of Neural Network

Since deep neutral network (DNN) is applied to the portfolio problem (2), Tsang and Wong (2020) have to define the notation for neural network (NN). The mathematical expression for a multi-layer multi-output feedforward neural network (FNN) with L hidden layers is:

$$x \in \mathbb{R}^d \to f_{L+1}(x; \theta) \in \mathbb{R}^p,$$

where θ is the parameters of the network.

Let $\mathcal{A}^L = \left\{f_{L+1,\theta} : \mathbb{R}^d \to \mathbb{R}^p : \theta \in \mathbb{R}^o\right\}$ be the class of functions computed by the standard multi-layer multi-output FNN with L hidden layers. $\mathcal{A}^L$ can be represented by successive composition of linear combination and activation function $\sigma_r(\cdot)$ (Tsang and Wong, 2020):

$$\begin{aligned} f_0 &= x \\ f_r &= \sigma_r(B_{r-1} f_{r-1} + C_{r-1}), r = 1, \ldots, L, \\ f_{L+1} &= B_L f_L + C_L \end{aligned}$$

where

B_r is the matrix weight and C_r is the vector bias.
$\theta = (B_r, C_r)_{r=0,\ldots,L}$.
$f_0 \in \mathbb{R}^d$ is called the input layer and $f_{L+1} \in \mathbb{R}^p$ is called the output layer.
$f_r \in \mathbb{R}^{\nu_r}$ is called the hidden layers,
where ν_r is the number of hidden units in layer$r, r = 1, \ldots, L$.

Tsang and Wong (2020) assume that the activation function $\sigma_r(\cdot)$ and the number of hidden units ν_r the same for r = 1, ..., L in order to simplify their notation. Tsang and Wong (2020) rewrite $\mathcal{A}^L$ as $\mathcal{A}_v$ to emphasize its dependence on ν.
To handle the bounded control, Tsang and Wong (2020) apply the sigmoid function to the output of $\mathcal{A}_v$. Define as the following:

$$\mathcal{G}_\nu = \left\{ f : \mathbb{R}^d \to \mathbb{R}^p : f(x) = \sigma\left(f_{L+1}(x)\right), f_{L+1} \in \mathcal{A}_\nu \right\} \tag{7}$$

where $\sigma(x) = \frac{1}{1+e^{-x}}$ is the sigmoid function with range of $[0, 1]$, and is applied to f_{L+1} elementwisely.

3 Deep-Learning Solution

Tsang and Wong (2020) introduce their deep-learning solution for the MDP problem (1) in this section. Suppose $D = [0, 1]^p$, denote $g_{(K-1)} = \{g_0, \ldots, g_{K-1}\}$ as a sequence of feedback control functions. The value function for the MDP problem (1) can be expressed as follows:

$$V_0(s_0) = \min_{\phi \in \mathcal{H}^{\mathcal{U}}} \mathbb{E}[\phi(\zeta)], \tag{8}$$

where $\zeta = (\eta_0, \ldots, \eta_{K-1})$ contains all the random variables throughout the entire process,

$$\mathcal{H}^{\mathcal{U}} = \left\{ \zeta \to \Psi\left(s_K^{g_{(K-1)}}(\zeta)\right) : \{s_k\}_{k=0}^K \text{ subject to } (1), g_0, \ldots, g_{K-1} \in \mathcal{U}(\cdot) \right\} \tag{9}$$

denotes the class of controlled process with admissible control function in $\mathcal{U}(\cdot)$. The notation $s_K^{g_{(K-1)}}$ is used to emphasize that the state process adopts the control $g_{(K-1)}$.

Assume that Tsang and Wong (2020) restrict our minimization over the feedback control policy for the functions $g_k(\cdot; \theta_k) \in \mathcal{G}_\nu$. Tsang and Wong (2020) define the following relaxed problem:

$$\begin{aligned} V_0^*(s_0) &= \min_{g_0, \ldots, g_{K-1} \in \mathcal{G}_\nu} \mathbb{E}\left[\Psi(s_K) \mid \mathcal{F}_0\right] \\ &= \min_{\theta_0, \ldots, \theta_{K-1}} \mathbb{E}\left[\Psi(s_K) \mid \mathcal{F}_0\right] \\ &= \min_{\phi \in \mathcal{H}^{\mathcal{G}_\nu}} \mathbb{E}[\phi(\zeta)], \end{aligned} \tag{10}$$

where $\mathcal{G}_\nu$ is defined in (7). Based on the above relaxed problem, the class of controlled process $\mathcal{H}^{\mathcal{G}_\nu}$ considered in (10) can be described by a deep neural network architecture as follows (Tsang and Wong, 2020):

1. Denote the computational unit $\{s_k\}_{k=0}^K, \{g_k\}_{k=0}^{K-1}, \{\eta_k\}_{k=0}^{K-1}$ and V_K.
   s_0 and $\{\eta_k\}_{k=0}^{K-1}$ are the input units (which means in-degree of node is 0).
   V_K is the only output unit of the network (which also means out-degree of this node is 0).

2. Define the following connections within the graph.

(a) For $k = 0, 1, \ldots, K-1, (s_k, g_k, \eta_k) \to s_{k+1} : s_{k+1} = h(s_k, g_k, \eta_k)$ models the transition to the next timestep.

(b) For $k = 0, 1, \ldots, K-1, s_k \to g_k : g_k = g_k(s_k; \theta_k)$ for some function $g_k(s_k; \theta_k) \in \mathcal{G}_\nu$. θ_k is the parameters of the subnetwork $s_k \to g_k$ at timestep k. $\mathcal{G}_\nu$ may have any number of hidden layers $\mathrm{L} \geq 1$.

