Advanced Methods for Portfolio and Risk Management - 20836
Final Project

Bahcivanoglu, Efecan
efecan.bahcivanoglu@studbocconi.it

May 19, 2024

Contents

1 Introduction

Remarkable progress of machine learning algorithms have never been more evident. This progress not only enhanced science; but, also had great impact in our daily lives. In this project our goal is to use Variational Autoencoders for asset pricing, sepcifically conditional on the risk factors. This paper can be divided into three parts; asset pricing, variatonal autoencoders and the application. First we'll cover some basic theory regarding the asset pricing. Since these topics have been explored during the course our summary will be at a fundamental level. Furthermore, great focus will be given to the Variational Autoencoders.

2 Emprical Asset Pricing

2.1 Capital Asset Pricing Model (CAPM)

The Capital Asset Pricing Model (CAPM) describes the relationship between systematic risk and expected return for assets. The expected return on a security is given by:

$$E(R_i) = R_f + \beta_i(E(R_m) - R_f) \tag{2.1.1}$$

where:

- $E(R_i)$ is the expected return on the security.
- R_f is the risk-free rate.
- β_i is the beta of the security.
- $E(R_m)$ is the expected return of the market.
- $E(R_m) - R_f$ is the market risk premium.

2.2 Roll's Critique

Richard Roll's critique points out that empirical tests of the CAPM are problematic because the true market portfolio is unobservable. The proxies used for the

market portfolio may not be mean-variance efficient, potentially leading to flawed conclusions about the CAPM's validity.

2.3 Black's CAPM

Black's CAPM extends the traditional CAPM by eliminating the need for a risk-free asset. Instead, it introduces a zero-beta portfolio, resulting in the following expected return formula:

$$E(R_i) = E(R_z) + \beta_i(E(R_m) - E(R_z)) \tag{2.3.1}$$

where:

- $E(R_i)$ is the expected return on the security.
- $E(R_z)$ is the expected return of the zero-beta portfolio.
- β_i is the beta of the security.
- $E(R_m)$ is the expected return of the market.

2.4 Ross's Arbitrage Pricing Theory (APT)

Stephen Ross's Arbitrage Pricing Theory (APT) models asset returns as a linear function of multiple macroeconomic factors. The expected return on a security is:

$$E(R_i) = R_f + \sum_{k=1}^{n} \beta_{ik}\lambda_k \tag{2.4.1}$$

where:

- $E(R_i)$ is the expected return on the security.
- R_f is the risk-free rate.
- β_{ik} is the sensitivity of the security to factor k.
- λ_k is the risk premium for factor k.

2.5 Fama-French Three-Factor Model

The Fama-French Three-Factor Model expands on the CAPM by including size and value factors. The expected return on a security is:

$$E(R_i) = R_f + \beta_i(E(R_m) - R_f) + s_i \cdot \text{SMB} + h_i \cdot \text{HML} \tag{2.5.1}$$

3 Autoenconders

Finding data representations that are useful for a variety of applications is one of deep learnings main goals. Auto-associative neural networks or autoencoders are one well-established technique for learning internal representations. This kind of network is trained to generate an output y that closely resembles the input x and it has an equal number of output units as input units. An internal layer of the neural network generates a representation z(x) for every new input after training. An encoder changes the input x into a hidden representation z(x) and a decoder returns the hidden representation to the output y(z). These two parts make up this network. Some constraints must be used to make sure an autoencoder finds meaningful solutions. Limitations can be applied by requiring z to have a sparse representation or by restricting the dimensionality of z relative to x. Alternatively the training procedure can be changed to force the network to rectify corruptions in the input vectors like additive noise or missing values in order to train the network to find meaningful solutions. For the network to produce good training results these limitations motivate it to find important data structures.

3.1 Deterministic Autoencoders

Principal component analysis is one of the well-known transformations that linearly transforms an input vector into a lower-dimensional manifold. We can use nonlinear neural networks to define a form of nonlinear PCA, where the latent manifold is no longer a linear subspace of the data space. This is done by using a network with the same number of outputs as inputs and optimizing the weights to minimize some measure of the reconstruction error between inputs and outputs for a set of training data.

