An Empirical Study of Autoencoder Asset Pricing Models and the Impact of Arbitrage Constraints

Master Thesis in Finance at the Stockholm School of Economics

Fall semester 2021

Authors

Benjamin von Essen[1]

Haohang Wu[2]

Abstract

Following Gu et al. (2021), we implement a state-of-the-art machine learning asset pricing model, the *conditional autoencoder*, to capture the time-varying interactions between observable stock characteristics and factor loadings, while simultaneously extracting latent factors from stock returns. Unlike their no-arbitrage model setup, we explicitly compare the out-of-sample performance between arbitrage models and their no-arbitrage counterparts. Our results show that arbitrage models exhibit better predictive out-of-sample performance, which indicates that stock characteristics proxy not only for factor risk premia, but also compensation for mispricing.

Acknowledgments: We would like to thank Professor Adrien d'Avernas for his excellent guidance and support throughout our thesis writing process.

Keywords: Empirical asset pricing, Conditional asset pricing model, Machine learning, Arbitrage, Multi-factor model, Dimensionality reduction, Nonlinear model

Supervisor: Adrien d'Avernas, Assistant Professor in Finance at the Stockholm School of Economics

[1]41945@student.hhs.se

[2]41947@student.hhs.se

I. Introduction

Over the past few decades, researchers have introduced multiple models to explain the variations in cross-sectional stock returns. Recent literature has argued that the differences in excess returns result from compensation of risk exposures and mispricings (Kelly et al., 2019; Chu et al., 2020).

From the basic asset pricing equation $E_t[m_{t+1}r_{i,t+1}] = 0$, risk premia on assets $r_{i,t+1}$ can be decomposed into the quantity of risk β times the price of risk λ as follows[1]:

$$E_t[r_{i,t+1}] = \beta_{i,t}^{'}\lambda_t \ . \tag{1}$$

If the pricing kernel m_{t+1} is driven by multiple factors and is linear in factors f_{t+1}, then equation (1) can be transformed into a factor model:

$$r_{i,t+1} = \alpha_{i,t+1} + \beta_{i,t}^{'}f_{t+1} + \epsilon_{i,t+1} \ , \tag{2}$$

where β_i is asset i's factor loadings to risk factor f, ϵ_i is the idiosyncratic risk associated to asset i and α_i stands for the arbitrage opportunity. Typically, empirical asset pricing analysis assumes $E_t[\epsilon_{i,t+1}] = E_t[\epsilon_{i,t+1}f_{t+1}] = 0$, and $E_t[f_{t+1}] = \lambda_t$ for all i and t.

When applying the above multi-factor model in empirical analysis, one of the most difficult challenges is that the factor loadings β and risk factors f in equation (2) are unobservable. Traditionally, researchers either pre-specify a factor structure using economic intuition and focus on estimating factor loadings, or they neglect the dynamic mechanism in the model and use statistical methods to compute a static factor structure. However, both approaches contain obvious drawbacks and thus struggle with describing the true factor structure.

One way to tackle these challenges is to utilize *machine learning*. As described by Israel et al. (2020), machine learning thrives in big data environments and provides methods to handle "large" models with many explanatory variables and nonlinear associations between explanatory and response variables. These features make machine learning methods a good fit for describing equity return variations in high-dimensional factor contexts. Furthermore, the field of machine learning provides multiple dimensionality reduction techniques, such as principal component analysis and autoencoders. These techniques can compress a large number of input variables to a lower dimensional latent space while retaining most of the information, which is useful for building more

[1]For detailed proof, see Chapter 6 in Cochrane (2009).

parsimonious models. In recent years, the field of asset pricing has seen an increase in the use of machine learning methods, with papers such as Gu et al. (2020) and Gu et al. (2021) providing empirical support of their effectiveness.

In this thesis, we aim to overcome the shortcomings of traditional asset pricing techniques by utilizing state-of-the-art machine learning models to estimate the multi-factor model given by equation (2). We estimate latent factors and their corresponding factor loadings, without ex ante knowledge of the factor structure, from a large number of observable stock characteristics. Furthermore, we aim to examine Gu et al. (2021)'s claim that stock characteristics only serve as a proxy for risk premia, and do not contain information about potential mispricings. To do this, we compare the performance of aforementioned state-of-the-art models with and without arbitrage constraints. In doing so, we to explore the importance of α in equation (2) and its relationship to observable stock characteristics.