(c) $s_K \to V_K : V_K = \Psi(s_K)$ denotes the terminal network loss function.

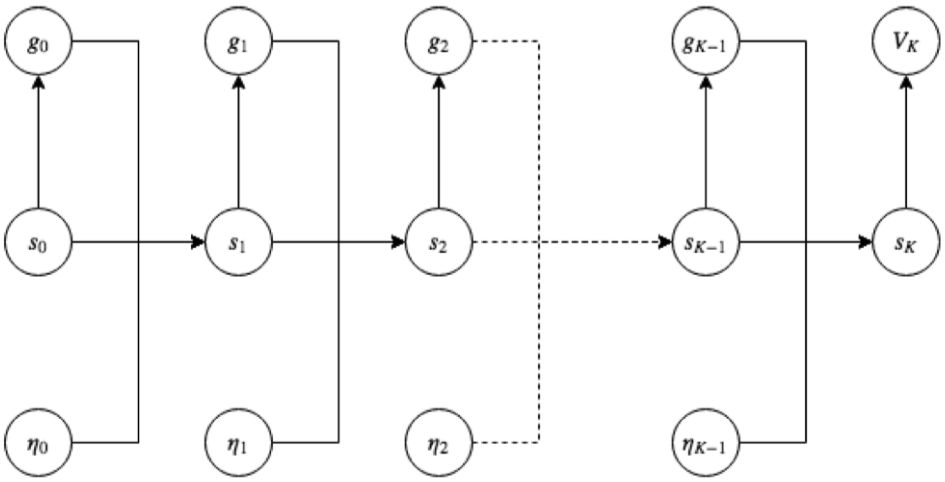

Figure 1: Graphical illustration of the proposed deep neural network (Tsang and Wong, 2020)

With the above architecture, Tsang and Wong (2020) can train the DNN with simulated data $\{\zeta_i\}_{i=1}^n$ by some optimization algorithm. Note that the optimization is with respect to the network parameters $\Theta = (\theta_0, \ldots, \theta_{K-1})$ in the training stage. This corresponds to the choice of the feedback control functions and hence is conceptually equivalent to the learning of the optimal feedback control policy.

Denote $\phi(\zeta; \Theta)$ as the output value V_K of the network with input s_0 and ζ for fixed s_0. The network training is described as, optimizing for $\hat{\Theta}$ which satisfies (Tsang and Wong, 2020),

$$n^{-1} \sum_{i=1}^{n} \phi\left(\zeta_i; \hat{\Theta}\right) = \min_{\Theta} n^{-1} \sum_{i=1}^{n} \phi(\zeta_i; \Theta) \tag{11}$$

Denote $\phi_\Theta(\zeta) = \phi(\zeta; \Theta)$. Observe that $\mathcal{H}^{\mathcal{G}_\nu} = \{\phi_\Theta(\zeta) : \Theta \in \mathbb{R}^w\}$ where w is the total number of parameters. Tsang and Wong (2020) can then write $\mathcal{H}^{\mathcal{G}_\nu}$ as the class of functions computed by the DNN defined above. In addition, $\hat{\phi}(\zeta) = \phi(\zeta; \hat{\Theta})$ satisfies,

$$n^{-1} \sum_{i=1}^{n} \hat{\phi}(\zeta_i) = \min_{\phi \in \mathcal{H}^{\mathcal{G}_\nu}} n^{-1} \sum_{i=1}^{n} \phi(\zeta_i) \tag{12}$$

4 Numerical Examples

In this section, the author mainly analyze and verify the convergence conclusion of AR(1) and CCC-GARCH(1,1) model where their utility function values will converge to a global minimum point.

The method to implement and get the deep-learning solution to this portfolio selection problem is Monte-Carlo simulation so that tons of data can be generated to feed into the deep-learning model, which can effectively handle high-dimensional problem according to the author. However, the procedure of generating random numbers, training models and finally plotting the capital allocation line is very time-consuming and computational expensive.

Firstly, some useful conclusions that are very helpful in programming will be introduced.

4.1 Preliminaries

4.1.1 I.I.D. Return Recurrence Relation

Under the i.i.d. return case, we assume that the asset prices follow the Black-Scholes model

$$\frac{dS_t}{S_t} = (r\mathbf{1} + \Sigma\lambda)\, dt + \Sigma dZ_t$$

where $r \in \mathbb{R}$, $\mathbf{1} = (1, \cdots, 1) \in \mathbb{R}^p$, $\lambda \in \mathbb{R}^p$, $\Sigma \in \mathbb{R}^{p\times p}$ and Z_t are p independent Wiener processes. With a constant rebalancing time period Δ, the risk free return $R_f = \exp(r\Delta) - 1$ and then we can derive the excess return vector R_k^e

$$\begin{aligned}
& S_k = \exp\left(\left(r\mathbf{1} + \Sigma\lambda - \frac{1}{2} diag\left(\Sigma\Sigma^T\right)\right)\Delta + \sqrt{\Delta}\Sigma\epsilon_k\right) S_{k-1} \\
\Rightarrow \quad & R_k^e = \frac{S_k}{S_{k-1}} - \mathbf{1} - R_f\mathbf{1} \\
\Rightarrow \quad & R_k^e = \exp\left(\left(r\mathbf{1} + \Sigma\lambda - \frac{1}{2} diag\left(\Sigma\Sigma^T\right)\right)\Delta + \sqrt{\Delta}\Sigma\epsilon_k\right) - \exp(r\Delta)\,\mathbf{1}
\end{aligned}$$

where $\epsilon_k \sim N_p(\mathbf{0}, I_p)$ are i.i.d. multivariate normal, I_p is the identity matrix. $diag(A) = (A_{11}, \cdots, A_{pp})$ return the vector of the diagonal elements of square matrix A.