3.1.1 Linear Autoencoders

The network's goal is to map the input onto itself, forming an autoassociative mapping. Since the hidden layer has fewer units than the input layer, perfect reconstruction of all inputs is not generally possible. Therefore, the network parameters are determined by minimizing an error function:

$$E(\mathbf{w}) = \frac{1}{2} \sum_{n=1}^{N} \|\mathbf{y}(\mathbf{x}_n, \mathbf{w}) - \mathbf{x}_n\|^2. \tag{3.1.1}$$

The limitations of a linear manifold might be overcome by using nonlinear activation functions for the hidden units. However, even with nonlinear hidden units, the minimum error solution is still the projection onto the principal component subspace. This means there is no advantage in using two-layer neural networks for dimensionality reduction over standard PCA techniques like singular value decomposition (SVD).

3.1.2 Deep Autoencoders

The scenario differs when additional nonlinear layers are added to the network. Consider a four-layer auto-associative network. In this network, the first and third layers have sigmoidal nonlinear activation functions, while the second layer units are linear and fewer than the input units. The network is trained by minimizing the sum-of-squares error.

This setup defines two successive functional mappings: the first maps the original high-dimensional data onto an M-dimensional subspace, and the second layer maps this subspace back to the original input space. This two-step mapping is very general and not limited to linear transformations.

Such a network effectively performs a nonlinear form of PCA, avoiding the constraints of linear transformations while still producing meaningful low-dimensional representations.

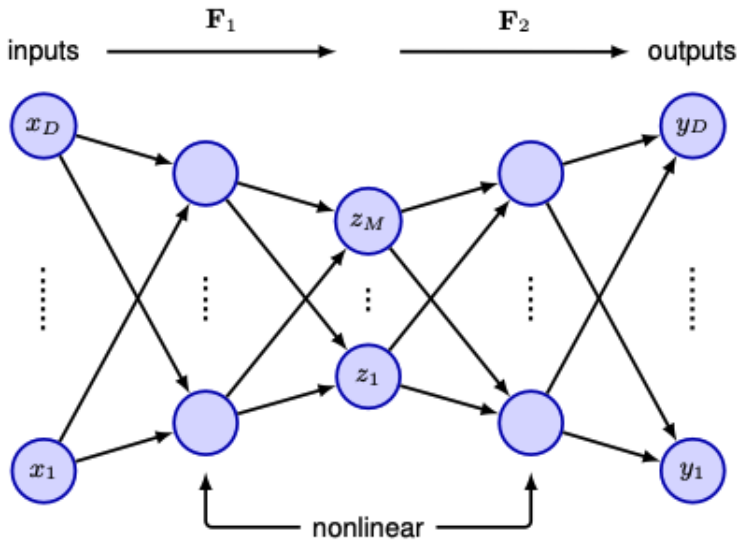


Figure 1: Adding extra hidden layers of nonlinear units produces an auto-associative network, which can perform a nonlinear dimensionality reduction.

3.1.3 Sparse Autoencoders

Instead of limiting the number of nodes in one of the hidden layers in the network, an alternative way to constrain the internal representation is to use a regularizer to encourage a sparse representation, leading to a lower effective dimensionality. A simple choice is the L_1 regularizer since this encourages sparseness, giving a regularized error function of the form:

$$\tilde{E}(\mathbf{w}) = E(\mathbf{w}) + \lambda \sum_{k=1}^{K} |z_k|, \tag{3.1.2}$$

where $E(\mathbf{w})$ is the unregularized error, and the sum over k is taken over the activation values of all the units in one of the hidden layers. Note that regularization is usually applied to the parameters of a network, whereas here it is being used on the unit activations. The derivatives required for gradient descent training can be evaluated using automatic differentiation, as usual.

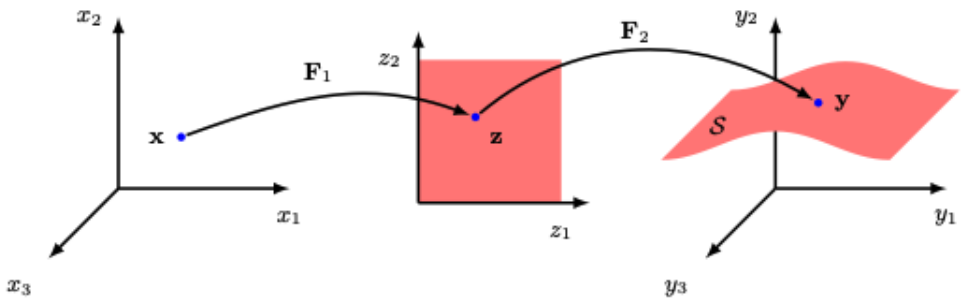


Figure 2: Geometrical interpretation of the mappings performed by the network in Figure 1 for a model with D = 3 inputs and M = 2 units in the second layer. The function F2 from the latent space defines the way in which the manifold S is embedded within the higher-dimensional data space. Since F2 can be nonlinear, the embedding of S can be non-planar, as indicated in the figure. The function F1 then defines a projection from the original D-dimensional data space into the M-dimensional latent space.