Inspired by Gu et al. (2021) (GKX hereinafter), we follow their methodology to build a *conditional autoencoder* (CAE) model, which accounts for the impact of stock characteristics on factor loadings and extracts latent factors from returns data. Our model differs from the original GKX model in that we assume the return predictability does not come solely from factor risk exposures. Instead, we assume that there exist portions that cannot be explained by risk exposures alone and introduce an intercept, which translates to $\alpha \neq 0$ in equation (2). Thus, our model does not adhere to the no-arbitrage constraint.

In the empirical analysis, we follow Kelly et al. (2019) (KPS hereinafter) and use the same measures to evaluate our model's ability to describe the contemporaneous factor-return relationship and predict future stock returns. To examine our model's performance under an economic setting, we build equal-weighted long-short portfolios based on predicted individual stock returns and compare the Sharpe ratios of the portfolios. Lastly, we carry out Monte Carlo simulations to showcase our model's performance on random samples.

The remainder of this thesis is organized as follows: Section II provides literature review on empirical asset pricing and machine learning and Section III presents the model used in this thesis. The empirical findings and discussion are presented in Section IV. In the end, Section V concludes the thesis. Appendix A gives a detailed description of the variables used in the empirical study.

II. Background

A. Empirical asset pricing

One of the fundamental goals of asset pricing is to understand the cross-sectional variations in average stock returns. Since the introduction of CAPM by Sharpe (1964), Lintner (1965) and Black (1972), theorists have proposed many models to study the behavior of stock returns. In the meantime, empiricists have applied those theories in statistical models and tested their explanatory power of real world stock returns.

Traditionally, observable factors have been used in empirical studies. There are two classical approaches to estimating factor risk premia with observable factors, both based on linear regression.

The first approach originated from the seminal work by Fama and MacBeth (1973), in which they devise a two-pass regression to test CAPM. In the first pass, they use time-series regression on sorted portfolios to estimate factor loadings, β. In the second pass, they run cross-sectional regressions on rolling windows to obtain a time-series of factor risk premia, and test if the mean of the factor risk premia is significant. One contribution of the Fama-MacBeth regression is that they replace individual stocks with sorted portfolios to estimate β, which alleviates the errors-in-variables issue when using estimated β as explanatory variable in the second pass regression. Since then, Fama-MacBeth two-pass regression has become a standard practice in estimating factor risk premia.

The second approach is best exemplified in the estimation of the three-factor model (FF3) from Fama and French (1993). Fama and French choose market capitalization and book-to-market ratio to construct double sorted portfolios. Then they use these portfolios to mimic true factors, and compute the value (HML) and size (SMB) factor risk premia. In their final step, factor loadings are estimated by running time-series regression on these factor risk premia. Their way of using sorted portfolios to estimate factor risk premia has been widely used in later empirical studies to explain variations in average stock returns (Carhart, 1997; Novy-Marx, 2013; Fama and French, 2015).

It should be noted that these classical linear regressions all require an explicit factor structure in order to do the estimation. This predetermined structure is often elusive even for well-defined risk factors such as the market factor in CAPM. As Roll (1977) points out, it is not clear which assets should be included in the market portfolio. In the case of the Fama-French factor model, Fama and French (1993) also admit that the choice of the factors' exact form is "somewhat arbitrary". To make matters worse, the emergence of the "factor zoo" has greatly increased the chance of

omitting variables in traditional linear regression models, which causes the estimator to be biased and inconsistent (Giglio and Xiu, 2021).

To bypass the drawbacks of traditional factor models, some researchers propose to instead treat true factors as latent and apply factor analysis to estimate the underlying factors and loadings. A classic technique in this approach is *Principal Component Analysis (PCA)*. Chamberlain and Rothschild (1983) take the lead to implement it for a stock returns dataset in order to describe the factor structure. However, there exist two main shortcomings to the classical PCA method. First of all, the extracted principal components are often exposed to different input variables and it is very difficult for researchers to draw meaningful interpretations from them. Second, the classical PCA method can only capture static relationships within the model, and this restricts its ability to estimate dynamic models, such as time-varying factor loadings in conditional asset pricing models.

