2.1.1 Name space

In a market consisting of N stocks, we denote the dividend-adjusted return on stock i at trading day t by $r_{i,t}$. We adopt a factor model for stock return,

$$r_t - r_f = \beta_t F_t + \epsilon_t, \quad t = 1, 2, \ldots, T$$

Here, $r_t = \{r_{i,t}\}_{i=1}^N \in \mathbb{R}^N$ are the dividend-adjusted daily return, $r_f \in \mathbb{R}$ is the risk-free rate, $F_t \in \mathbb{R}^{K\times 1}$ are the underlying factors, $\beta_t \in \mathbb{R}^{N\times K}$ are the corresponding loadings on K factors, and $\epsilon_t \in \mathbb{R}^N$ are the residual returns. Factor candidates varies widely, ranging from economical-driven factors such as the Fama-French factors, to statistically-driven factors derived from PCA. In our approach, factors are selected as the leading eigenvectors in PCA. The number of factors K is chosen based on the eigenvalue spectrum of the empirical correlation of daily returns (Fig. 6(c1-c6)).

Without loss of generality, these factors can be interpreted as portfolios of stocks,

$$F_t = \omega_t \left(r_t - r_f\right)$$

, where $\omega_t \in \mathbb{R}^{K\times N}$ contains corresponding portfolio weights. Eq. 2.1.1 and Eq. 2.1.2 give

$$r_t - r_f = \beta_t \omega_t \left(r_t - r_f\right) + \epsilon_t \Rightarrow \epsilon_t = \left(I - \beta_t \omega_t\right)\left(r_t - r_f\right) := \Phi_t \left(r_t - r_f\right)$$

Here,

$$\Phi_t := \left(I - \beta_t \omega_t\right)$$

defines a linear transformation from r_t to ϵ_t. More importantly, $\epsilon_{i,t}$ can be viewed as the return of a tradable portfolio with weights specified by the i-th row of Φ_t. Consequently, the investing universe spanned by r_t is termed as name equity space, and that spanned by ϵ_t as name residual space.

We denote the portfolio weights in name equity space as $w_t^{R\text{ name}}$ and portfolio weights in name residual space as $w_t^{\epsilon,\text{name}}$. These weights are related by

$$w_t^{R,\text{ name}} = \Phi_t^T w_t^{\epsilon,\text{ name}}$$

directly following the equality in portfolio return

$$\left(w_t^{\epsilon\text{ name}}\right)^T \epsilon_t = \left(w_t^{\epsilon,\text{ name}}\right)^T \Phi_t \left(r_t - r_f\right) = \left(w_t^{R,\text{ name}}\right)^T \left(r_t - r_f\right)$$

For factors derived by PCA, we have

$$\Phi_t \beta_t = 0 \Longrightarrow \left(w_t^{R,\text{ name}}\right)^T \beta_t = \left(w_t^{\epsilon\text{ name}}\right)^T \Phi_t \beta_t = 0, \quad \forall w_t^{\epsilon\text{ name}}$$

with proof given in the appendix. It means that for any $w_t^{\epsilon,\text{name}}$, the $w_t^{R,\text{ name}}$ calculated by Eq. 2.1.5 satisfy,

$$\left(w_t^{R,\text{ name}}\right)^T (r_t - r_f) = \left(w_t^{\epsilon,\text{name}}\right)^T \Phi_t \left(\beta_t F_t + \epsilon_t\right) = \left(w_t^{\epsilon,\text{name}}\right)^T \Phi_t \epsilon_t = \left(w_t^{R,\text{name}}\right)^T \epsilon_t$$