4.1.2 Single-period Models

In order to compare the performance of single-period model and multi-period model in the following sections, a way must to be figured out to derive the capital allocation line for the single-period case which is not clearly explained by the author.

Quadratic programming can be used to solve the minimum-variance portfolio (MVP) problem with the box constraints of the portfolio weights given the expected return μ^*, and write it using the notation in this paper

$$\begin{aligned}
& \min_{g_0} g_0^T \Sigma_{R_0^e} g_0 \\
\text{s.t.} \quad & g_0^T R_0^e = \mu^* - R_f \\
& g_0 \in [l, u]^p
\end{aligned}$$

Alternatively, if there are no constraints on the weights of the portfolio, the conclusion in the textbook can be directly used (Lai and Xing, 2008).

$$\min_{g_0} g_0^T \Sigma_{R_0^e} g_0 \quad \text{s.t.} \quad g_0^T \boldsymbol{\mu_0} + \left(1 - g_0^T\mathbf{1}\right) R_f = \mu^* \quad \Rightarrow \quad \min_{g_0} g_0^T \Sigma_{R_0^e} g_0 \quad \text{s.t.} \quad g_0^T R_0^e + R_f = \mu^*$$

$$g_0^* = \frac{(\mu^* - R_f)}{\left(\boldsymbol{\mu_0} - R_f\mathbf{1}\right)^T \Sigma_{R_0^e}^{-1}\left(\boldsymbol{\mu_0} - R_f\mathbf{1}\right)} \Sigma_{R_0^e}^{-1}\left(\boldsymbol{\mu_0} - R_f\mathbf{1}\right) \quad \Rightarrow \quad g_0^* = \frac{(\mu^* - R_f)\,\Sigma_{R_0^e}^{-1} R_0^e}{R_0^{eT} \Sigma_{R_0^e}^{-1}}$$

Using these formulas, terminal wealth for the single-period case without constraints can be derived

$$W_{K+1} = W_0 \prod_{k=0}^{K} \left[(\mu^* - R_f) \frac{R_k^{eT} \Sigma_{R_0^e}^{-1} R_0^e}{R_0^{eT} \Sigma_{R_0^e}^{-1} R_0^e} + R_f + 1 \right]$$

and as well as the formula for single-period case with constraints

$$W_{K+1} = W_0 \prod_{k=0}^{K} \left(g_0^{*T} R_k^e + R_f + 1 \right)$$

In this way, mean and variance of the terminal wealth using simulated samples can be calculated.

4.1.3 Multi-period Models

Multi-period portfolio optimization problems can be embedded into the MDP problem of expected utility maximization (EUM) in discrete-time framework.

Quadratic utility function will be used for analysis in this section like the author. And to simulate risky assets return paths for different multi-period models, which is one of the very important parts, the recurrence relation and dynamics should be paid attention to, while the objective for each problem is exactly the same

$$\begin{aligned} V_0(\cdot) = & \min_{\{g_K\}_{K=0}^{K-1}} E\left[\left(W_K - \frac{\gamma}{2} \right)^2 | \mathcal{F}_0 \right] \\ \text{s.t.} \quad & W_{k+1} = W_k \left(g_k^T R_k^e + R_f + 1 \right) \end{aligned}$$

I.I.D. Return Model

$$\text{s.t.} \quad R_k^e = \exp\left(\left(r\mathbf{1} + \Sigma\lambda - \frac{1}{2} diag\left(\Sigma\Sigma^T \right) \right) \Delta + \sqrt{\Delta} \Sigma \epsilon_k \right) - \exp\left(r\Delta \right) \mathbf{1}, \quad k = 0, 1, \cdots, K-1$$

where the $\mathcal{F}_k$-measurable $g_k \in D \subseteq \mathbb{R}^p$ is the investment policy on p risky assets at time k while D is the constrained investment opportunity set. $R_k^e \in \mathbb{R}^p$ is the excess returns of risky assets between the time points k and $k+1$. R_f is the constant risk-free return for each period. In this case, $\{R_k\}$ is serially-independent, and the state vector only consists of W_k.

AR(1) Model

$$\begin{aligned} \text{s.t.} \quad & R_k = \alpha + A R_{k-1} + \epsilon_k \\ & R_k^e = R_k - R_f, \quad k = 0, 1, \cdots, K-1 \end{aligned}$$

where $\alpha \in \mathbb{R}^p$, $A \in \mathbb{R}^{p \times p}$ and ϵ_k is identically and independently distributed p-dimensional random variables.

CCC-GARCH(1,1) Model

$$\begin{aligned} \text{s.t.} \quad & R_{k,i} = \mu_i + z_{k,i} \\ & z_{k,i} = \sigma_{k,i} \epsilon_{k,i} \\ & \sigma_{k,i}^2 = \omega_i + \alpha_i \sigma_{k-1,i}^2 + \beta_i z_{k-1,i}^2 \\ & R_k^e = R_k - R_f, \quad i = 1, \cdots, p, \quad k = 0, 1, \cdots, K-1 \end{aligned}$$

where $\omega_i,\ \alpha_i,\ \beta_i \geq 0$ for $i = 1, \cdots, p$. $\mu = (\mu_1, \cdots, \mu_p) \in \mathbb{R}^p$, $z_k = (z_{k,1}, \cdots, z_{k,p}) \in \mathbb{R}^p$ and $\sigma_k^2 = \left(\sigma_{k,1}^2, \cdots, \sigma_{k,p}^2 \right)$. $\epsilon_k = (\epsilon_{k,1}, \cdots, \epsilon_{k,p})$ are i.i.d. random variables with constant correlation matrix Γ which is positive definite and satisfies $\Gamma_{ii} = 1$, $\Gamma_{ij} = \Gamma_{ji} = \rho_{ij}$ and $\rho_{ij} \in [-1, 1]$.