There are several recent papers trying to tackle these shortcomings. In particular, Kelly et al. (2020) and Kelly et al. (2019) propose a method named *instrumented principal components analysis (IPCA)*, in which they set firm characteristics as instrument for the time-varying factor loadings. Their papers do not make presumptions about the exact factor structures in the model. Instead, they rely on statistical techniques (e.g., PCA) to uncover latent factors. By setting firm characteristics as instrument, KPS succeed in modeling dynamic factor loadings as a function of the time-varying stock characteristics. This approach acknowledges the popular view in recent literature that firm characteristics might be a good proxy for factor loadings (Jegadeesh et al., 2019; Pukthuanthong et al., 2019). The empirical results of IPCA have shown better explanatory power on US stock data than classical models such as CAPM and FF3. The centerpiece assumption of the IPCA model is that both factor loadings and latent factors are linearly determined. However, while this assumption is convenient, it is also highly questionable (Campbell and Cochrane, 1999; Bansal and Yaron, 2004).

Gu et al. (2021) propose another latent factor model, the *conditional autoencoder*, to account for the nonlinearities between observable stock characteristics and factor loadings. In their model, they replace the linear relationship in the beta specification with a more complex and possibly more realistic nonlinear one. Their model's return prediction procedures resemble IPCA, in which they split the return prediction into two parts: the factor loadings specification and the latent factors formulation.For the factor loadings specification, GKX use a *deep neural network (DNN)* to model the nonlinear relationship between factor loadings and asset characteristics. For the latent factor formulation, they use an *autoencoder* to generate latent factors from realized returns. In the end,

GKX multiplies the estimated factor loadings and the latent factors to produce the asset returns. The network is trained to output the same returns it receives as input. GKX show that IPCA is equivalent to a conditional autoencoder with a linear factor loadings network. Their empirical results also demonstrate that their model achieves far better out-of-sample performance in both real-world data and simulated data compared to other factor models, such as the Fama-French factor model and IPCA.

Both Gu et al. (2021) and Kelly et al. (2019) argue that stock characteristics most likely proxy for compensation from risk factor exposures, rather than that from mispricing opportunities. GKX draw this conclusion since they assume *no-arbitrage* in their model setup, as the CAE lacks an intercept, and find that their models perform quite similar to the best machine learning model in Gu et al. (2020), which is unconstrained in terms of arbitrage. They strengthen their argument by studying the significance of their no-arbitrage model's residuals for different assets. Their empirical results fail to reject the null hypothesis of zero average residuals for most, but not all, of the assets and thus corroborate their no-arbitrage assumption.Kelly et al. (2019) came to the same conclusion by evaluating the IPCA empirically both with and without an intercept. Though they find that the introduction of an intercept into the one-factor model can significantly improve return predictability, the significance of the intercept quickly falls to effectively zero as more factors are included in the model.

There are however studies suggesting that mispricing plays a big role in return prediction. Chu et al. (2020) leverage the Regulation SHO pilot program[2], which temporarily lightened short-selling constraints for a number of stocks, to study the effect of limits to arbitrage on common asset pricing anomalies. They find that decreased short-selling constraints weakened the anomalies, but only on the short-leg of the portfolios. Furthermore, they argue that their findings provide a strong confirmation of anomalies reflecting mispricing and point to a casual link between limits to arbitrage and anomalies.

B. Machine learning

Machine learning can be described as the methods used to construct computer programs that use experience to automatically improve, and thereby "learn" (Mitchell et al., 1997). In this context, experience is often synonymous with *data*, and thus, the field of machine learning is intimately

[2]Regulation SHO pilot program was introduced by SEC to investigate whether short-selling restrictions imposed in NYSE, Amex, and NASDAQ affect the market quality. This pilot program was conducted between July 28, 2004 and August 6, 2007.

connected with the fields of data analysis and statistics (Mohri et al., 2018).

The field of machine learning includes a vast array of techniques. In recent years, deep learning has emerged as a popular family of machine learning models. Among the techniques of deep learning, we find deep neural networks such as multi-layer perceptrons and autoencoders.

One of the fundamental goals of machine learning is to achieve *generalization*, the power to make accurate predictions when confronted with unseen data (Mohri et al., 2018). Thus, a separation of training data, *past* information, and unseen data, *new* information, is needed.