4 Application - Python

Python code is created with a guidance from Machine Learning for Algorithmic Trading by Stefan Jansen with necessary modificiations. The stock data and the training, validation and test data are also different: NYSE is used for this notebook. Furthermore, Alphalens library is used to analyze the predictions from a alpha perspective. One detail is, modification to the source of Alphalens had to be done so utils.py file is added to the notebook.We import the OHLCV, sector, shares outstanding and market capitalization data from New York Stock Exchange using yahoo finance. After getting rid of the ETFs we filter out the sectors that has less than 50 stocks. Then we calculate the risk factors, more specifically:

- **Price Trends**
  - Short-Term Reversal: 1-month cumulative return
  - Stock Momentum: 11-month cumulative returns ending 1-month before month end
  - Momentum Change: Cumulative return from months t-6 to t-1 minus months t-12 to t-7.

  - Industry Momentum: Equal-weighted avg. industry 12-month returns
  - Recent Max Return: Max daily returns from calendar month t-1
  - Long-Term Reversal: Cumulative returns months t-36 to t-13.

- **Liquidity Metrics**
  - Turnover: Avg. monthly trading volume for most recent three months scaled by number of shares; we are using the most recent no of shares from yahoo finance
  - Turnover Volatility: Monthly std dev of daily share turnover
  - Log Market Equity: Natural log of market cap at end of month t-1
  - Dollar Volume: Natural log of trading volume time price per share from month t-2
  - Amihud Illiquidity: Average of daily (absolute return / dollar volume)

- **Risk Measures**
  - Return Volatility: Standard dev of daily returns from month t-1.
  - Market Beta: Estimated market beta from weekly returns and equal weighted market returns for 3 years ending month t-1 with at least 52 weeks of returns.
  - Beta Squared: Market beta squared
  - Idiosyncratic return volatility: Standard dev of a regression of residuals of weekly returns on the returns of an equal weighted market index returns for the prior three years.

We'll be feeding these inputs into the Varitional Autoencoder. Our goal is as explained below using a sparse representation in the hidden space we are trying to have a non-linear transformation of the risk factors that'll capture the structure of the risk factors. Jupyter notebook contains explanation of the code. Within in each characteristics we apply a quantile transformation with 200 quantiles to rank normalize the inputs.