2.1.2 Rank space

We initiate our formulation for market decomposition in rank space by introducing key notations. Let $c_{i,t}$ denote the capitalization of stock i at day t, and $c_{(k),t}$ represent the capitalization of stock which occupies k-th rank in descending order at day t. Additionally, $\mathcal{R}_{i,t}$ represents the rank in capitalization for stock i at day t, and $\mathcal{I}_{(k),t}$ represents the stock index (name) that occupies the rank k at day t. We define the daily return on rank k at day t in the continuous-time limit as $\tilde{r}_{(k),t}$, that

$$\tilde{r}_{(k),t} := \frac{c_{(k),t} - c_{(k),t-1}}{c_{(k),t-1}} = \frac{c_{\mathcal{I}_{(k),t},t} - c_{\mathcal{I}_{(k),t-1},t-1}}{c_{\mathcal{I}_{(k),t-1},t-1}}$$

Notably, $\tilde{r}_t$ does not necessarily correspond to direct financial quantity because stock names occupying rank k may be different between day t and $t-1$ (i.e. $\mathcal{I}_{(k),t} \neq \mathcal{I}_{(k),t-1}$). The realization of $\tilde{r}_t$ poses a critical challenge, which will be further elaborated in the section 2.3. Nevertheless, we assume a factor model for $\tilde{r}_t$ parallel to name space:

$$\tilde{r}_t - r_f = \tilde{\beta}_t \tilde{F}_t + \tilde{\epsilon}_t$$

, where $\tilde{r}_t = \left\{r_{(k),t}\right\}_{k=1}^N \in \mathbb{R}^N, \tilde{\beta}_t \in \mathbb{R}^{N\times K}, \tilde{F}_t \in \mathbb{R}^{K\times 1}$, and $\tilde{\epsilon}_t \in \mathbb{R}^N$. Following a similar transformation as in name space, we have:

$$\tilde{r}_t - r_f = \tilde{\beta}_t \tilde{\omega}_t \left(\tilde{r}_t - r_f\right) + \tilde{\epsilon}_t \Rightarrow \tilde{\epsilon}_t = \left(I - \tilde{\beta}_t \tilde{\omega}_t\right)\left(\tilde{r}_t - r_f\right) := \tilde{\Phi}_t \left(\tilde{r}_t - r_f\right)$$

, where

$$\tilde{\Phi}_t := \left(I - \tilde{\beta}_t \tilde{\omega}_t\right)$$

defines a linear transformation from $\tilde{r}_t$ to $\tilde{\epsilon}_t$. Remarkably, if $\tilde{r}_t$ becomes realizable, the $\tilde{\epsilon}_{i,t}$ will also become the return of a tradable portfolio with weights on the artificial financial instruments realizing $\tilde{r}_t$ given by the i-th row of $\tilde{\Phi}_t$. Consequently, we define the investing universe spanned by

$\tilde{r}_t$ as rank equity space and that spanned by $\tilde{\epsilon}_t$ as rank residual space, echoing our definition in the name space. The portfolio weights in rank equity space are denoted as $w_t^{R,\text{ rank}}$ and that in rank residual space as $w_t^{\epsilon,\text{ rank}}$. From similar reasoning in name space, these weights are related by

$$w_t^{R,\text{rank}} = \tilde{\Phi}_t^T w_t^{\epsilon,\text{rank}}$$

, and therefore,

$$\left(w_t^{R,\text{rank}}\right)^T \beta_t = \left(w_t^{\epsilon,\text{rank}}\right)^T \tilde{\Phi}_t \beta_t = 0, \quad \forall w_t^{\epsilon,\text{rank}}$$

. This suggests that the constructed portfolios in rank space satisfy market neutrality and consequently approximate dollar neutrality, similar to those in name space.