The strive for generalization introduces us to the problem of over- and underfitting. *Overfitting* occurs when a model is fit too closely to the training data, so that a model that is fit less well to the training data would actually outperform the better fit model on the entire distribution of data, including unseen data (Mitchell et al., 1997). For example, we overfit when we learn the noise in the training data too well and therefore generalize worse to unseen data, where that exact sequence of noise is not present. Other scenarios that can result in overfitting is when the training data contains many outliers and when the training data is not diverse enough; the distribution of the training data is seldom exactly the same as the true distribution, since the training data is but one possible realization of the true distribution. *Underfitting* is the opposite of overfitting, and thus a model is underfit if it would perform better on the entire distribution of data if it was better fit on the training data. Striking a balance between over- and underfitting is a fundamental problem in the field of machine learning (Maki, 2020).

Related to the the challenge of overfitting is the the *bias-variance tradeoff*. One can derive that the expected value of the mean squared error, *MSE*, of an estimated model's prediction, $\hat{f}_D(x)$, compared to the actual *true* model, $f(x)$, can be decomposed as in equation (3)

$$E_D[MSE] = E_D[(\hat{f}_D(x) - f(x))^2] = \underbrace{\left(E[\hat{f}_D(x)] - f(x)\right)^2}_{bias^2} + \underbrace{E_D\left[\left(\hat{f}_D(x) - E[\hat{f}_D(x)]\right)^2\right]}_{\text{variance}}, \quad (3)$$

where $E_D[\cdot]$ is the expectation with respect to different sample sets (Maki, 2020). Since both the squared bias and the variance are positive, it is easy to see that to minimize the MSE, and thus maximize the performance of the model, one must simultaneously minimize the magnitude of the bias and the variance. A key parameter in model selection is model *complexity*, which we can use to minimize the MSE.

While one could imagine that a more complex model should always perform better, the bias-variance tradeoff tells us that it is not the case. When we make a model more complex, the

magnitude of the bias decreases while the variance increases (Friedman et al., 2001). This is because the new complexity allows the model to become more *sensitive* to data and thus improve its fit to the training data. The opposite is also true, if we decrease complexity, the magnitude of the bias increases, while the variance decreases. To find the optimal complexity, one could test different models and compare how similar they are to the *true* model, as in equation (3). However, in practice we will never now what the true model is. Thus, we instead use the *generalization error*, which is the MSE between the prediction and some test data. The expected value of the generalization error is very similar to equation (3), with the addition of the irreducible error caused by the test data itself. To compare different models test performance, it is common to split the dataset into three separate subsets (as long as the amount of data is sufficient) (Friedman et al., 2001):

1. Training set, which is used for training the models.

2. Validation set, which is used for comparing the different models, mainly by calculating the *validation* error, i.e., the generalization error using the validation set.

3. Test set, which is used for evaluating the generalization performance of the final model, mainly by calculating the *test* error, i.e., the generalization error using the test set.

Empirical data shows that the optimal complexity lies somewhere in the middle and not by either extreme, as illustrated in Figure 1 (Friedman et al., 2001). Thus, it is important to let go of the quite common gut feeling that bias should be avoided, since that could lead to a model with too high variance, which would be too dependent on the data used. Instead, we should strive to match the model complexity with the available data resources (Maki, 2020).

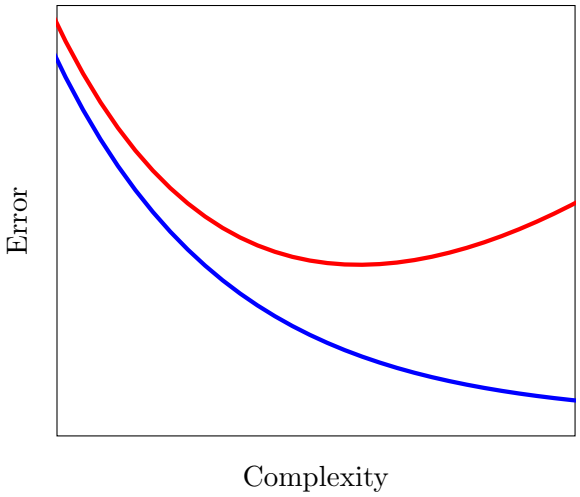

Figure 1. Error as a function of complexity, with training error in blue and test error in red.

While large datasets are often seen as a blessing, it has been known for a long time that datasets with a large number of dimensions present us with an array of, often counter intuitive, problems; Already in 1961, the term *the curse of dimensionality* was used in Bellman (1961). The following are some of the problems that arise with the use of high-dimensional data:

- As the number of dimensions increases, estimating a function dependant on those dimensions becomes increasingly difficult, as the number of data samples needed to produce a low-variance estimate increases exponentially (Lee and Verleysen, 2007). This is often referred as the *empty space phenomenon* (Lee and Verleysen, 2007).