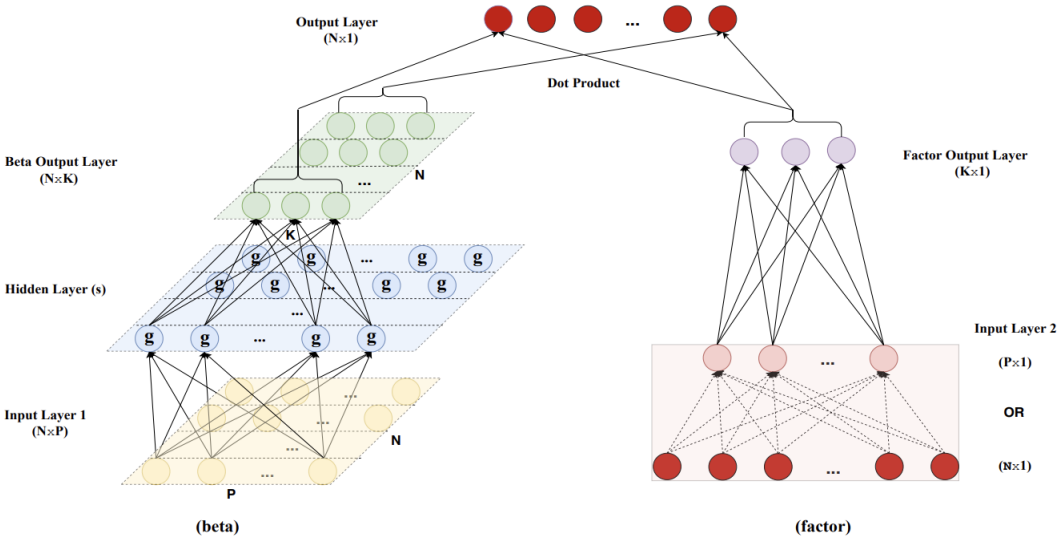


Figure 3: Variational Autoencoder Network Structure. This figure presents the diagram of an autoencoder augmented to incorporate covariates in the factor loading specification. The left-hand side describes how factor loadings β_{t-1} at time $t-1$ (in green) depend on firm characteristics Z_{t-1} (in yellow) of the input layer 1 through an activation function g on neurons of the hidden layer. Each row of yellow neurons represents the $P \times 1$ vector of characteristics of one ticker. The right-hand side describes the corresponding factors at time t. f_t nodes (in purple) are weighted combinations of neurons of the input layer 2, which can either be P characteristic-managed portfolios x_t (in pink) or N individual asset returns r_t (in red). In the latter case, the input layer 2 is exactly what the output layer aims to approximate, which is the same as a standard autoencoder.

Hidden Layer (Dense): The first dense layer with 8 neurons receives input from input-beta. This layer likely serves as a compression mechanism, reducing dimensionality from the original input space to a more compact representation.Batch Normalization: This layer follows the hidden dense layer and is typically used to standardize the inputs to the next layer by adjusting and scaling activations. It helps in speeding up training and achieving better performance.Output Layers: n-factor in the output layer represents the dimension of the data after decoding was applied. Later we'll find the best n-factors for model to perform. Model is fitted using daily data from starting from 1993 to 2021.We use spearman rank correlation between the actual returns and the predicted returns as the information

criteria. Models with the highest 5 ic_mean score are 16 units and 5 factor model.

```
Model: "model"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 input_beta (InputLayer)        [(None, 500, 15)]    0           []

 hidden_layer (Dense)           (None, 500, 8)       128         ['input_beta[0][0]']

 batch_norm (BatchNormalization  (None, 500, 8)      32          ['hidden_layer[0][0]']
 )

 input_factor (InputLayer)      [(None, 500)]        0           []

 output_beta (Dense)            (None, 500, 3)       27          ['batch_norm[0][0]']

 output_factor (Dense)          (None, 3)            1503        ['input_factor[0][0]']

 output_layer (Dot)             (None, 500)          0           ['output_beta[0][0]',
                                                                  'output_factor[0][0]']

==================================================================================================
Total params: 1,690
Trainable params: 1,674
Non-trainable params: 16
__________________________________________________________________________________________________
```

Figure 4: Model Summary

		units	n_factors	epoch	ic_mean	ic_daily_mean	ic_daily_median
n_factors	**units**						
5	**16**	16	5	0	0.363616	0.136418	0.143673
	16	16	5	1	0.360924	0.136172	0.139040
	16	16	5	2	0.356650	0.134874	0.142294
	16	16	5	3	0.354313	0.134496	0.137511
	16	16	5	4	0.348498	0.135062	0.134965

Figure 5: Top 5 Models

5 Predictions and Alpha Analysis

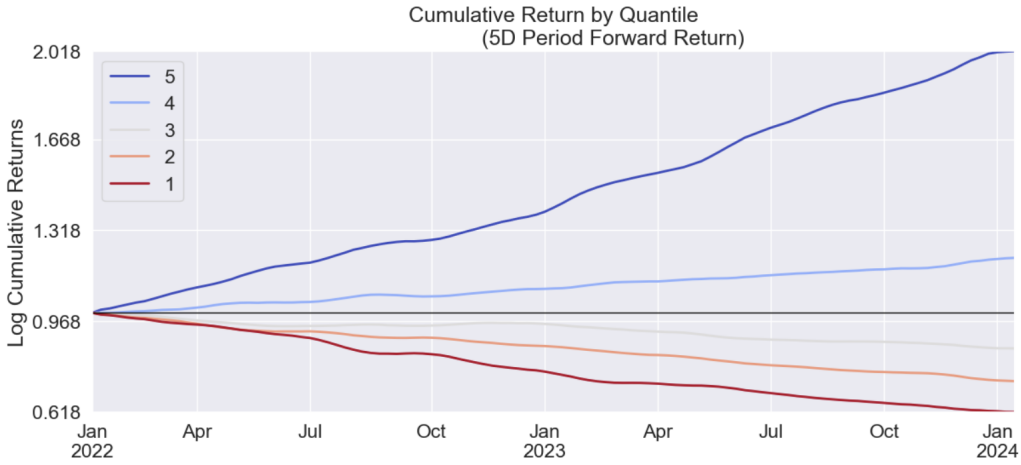
In this section we'll use the hyperparameters and the model weights we received from the cross validation and produce predictions for the time period: 07-01-2022 to 17-05-2024. We'll conduct portfolio analysis by using breaking points to create 5 quantile portfolios and shorting the bottom and longing the top decile. We have analysed the result of this strategy based on 5-days, 10-days and 21-days periods

to compute the forward returns. 5-days period returns are superior to others. Selected graphs are presented here but one can find more in the notebook.