2.2 Trading signals and portfolio weights

Despite that the residual returns ϵ_t or $\tilde{\epsilon}_t$ may be calculated in either name space or rank space, deriving the corresponding trading signal and portfolio weights takes a unified framework, which we elaborate on in this section. We first define the cumulative residual returns x_t^L as

$$x_t^L = (x_{t-L+1}, x_{t-L+2}, \ldots, x_t)$$

, where

$$x_{t-L+\alpha} = \sum_{j=1}^{\alpha} \epsilon_{t-L+j}, \quad \alpha = 1, 2, \ldots, L$$

for name space and

$$\tilde{x}_{t-L+\alpha} = \sum_{j=1}^{\alpha} \tilde{\epsilon}_{t-L+j}, \quad \alpha = 1, 2, \ldots, L$$

for rank space. The cumulative residual returns x_t^L and $\tilde{x}_t^L$ are used to calculate w_t^ϵ in name space and rank space, respectively. In the following, we focus on two approaches to calculate w_t^ϵ : (i) a parametric model based on OU process [3, 24], and (ii) deep neural networks [12] that combine the convolutional neural networks (CNN) [20] with transformers [7, 23].

2.2.2 Deep neural networks We adopt deep neural networks as a data-driven method to calculate portfolio weights in residual space w_t^ϵ for both name space and rank space. Specifically, the input of the neural networks is the cumulative residual returns x_t^L and the output of the neural networks is $w_t^{\epsilon|\text{NN, name/rank}}$,

$$\mathcal{N} : x_t^L \to w_t^{\epsilon|\text{NN, name}}$$

for name space and

$$\mathcal{N} : \tilde{x}_t^L \to w_t^{\epsilon|\text{NN,rank}}$$

for rank space. The neural networks are trained similarly in both name and rank space through mean-variance optimization,

$$\begin{aligned}
&\text{Maximize}_{\mathcal{N}(\cdot)} \mathbb{E}\left[\left(w_t^{R|\text{NN,name}}\right)^T (r_{t+1} - r_f)\right] - \gamma \,\text{Var}\left[\left(w_t^{R|\text{NN, name}}\right)^T (r_{t+1} - r_f)\right] \\
&\text{s.t. } w_t^{R|\text{NN,name}} = \frac{\Phi_t^T w_t^{\epsilon|\text{NN, name}}}{\left\| \Phi_t^T w_t^{\epsilon|\text{NN, name}} \right\|_1} \\
&w_t^{\epsilon|\text{NN,name}} = \mathcal{N}\left(x_t^L\right)
\end{aligned}$$

in name space and

$$\begin{aligned}
&\text{Maximize}_{\mathcal{N}(\cdot)} \mathbb{E}\left[\left(w_t^{R|\text{NN,rank}}\right)^T (\tilde{r}_{t+1} - r_f)\right] - \gamma \,\text{Var}\left[\left(w_t^{R|\text{NN,rank}}\right)^T (\tilde{r}_{t+1} - r_f)\right] \\
&\text{s.t. } w_t^{R|\text{NN,rank}} = \frac{\tilde{\Phi}_t^T w_t^{\epsilon|\text{NN,rank}}}{\left\| \tilde{\Phi}_t^T w_t^{\epsilon|\text{NN,rank}} \right\|_1} \\
&w_t^{\epsilon|\text{NN,rank}} = \mathcal{N}\left(\tilde{x}_t^L\right)
\end{aligned}$$

in rank space. γ is the risk-aversion factor. The empirical expectation and variance are obtained over a consecutive time window of length T,

$$\begin{aligned}
\mathbb{E}\left[\left(w_t^{R|\text{NN}}\right)^T (r_{t+1} - r_f)\right] &\approx \frac{1}{T} \sum_{\alpha=1}^{T} \left(w_{t+\alpha}^{R|\text{NN}}\right)^T (r_{t+\alpha+1} - r_f) \\
\text{Var}\left[\left(w_t^{R|\text{NN}}\right)^T (r_{t+1} - r_f)\right] &\approx \frac{1}{T} \sum_{\alpha=1}^{T} \left[\left(w_{t+\alpha}^{R|\text{NN}}\right)^T (r_{t+\alpha+1} - r_f) - \mathbb{E}\left(\left(w_t^{R|\text{NN}}\right)^T (r_{t+1} - r_f)\right)\right]^2
\end{aligned}$$

. The problems here share a similar spirit to the Markowitz portfolio optimization [1]. In our empirical analysis, we choose the risk-aversion factor $\gamma = 2$ and length of time window $T = 24$ days.