4.2 Numerical Results

We will mainly analyze three aspects of the algorithm. The first one is its convergence in I.I.D case, the second one is the model dynamics by using CCC-GARCH(1,1) model, and the last one is the capital allocation line of the portfolio (efficient frontier in original paper) in both AR(1) and CCC-GARCH(1,1) case.

4.2.1 Coding Environment

- Windows 10
- Python 3
- Google Colab
- Keras & Tensorflow (for constructing DNN model in multi-period model)
- cvxopt (for quadratic programming in single-period model with box constraints)

In terms of setting up a neuron network, following what the author did, we use tanh activation function and $L = 2$ for each subnetwork, and the neurons each hidden layer consists of varies in different examples. The model is trained without using GPU speed-up. The training is performed with mini-batch optimization with batch size 64 by ADAM optimizer (Kingma and Ba (2014)) with default setting. The parameters are initialized using Glorot uniform initializer (Glorot and Bengio (2010)) without pre-training.

4.2.2 Data

The original data is from Yahoo Finance. They are from 2017-11-01 to 2022-11-01 (1259 observations) of 21 stocks in S&P 500 list, which will be used in analyzing CCC-GARCH(1,1) model. And data pre-processing procedure is done to convert the stock adjusted close prices to stock returns.

And all of the returns are fitted into univariate GARCH(1,1) models to get the parameters for CCC-GARCH(1,1) model, which will be used in generating random paths of stock returns. Table[1] shows the results.

4.2.3 Convergence Analysis - IID case

Firstly, the performance of the algorithm convergence will be tested when training sample size increases with different number of neurons v. For each p, we train 2 models with different values of v. Each model is trained by 100 steps described as follows. For each step, the model is trained with 10,000 simulated training samples (batch size = 64 and epochs = 1), and then the outsample score (loss) of the model is evaluated using a set of simulated testing samples. The same set of simulated testing samples is applied to the model at all steps. And Table[2] shows the other parameters.

Figure[2] gives the results. On the left, the $p = 5$ case, all of the three curves show decreasing trends as the number of training samples increases and finally converge to a level that is very close to 0. And we can infer from the graph that the benchmark error will continue decreasing when time exceeds 100. Algorithm has a not bad performance in this case. However, on the right, which is the $p = 100$ case, all of the three curves decrease very quickly, but as the number of training samples increases, they no longer show decreasing trends and finally reach stable levels a little bit far away from 0. And what more interesting is, in neuron number $v = 5$ case, it performs even better than that when neuron number $v = 10$, which may show a problem of overfitting when the number of parameters is very large. Algorithm performs not so well as when $p = 5$, but this kind of difference is reasonable because of the difference of the number of dimensions.

4.2.4 Model Dynamics - CCC-GARCH(1,1) case

Next, the model dynamics of CCC-GARCH(1,1) will be analyzed as well as the robustness of algorithm and coding.

A set of excess returns can be generated, and be fitted into all of the single-period model and trained DNN model for multi-period case, which yield the stock positions and portfolio wealth as well as the excess returns at each time, we can display them with heat maps and curves in Figure[3]. Table[1] and Table[3] show the parameters in these 5 models, while graphical illustration of the deep neural network structure of CCC-GARCH(1,1) model is given by Figure[4], which is drawn by Keras package.

In Figure[3], we can see that the algorithms work very well since all of the portfolio weights lie in assigned bounds and the values of portfolio wealth are controlled at levels of 1.25 at the first day in single-period case. Besides, we can see very clear characteristics that the excess returns are serially dependent, which is in accordance with CCC-GARCH model.

4.2.5 Capital Allocation Line - AR(1) & CCC-GARCH(1,1) case

Finally, we compute the capital allocation lines using different parameters from Table[4] and Table[5] in AR(1) and CCC-GARCH(1,1) model as shown in Figure[5] and Figure[6].

In Figure[4], we can see that all of the three curve overlap to each other, which demonstrates that they behave similarly in term of accumulating wealth. This result is totally different from the conclusion that the author draws, which states that multi-period model is much better. On the one hand, maybe the insufficient samples are used to calculate in our model, which is likely to yield a rough estimation. On the other hand, there must be some logic errors in our codes or the author's code, so that very large difference can be explained.

In Figure[5], the results begin to become weirder. The multi-period model performs even worse than the single-period model, while constraint bounds do not matter a lot in terms of the final wealth, which is in accordance with the author's observation. However, due to the limited time, very limited samples and wide time intervals are used to compute the CAL, so that our results are likely to suffer from some errors like truncation error and divergence error.

5 Conclusions

5.1 Observations

From last section, we can conclude that the convergence statements about the algorithms hold true in a general sense, which demonstrates that this algorithm is very efficient in handling high dimensionality case up to 100. However, as the number of dimension becomes large, the number of parameters will grow exponentially, which is likely to yield overfitting issue. Then it is not likely to get accurate utility values in very high dimensionality, which is an obvious drawback of this algorithm.

From our results, the performance of multi-period models are not better than single-period ones, which is totally different from the author's results. On the one hand, there must be some logic problems in either the author's code or ours. On the other hand, if our code is right, we can conclude from our results that the multi-period models are likely to fit the noises in the excess return series and finally yields terrible results because of the problem of overfitting. There may be two things that can be done to improve the models. One is to reduce the number of parameter, i.e., we can reduce the number of hidden layer number or neuron number in each layer. The other one is that we can furtherly process the excess return to show their characteristics more clearly.