Information Analysis

	5D	10D	21D
IC Mean	0.248	0.170	0.135
IC Std.	0.185	0.145	0.121
Risk-Adjusted IC	1.341	1.172	1.114
t-stat(IC)	13.545	11.841	11.251
p-value(IC)	0.000	0.000	0.000
IC Skew	-0.168	-0.049	-0.087
IC Kurtosis	-0.255	0.540	0.849

Figure 6: Information Analysis

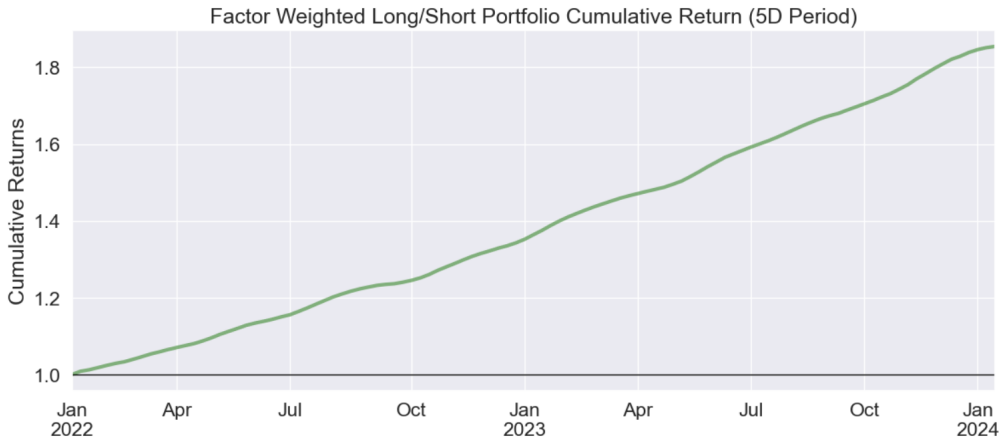


Quantiles Statistics

factor_quantile	min	max	mean	std	count	count %
1	-1.507620	0.268162	-0.657784	0.215718	10151	20.067214
2	-1.185615	0.425635	-0.497399	0.312712	10098	19.962439
3	-1.061077	0.642897	-0.354591	0.381362	10107	19.980231
4	-0.896839	0.832944	-0.164664	0.414543	10098	19.962439
5	-0.771365	1.330265	0.184175	0.387370	10131	20.027676

Returns Analysis

	5D	10D	21D
Ann. alpha	3.244	1.041	0.427
beta	0.122	0.055	0.086
Mean Period Wise Return Top Quantile (bps)	340.371	162.614	89.794
Mean Period Wise Return Bottom Quantile (bps)	-223.164	-103.708	-65.947
Mean Period Wise Spread (bps)	563.535	266.366	155.541



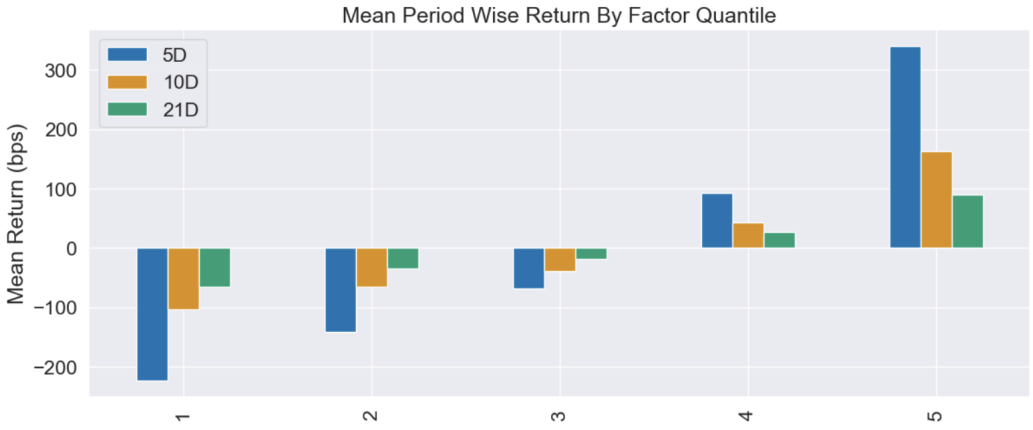


Figure 7: Results