With $w_t^{\epsilon|\text{NN, name/rank}}$ as the output from the neural network, the portfolio weights in equity space $w_t^{R|\text{NN,name/rank}}$ is seemingly integrated using Eq. 2.1.5 in name space

$$w_t^{R|\text{NN,name}} = \Phi_t^T w_t^{\epsilon|\text{NN,name}}$$

and using Eq. 2.1.13 in rank space,

$$w_t^{R|\text{NN,rank}} = \tilde{\Phi}_t^T w_t^{\epsilon|\text{NN,rank}}$$

In the following, we delve into the specific architecture of our neural networks, illustrated in Fig. 2. Our CNN-transformer architecture harnesses the strengths of CNN in extracting local patterns and transformers in capturing long-term dependencies. The inputs of our neural networks are the trajectories of cumulative residual returns, $x_t^L \in \mathbb{R}^{N\times L}$, processed through two layer of multi-channel convolutional networks, followed by a standard transformer encoder layer that models global relationships via multi-head attention. Specifically, in the convolutionary layer,

$$\begin{aligned}
x_t^{(1)} &= \frac{x_t^L - \mathbb{E}\left(x_t^L\right)}{\sqrt{\text{Var}\left(x_t^L\right)+\epsilon}} \times \gamma^{(1)} + \beta^{(1)}, \quad y_t^{(1)} = W^{(1)} * x_t^{(1)} + b^{(1)}, \quad z_t^{(1)} = \text{ReLu}\left(y_t^{(1)}\right) + x_t^{(1)}; \\
x_t^{(2)} &= \frac{z_t^{(1)} - \mathbb{E}\left(z_t^{(1)}\right)}{\sqrt{\text{Var}\left(z_t^{(1)}\right)+\epsilon}} \times \gamma^{(2)} + \beta^{(2)}, \quad y_t^{(2)} = W^{(2)} * x_t^{(2)} + b^{(2)}, \quad z_t^{(2)} = \text{ReLu}\left(y_t^{(2)}\right) + x_t^{(2)}
\end{aligned}$$

. The superscript (1) or (2) specifies the layer number. $x_t^{(1)} \in \mathbb{R}^{N\times L}$ is the input of the first convolutional layer. $W^{(1)} \in \mathbb{R}^{D_{\text{channel}}\times 1\times D_{\text{kernel}}}$ and $W^{(2)} \in \mathbb{R}^{D_{\text{channel}}\times D_{\text{channel}}\times D_{\text{kernel}}}$ are the convolutionary kernels for the convolution operator denoted by $*, b^{(1,2)} \in \mathbb{R}^{D_{\text{channel}}}$ is the bias, and $y_t^{(1,2)} \in \mathbb{R}^{N\times D_{\text{channel}}\times L}$ is the output of convolutionary operator. D_{channel} is the number of channels and D_{kernel} is the size of the convolution kernel. We adopt a rectified linear unit (denoted as $\text{ReLu}(\cdot)$) as our activation function. We also apply (i) instance normalization [22] with learnable parameter $\gamma^{(1,2)}$ and $\beta^{(1,2)}$ at the input of each convolution layer to accelerate the training process, and (ii) residual connection [13] to avoid vanishing gradients by directly connecting the input $x_t^{(1,2)}$ to the output $z_t^{(1,2)} \in \mathbb{R}^{N\times D_{\text{channel}}\times T}$. We choose the hyper-parameters for our neural networks as number of channels $D_{\text{channel}} = 8$ and size of the convolution kernel $D_{\text{kernel}} = 2$.