5.2 Criticism

Firstly, from our view, the aim to analyze the concept of efficient frontier in the author's paper is not appropriate. Since we only focus on the process of wealth accumulating, it does not have a lot of things to do with the current weights of the each assets, so that it does not make sense to analyze the efficient frontier in the last day. CAL(Capital Allocation Line) in the last day is more proper and can be compared in different setting of single- or multi-period. Therefore, we analyze the CAL of different models instead.

Moreover, there might be something wrong in the implementation of single-period model. Since we can only control the portfolio wealth at the first day, and have no means to control it later on. And when [0,1] constraints are imposed, the maximum value of the return of the portfolio is to invest all of the stocks in weights of ones which have positive returns, which is actually bounded in terms of the maximum wealth at the first day. Thus, there must be some samples that can not satisfy the high expect return of the first day, which will yield a biased result if we simply discard these samples. So, we are highly suspicious for the accuracy of author's single-period results, where the author does not explain what he has done.

Finally, we can see that the curves of the author are extremely smooth, there must be some smoothing methods applied. However, the author does not say anything about them, which make us doubt the authenticity about how he get these curves.

References

Bachouch, A., Huré, C., Langrené, N., and Pham, H. (2021). Deep neural networks algorithms for stochastic control problems on finite horizon: Numerical applications. *Methodology and Computing in Applied Probability*, 24(1):143–178.

Ban, G.-Y., El Karoui, N., and Lim, A. E. (2018). Machine learning and portfolio optimization. *Management Science*, 64(3):1136–1154.

Lai, T. L. and Xing, H. (2008). *Statistical models and methods for financial markets*, volume 831. Springer.

Li, D. and Ng, W.-L. (2000). Optimal dynamic portfolio selection: Multiperiod mean-variance formulation. *Mathematical finance*, 10(3):387–406.

Sirignano, J. A. (2019). Deep learning for limit order books. *Quantitative Finance*, 19(4):549–570.

Tsang, K. H. and Wong, H. Y. (2020). Deep-learning solution to portfolio selection with serially dependent returns. *SIAM Journal on Financial Mathematics*, 11(2):593–619.

Zanger, D. Z. (2018). Convergence of a least-squares monte carlo algorithm for american option pricing with dependent sample data. *Mathematical Finance*, 28(1):447–479.

Appendix A Tables

Stock - i	μ_i	α_i	β_i	ω_i	$\sigma_{0,i}$
AAL	-0.00004	0.00003	0.10000	0.88000	0.02530
AAPL	0.00220	0.00001	0.10000	0.88000	0.02885
ABBV	0.00069	0.00003	0.10000	0.80000	0.01823
ABC	0.00093	0.00004	0.10000	0.80000	0.01709
ABT	0.00077	0.00003	0.10000	0.80000	0.01795
ADBE	0.00162	0.00001	0.10027	0.87973	0.02042
ADI	0.00070	0.00001	0.09003	0.87894	0.02120
ADM	0.00140	0.00003	0.20000	0.70000	0.01532
AEE	0.00065	0.00001	0.10000	0.88000	0.01691
AEP	0.00068	0.00000	0.10000	0.88000	0.01747
AES	0.00141	0.00001	0.10000	0.88000	0.02149
AFL	0.00012	0.00001	0.20000	0.78000	0.01986
AIG	0.00037	0.00001	0.20000	0.78000	0.01851
AIV	0.00077	0.00001	0.09900	0.87120	0.02282
AJG	0.00108	0.00001	0.10000	0.88000	0.02286
AMD	0.00253	0.00013	0.10000	0.80000	0.03390
AME	0.00092	0.00001	0.10000	0.88000	0.01950
AMG	-0.00010	0.00001	0.05000	0.93000	0.02292
AMGN	0.00079	0.00003	0.10000	0.80000	0.01651
AMP	0.00119	0.00001	0.10000	0.88000	0.02808
AMT	0.00077	0.00001	0.10000	0.88000	0.02269

Table 1: Estimated parameters in CCC-GARCH(1,1) model.

Portfolio Number p	$p = 5$	$p = 100$
Continuous Risk-free Rate r	0.03	0.03
Rebalancing Time Δ	1/40	1/30
Multi-period Number K	40	30
Initial Wealth W_0	1	1
Control Level γ	4	6
Weights Domain D	$[0, 1.5]^p$	$[0, 0.5]^p$
Benchmark - HJB Solution V_0	0.821	3.460

Table 2: Parameters Setting for I.I.D. Return Case.

*Under $p = 5$, $\lambda_i = 0.1$ for $i = 1, 2$, $\lambda_i = 0.2$ for $i = 3, 4, 5$. $\Sigma_{ii} = 0.15$, $\Sigma_{ij} = 0.01$ for $i \neq j$.
Under $p = 100$, $\lambda_i = 0.01$ for $i = 1, \cdots, 50$, $\lambda_i = 0.05$ for $i = 51, \cdots, 100$. $\Sigma_{ii} = 0.15$, $\Sigma_{ij} = 0.0025$ for $i \neq j$.