- Since in practice, datasets contain a limited amount of observations, high dimensional data is *sparse* (Lee and Verleysen, 2007).

- When the number of dimensions increases, the relationship between volumes of multi-dimensional spheres and cubes changes in unexpected ways. For example, one can show that a high-dimensional sphere's volume becomes insignificant in comparison to a corresponding high-dimensional cube's (Lee and Verleysen, 2007). In fact, not only does more and more of the hypercube's volume get concentrated in the edges, or "spikes", but the sphere's volume actually goes to 0 as the number of dimensions increase (Lee and Verleysen, 2007). Furthermore, a high-dimensional sphere's volume is concentrated in its outermost layer, or "shell" (Wegman, 1990).

  These properties of high-dimensional spaces have direct effects on statistics operating in those domains. A striking example of this is the effect it has on the density of Gaussian distributions.

While in 1 dimension, 95% of the distribution is located within 1.96 units of standard deviation from the mean. However, this distance is not constant with the number of dimensions, as it in 2 dimensions is 2.54 units of standard deviation and in 6 dimensions already has reached 3.54 units of standard deviation (Lee and Verleysen, 2007). Thus, one could say that Gaussian distributions are less dense in higher dimensional space, which echoes our earlier results that shows the volume of spheres decreases with dimensionality. This results in normal data with a given variance being more sparse in higher dimensional spaces and makes many statistical methods less effective.

As high-dimensional data introduces many challenges, it is common to, when presented with high-dimensional datasets, use some dimensionality reduction techniques. These techniques can broadly be separated into two categories:

- Eliminating irrelevant variables. Often, not all variables are relevant for the question at hand and thus some variables can potentially be eliminated. These kinds of techniques are often supervised (Lee and Verleysen, 2007).

- Creating new, condensed variables. Often, variables are related to each other, be it by simple correlation or more complex relationships. When this is the case, it is possible to create new variables, which encompass most of the information in the original variables. Thus the original dataset is projected into a lower dimensional space, while keeping most of the information. However, for this to be possible one most be able to gain insight into the dependencies between the variables.

In this thesis we will focus on the second category of techniques, which creates new variables. Today, there are many such techniques available, many of them being unsupervised (Lee and Verleysen, 2007).

Unsupervised dimensionality reduction techniques fundamentally aim to find and untangle *manifolds*, i.e., P-dimensional structures embedded into $D > P$ dimensions so that it is locally almost "flat" (Lee and Verleysen, 2007). Of these unsupervised techniques, the most well known one is PCA. Furthermore, some of these techniques are deep neural networks, such as the autoencoder.

Principal component analysis, PCA, is a very widely adopted unsupervised *linear* dimensionality reduction technique, which assumes that the data can be represented as a linear transformation of the latent variables. It simultaneously provides the optimal linear solutions to both the problem of maximising the preserved variance and the problem of minimizing the maximization error (as

they are equivalent). Thus, the PCA solution can be derived in multiple ways (Lee and Verleysen, 2007).

Deep neural networks, *DNNs*, are a subtype of artificial neural networks which are studied in the field of *deep learning*. Their defining property is that they are deep, i.e., focused on chaining multiple layers instead of using one wider layer. This property shares similarities with how our brains function and allows for the networks to learn complex, nonlinear behaviors.

One of the most fundamental models in deep learning is the multi-layer perceptron (MLP), also known as a feed-forward neural network (Bishop, 2006), which has been used for many decennia already, see for example Murtagh (1991). As it is one of the most basic DNNs, many other neural network architectures use parts of the MLP and connect them in novel ways. Therefore, it is important to have a solid understanding of the MLP.

The MLP is made of a collection of interconnected *neurons*, also known as *nodes* (Gardner and Dorling, 1998). Each node is in itself quite simple and performs the following function (Bishop, 2006)

$$n_i = \underbrace{g}_{\substack{activation \\ function}}(\sum_{j=1}^{D}(\underbrace{W_{ij}}_{weight}\underbrace{x_j}_{\substack{neuron \\ input}}) + \underbrace{b_i}_{bias}) = g(\underbrace{W_i}_{vector}x + b_i).$$