The outputs of convolutionary layers, $z_t^{(2)} \in \mathbb{R}^{N\times D_{\text{channel}}\times T}$ are subsequently fed into a standard transformer encoder layer [23]. The transformer encoder layer utilizes the multi-head attention modeled by the inner product between the famous key-query-value matrices. To elaborate,

$$
\begin{aligned}
x_t^{\text{transformer}} &= \left(z_t^{(2)}\right)^T \\
&\begin{cases} Q_i = \text{DropOut}\left(W_i^Q x_t^{\text{transformer}} + b_i^Q\right) \\ K_i = \text{DropOut}\left(W_i^K x_t^{\text{transformer}} + b_i^K\right), \quad i = 1, 2, \ldots, H \\ V_i = \text{DropOut}\left(W_i^V x_t^{\text{transformer}} + b_i^V\right) \end{cases} \\
&\text{head}_i = \text{softmax}\left(\frac{Q_i K_i^T}{\sqrt{d_{\text{channel}}/H}}\right), \quad i = 1, 2, \ldots, H \\
&y_t = \text{Concat}\left(\text{head}_1 V_1, \ldots, \text{ head }_H V_H\right) \\
&z_t = \text{LayerNorm}\left(x_t^{\text{transformer}} + y_t\right) \\
&o_t = \text{LayerNorm}\left(W^O z_t + b^O + y_t\right)
\end{aligned}
$$

, where $x_t^{\text{transformer}} \in \mathbb{R}^{N \times T \times D_{\text{channel}}}$ is the input of the transformer encoder layer. In addition, $W_i^Q \in \mathbb{R}^{(D_{\text{channel}}/H) \times D_{\text{channel}}}, W_i^K \in \mathbb{R}^{(D_{\text{channel}}/H) \times D_{\text{channel}}}$, and $W_i^V \in \mathbb{R}^{D_{(\text{channel}}/H) \times D_{\text{channel}}}$ are the linear weights. $b_i^Q \in \mathbb{R}^{D_{\text{channel}}/H}, b_i^K \in \mathbb{R}^{D_{\text{channel}}/H}$, and $b_i^V \in \mathbb{R}^{D_{\text{channel}}/H}$ are the bias. Softmax $(\cdot)$ stands for softmax function and Concat $(\cdot)$ for matrix concatenation, and $y_t \in \mathbb{R}^{N \times T \times D_{\text{channel}}}$ is the output of multi-attention layer. $W^O \in \mathbb{R}^{D_{\text{channel}} \times D_{\text{channel}}}$ and $b^O \in \mathbb{R}^{D_{\text{channel}}}$ are the linear weights and bias in the output linear layer. $o_t \in \mathbb{R}^{N \times T \times D_{\text{channel}}}$ is the output of the transformer. In addition to the residual connection similar to the convolutional layer, we also introduce the drop-out technique, denoted as Dropout $(\cdot)$, to regularize overfitting with drop-out probability p, and layer normalization [4], denoted as LayerNorm $(\cdot)$, to improve training stability. We choose the hyper-parameters for our neural networks as the number of heads $H = 4$ and the drop-out probability $p = 0.25$.

Finally, we choose the last slice along the time axis in the output of the transformer, $o_t \in \mathbb{R}^{N \times T \times D_{\text{channel}}}$, as the hidden state summarizing the information up to time t. The portfolio weights in residual space $w_t^{\epsilon\text{NN, name/rank}}$ are calculated by a linear relationship,

$$
w_t^{\epsilon\text{NN, name /rank}} = W^F \left(o_t[:, -1, :]\right) + b^F
$$

, where $o_t[:, -1, :] \in \mathbb{R}^{N \times D_{\text{channel}}}$ means the last slice along the second-dimension (time-axis) of o_t, and $W^F \in \mathbb{R}^{1 \times D_{\text{channel}}}, b^F \in \mathbb{R}$ are the parameters.

0.1. Understanding Convolutional Operations in Financial Time Series

Concept and Mechanics

In traditional neural networks, each input is processed independently. However, financial time series exhibit temporal patterns that are crucial for prediction. Convolutional operations help detect these patterns by analyzing sequences of data points collectively through a sliding window mechanism.