Model Index	1	2	3	4	5
Portfolio Number p			21		
Multi-period Number K			10		
Rebalancing Time Δ			1/10		
Initial Wealth W_0			1		
Continuous Risk-free Rate r			$\exp\left(\frac{0.03}{360}\right)$		
Model Type	Single-period		Multi-period		
Control Level	$\mu^* = 0.25$		$\gamma = 3$		
Weights Domain D	$[-\infty, \infty]^p$	$[0,1]^p$	$[0,1]^p$	$[-0.2,1]^p$	$[-1,1]^p$
Trained Samples	-		100,000		
Neuron Number v	-		50		

Table 3: Parameters Setting for Robustness checks of algorithm in CCC-GARCH(1,1) Case.

*Multi-period models may not give very accurate results since small numbers of trained samples are used.

Model	1	2	3
Portfolio Number p		30	
Multi-period Number K		10	
Rebalancing Time Δ		1/10	
Initial Wealth W_0		1	
Continuous Risk-free Rate r		0.03	
Model Type	Multi-period	Single-period	Single-period
Control Level	$\gamma \in [3, 15]$	$\mu^* \in [0.03, 0.5]$	$\mu^* \in [0.03, 1]$
Weights Domain D	$[0,1]^p$	$[0,1]^p$	$[-\infty, \infty]^p$
Trained Samples	1,000,000	100	100
Neuron Number v	50	-	-

Table 4: Parameters Setting for computing CAL in AR(1) Case.

*$\alpha = 0.015$ for $i = 1, \cdots, p$. $A_{ii} = -0.15$, $A_{ij} = 0$ for $i \neq j$. $\Sigma_{ii} = 0.0238$, $\Sigma_{ij} = 0.0027$ for $i \neq j$
**Both single-period and multi-period models may not give very accurate results since small numbers of trained samples are used.

Weights Domain D	$[-\infty, \infty]^p$	$[0,1]^p$	$[-0.2,1]^p$	$[-1,1]^p$
Portfolio Number p		21		
Multi-period Number K		10		
Rebalancing Time Δ		1/10		
Initial Wealth W_0		1		
Continuous Risk-free Rate r		$\exp\left(\frac{0.03}{360}\right)$		
Model Type	Single-period	Multi-period		
Control Level	$\mu^* \in [0, 0.5]$	$\gamma \in [2.5, 8]$		
Trained Samples	1,000	100,000		
Neuron Number v	-	50		

Table 5: Parameters Setting for computing CAL in CCC-GARCH(1,1) Case.

*Both single-period and multi-period models may not give very accurate results since small numbers of trained samples are used.

Appendix B Graphs

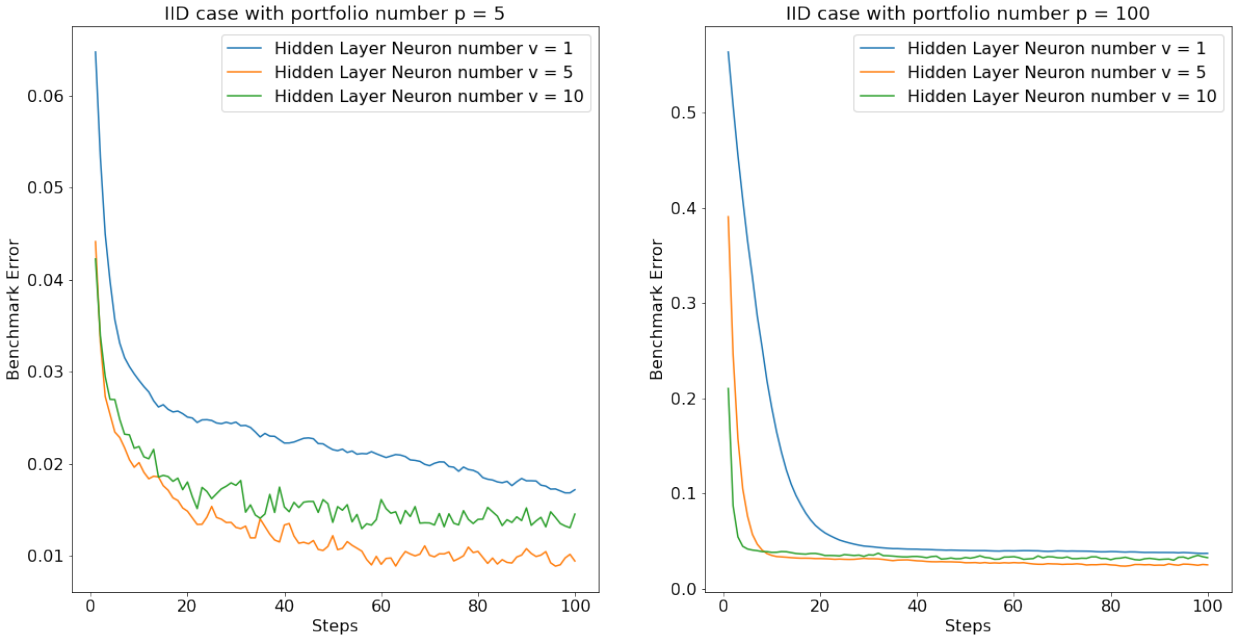

Figure 2: Convergence Analysis - IID case.

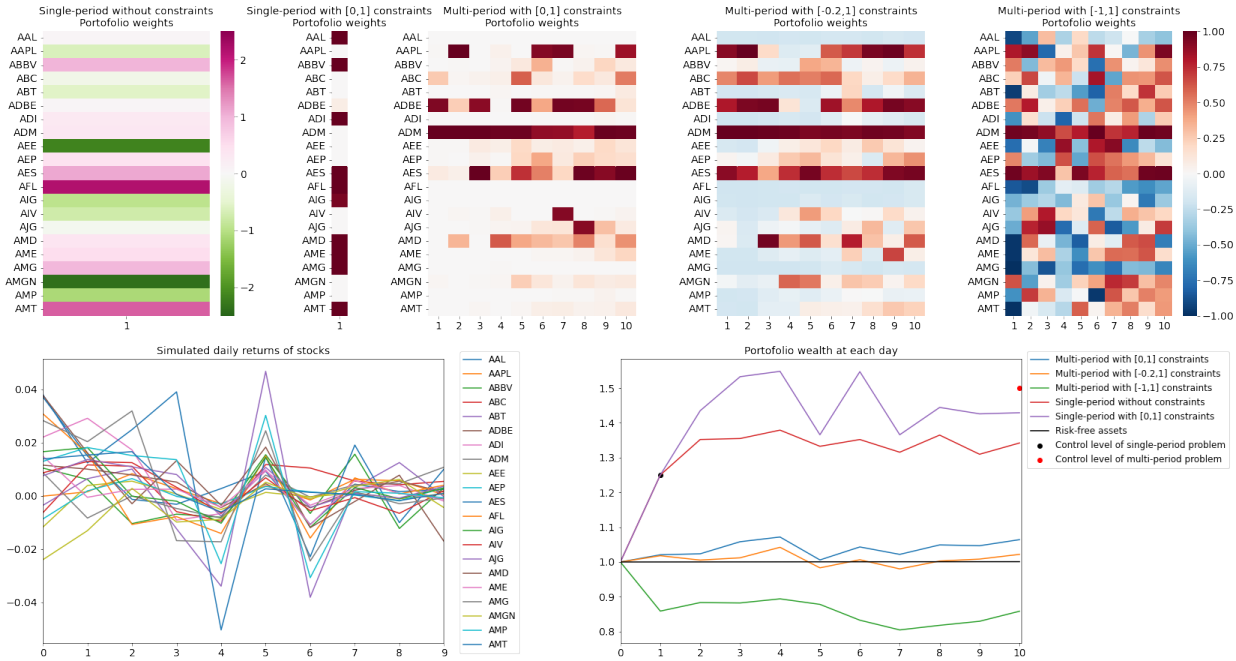

Figure 3: Robustness checks of algorithm - CCC-GARCH(1,1) case.

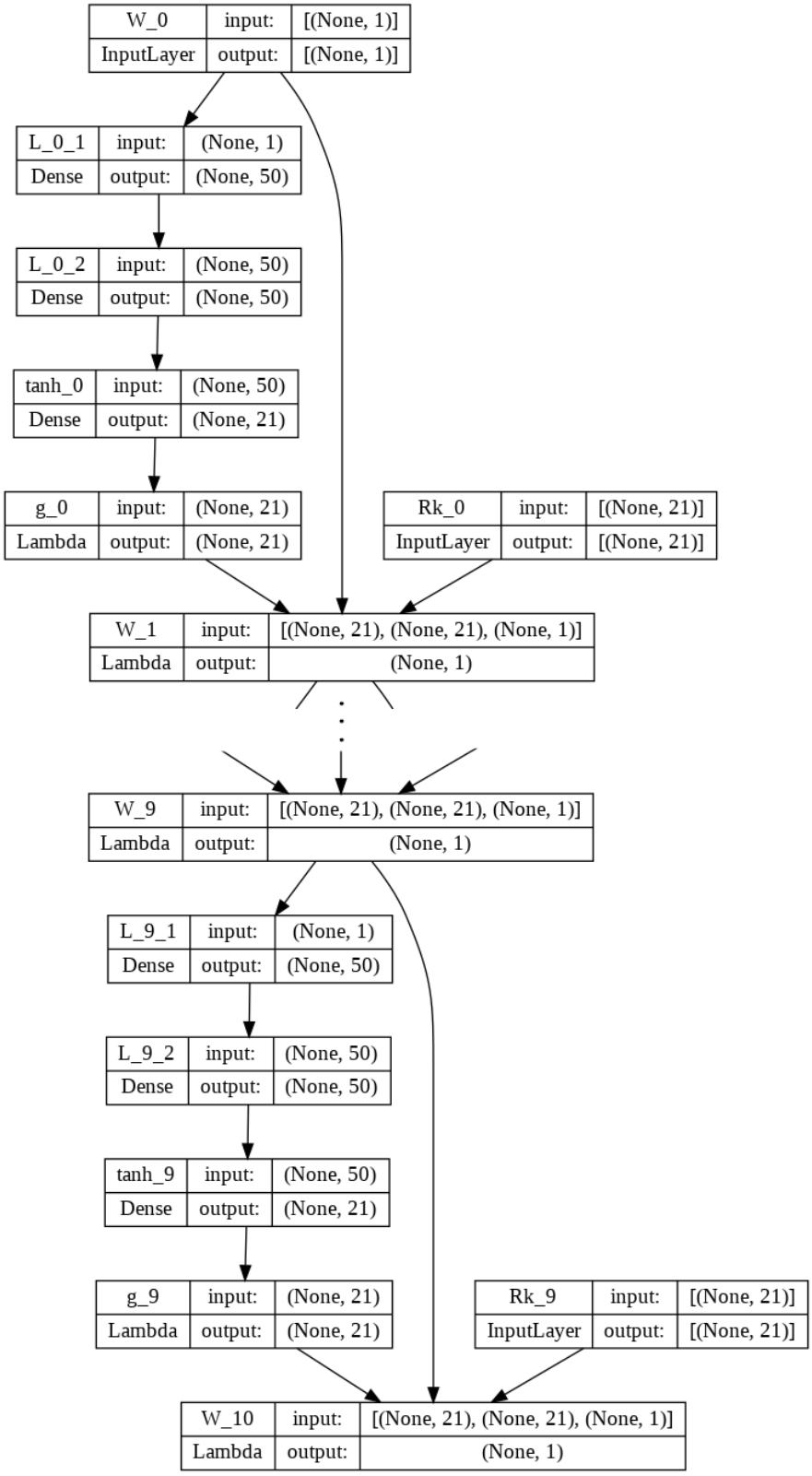

Figure 4: Graphical illustration of the deep neural network structure of CCC-GARCH(1,1) model.