The convolutional layer in our architecture processes the normalized returns matrix $x_t^{(1)} \in \mathbb{R}^{N \times L}$, where N represents the number of assets and L the length of the time window. The convolution operation uses a kernel $W^{(1)} \in \mathbb{R}^{D_{\text{channel}} \times 1 \times D_{\text{kernel}}}$, where D_{channel} represents the number of pattern detectors and D_{kernel} the size of the temporal window.

Detailed Example

Let's consider a concrete example with three assets ($N = 3$), a five-day time window ($L = 5$), three pattern detectors ($D_{\text{channel}} = 3$), and a two-day window ($D_{\text{kernel}} = 2$).

Suppose our normalized return matrix (after instance normalization) is:

$$x_t^{(1)} = \begin{bmatrix} 0.5 & -0.3 & 0.2 & 0.4 & -0.1 \\ 0.3 & 0.4 & -0.2 & -0.3 & 0.1 \\ -0.4 & 0.2 & 0.3 & -0.1 & 0.2 \end{bmatrix}$$

Our convolution kernel consists of three channels, each designed to detect different patterns:

$$W^{(1)} = \begin{bmatrix} [0.8 & -0.8] & \text{(reversal detector)} \\ [0.7 & 0.7] & \text{(trend detector)} \\ [1.0 & 0.0] & \text{(level detector)} \end{bmatrix}$$

with bias terms $b^{(1)} = [0.1, 0.2, 0.0]$.

Let's examine how each channel processes the first asset's returns:

Channel 1 (Reversal Detection): Looking for price reversals with opposite-signed weights:

$$\begin{aligned} &\text{Days 1-2:} && (0.5 \times 0.8) + (-0.3 \times -0.8) + 0.1 = 0.64 \\ &\text{Days 2-3:} && (-0.3 \times 0.8) + (0.2 \times -0.8) + 0.1 = -0.34 \\ &\text{Days 3-4:} && (0.2 \times 0.8) + (0.4 \times -0.8) + 0.1 = -0.14 \\ &\text{Days 4-5:} && (0.4 \times 0.8) + (-0.1 \times -0.8) + 0.1 = 0.42 \end{aligned}$$

Channel 2 (Trend Detection): Looking for continuing trends with same-signed weights:

$$\begin{aligned}
\text{Days 1-2:} \quad & (0.5 \times 0.7) + (-0.3 \times 0.7) + 0.2 = 0.34 \\
\text{Days 2-3:} \quad & (-0.3 \times 0.7) + (0.2 \times 0.7) + 0.2 = 0.13 \\
\text{Days 3-4:} \quad & (0.2 \times 0.7) + (0.4 \times 0.7) + 0.2 = 0.62 \\
\text{Days 4-5:} \quad & (0.4 \times 0.7) + (-0.1 \times 0.7) + 0.2 = 0.41
\end{aligned}$$

Channel 3 (Level Detection): Looking at absolute price levels:

$$\begin{aligned}
\text{Days 1-2:} \quad & (0.5 \times 1.0) + (-0.3 \times 0.0) + 0.0 = 0.50 \\
\text{Days 2-3:} \quad & (-0.3 \times 1.0) + (0.2 \times 0.0) + 0.0 = -0.30 \\
\text{Days 3-4:} \quad & (0.2 \times 1.0) + (0.4 \times 0.0) + 0.0 = 0.20 \\
\text{Days 4-5:} \quad & (0.4 \times 1.0) + (-0.1 \times 0.0) + 0.0 = 0.40
\end{aligned}$$

After applying the convolution to all assets, our output $y_t^{(1)}$ becomes a tensor of shape $(3 \times 3 \times 4)$, representing: - 3 assets - 3 channels (pattern detectors) - 4 time points (reduced from 5 due to the kernel window of size 2)

This example demonstrates how each channel captures different aspects of the price movement: - Channel 1 produces large positive values when prices reverse - Channel 2 produces large positive values when prices trend in the same direction - Channel 3 simply captures the level of the first day in each window

The subsequent ReLU activation function and residual connection will further process these values before they move to the next layer.