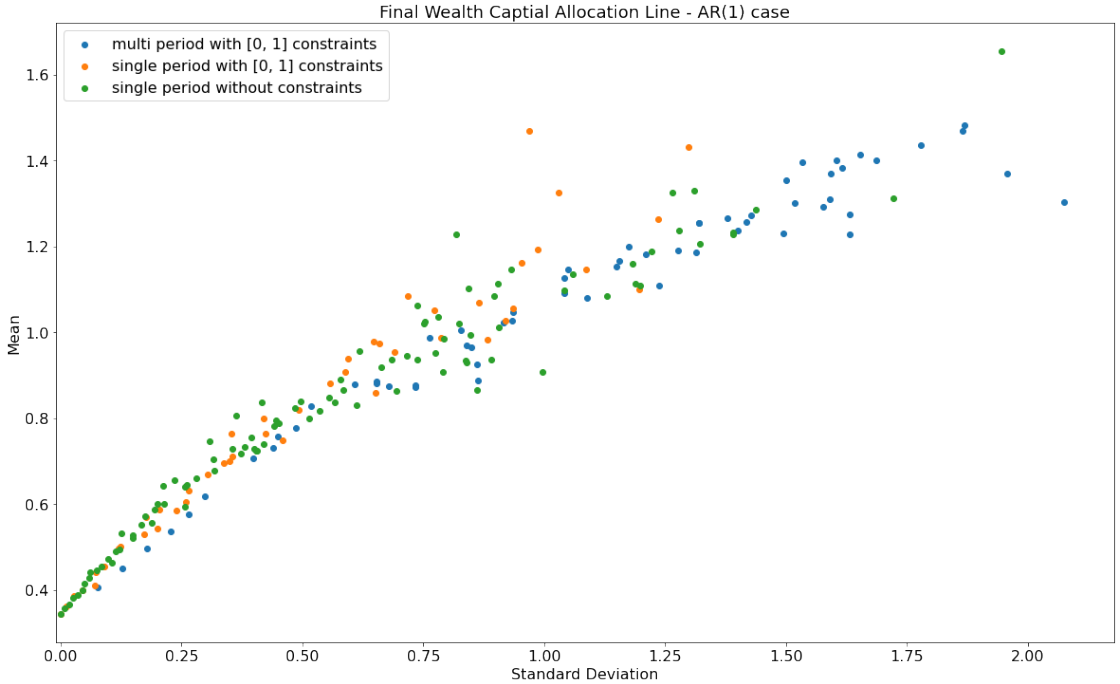

Figure 5: Final Wealth Capital Allocation Line - AR(1) case.

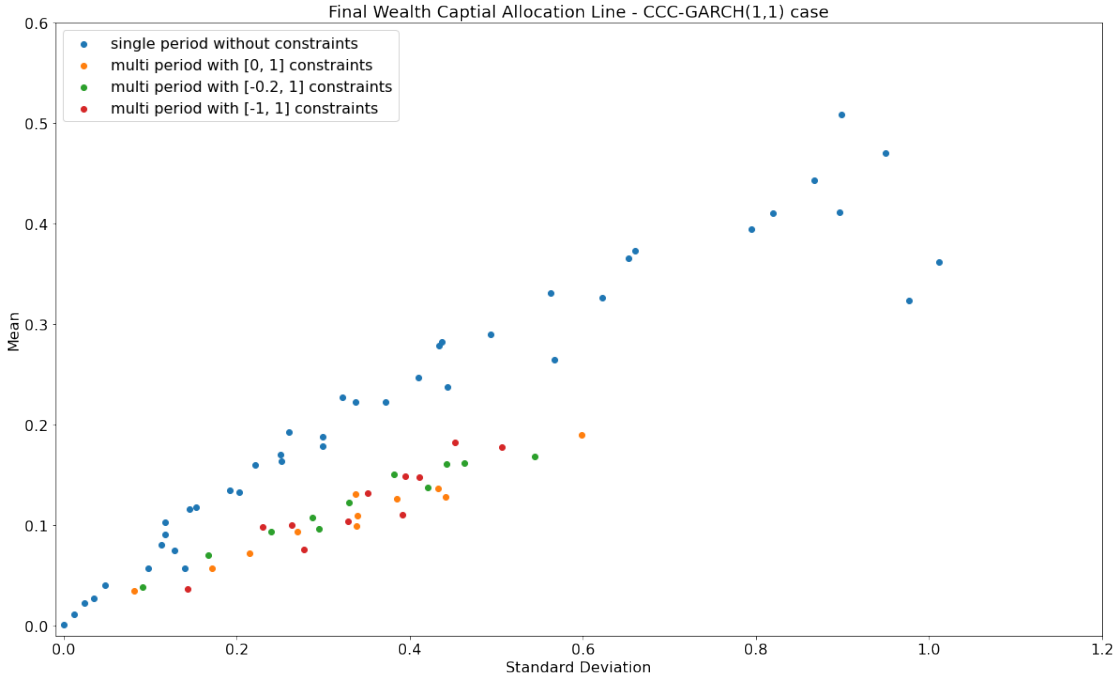

Figure 6: Final Wealth Capital Allocation Line - CCC-GARCH(1,1) case.