A number of such neurons are then stacked and form a *layer*, l, which can be represented in matrix notation as

$$n^{(l)} = g^{(l-1)}(\mathbf{W}^{(l-1)}x + b^{(l-1)}), \tag{4}$$

where the superscript indicates the current layer. A number of layers are then composed, put one after another, to form the completed MLP, as illustrated in Figure 2. Note that the input layer is not really a layer, but simply an input, even though it is often referred to as the input layer. The final layer of the network is called the output layer, as it produces the network's output. All other layers are called *hidden layers*, which process intermediate results and are what makes the network deep. However, for the deepness to have any effect, a nonlinear activation function is needed. This is because multiple composed linear layers are mathematically equivalent to a single linear layer (Bishop, 2006). Thus, MLPs use nonlinear activation functions in the hidden layers to allow for nonlinear behavior. Traditionally, the logistic Sigmoid function, which yields a range between 0 and 1, and tanh, which yields a range between -1 and 1, have been common choices. However, in

recent years, the rectified linear unit, $ReLU$,

$$ReLU(x) = max(0, x) \tag{5}$$

has grown in popularity, as its use has been shown to improve performance (Nair and Hinton, 2010).

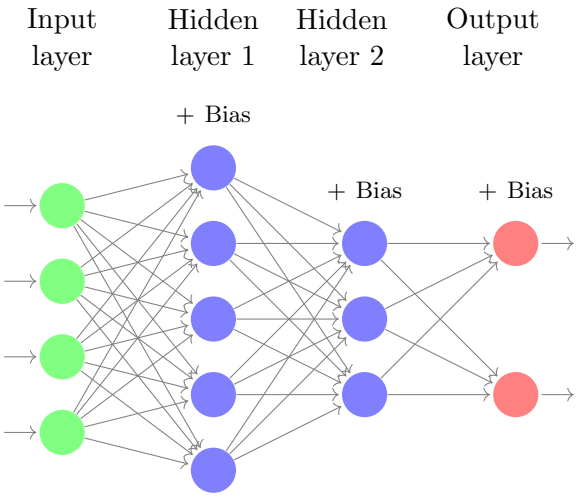

Figure 2. A multilayer perceptron for multivariate regression with 2 hidden layers.

The output of the network can be calculated by propagating the input forward through the network. However, in order to get satisfactory results, the network must first be trained, so that it can learn the correct behavior. Since the MLP is a supervised learning model, one needs training data that contains the correct output for every input in order to train the network. Furthermore, a gradient descent algorithm is used in order to train the network, which leverages the backpropagation algorithm first considered by Rumelhart et al. (1986), performed in the following manner (Gardner and Dorling, 1998; Bishop, 2006):

- Initialize the weights of the network, often with a random seed.
- Propagate the first input through the network, which calculates the output.
- Calculate the *loss*, also known as cost or error, for the given loss function by comparing the output to the correct answer.
- Calculate the gradient of the loss.
- Backpropagate the gradient from the output layer to the first layer via the chain rule.
- Update the weights and biases by subtracting the gradient times the learning rate.

- Repeat, for a number of *epochs*, until the network converges at a local minima or until it starts overfitting.

The gradient descent can be done in a couple of different variations. While it is possible for each epoch to give all the training data as input, this is often very computationally expensive. To tackle this, one can instead use a method called stochastic *gradient descent*, *SGD*. When training by SGD, for each epoch one first shuffles all the samples, and then updates the weights of one mini-batch, i.e., a subset of the data, (or even one sample) at a time (Gu et al., 2021). This approach will result in a less stable descent, due to its random nature. However, for large datasets, it has been shown to converge faster (Le Cun et al., 1989). Furthermore, it has the potential to escape from local minima (Bishop, 2006), which can aid in finding better minima. An important parameter in SGD is the *learning rate*. One prominent algorithm for choosing a suitable, adaptive, learning rate is *Adam* (Kingma and Ba, 2014; Gu et al., 2021).

MLPs can be applied to a number of different problems and work well for both classification and regression (i.e., predicting a value). However, the two problems in general require different setups. It is important to use a suitable activation function in the output layer. For classification, softmax is a popular choice. When doing regression, it is important to choose an activation function that can yield results in the necessary range. Thus, a safe choice for regression is to use a linear activation function in the output layer, as that allows for the network to have output of any magnitude.

Another well known deep neural network architecture is the *autoencoder*, which is used for dimensionality reduction (Gu et al., 2021). The autoencoder is technically equivalent to the above described MLP, with the difference laying in how we choose to use it. Thus, it can be both linear and non-linear. Instead of using training data which needs to contain the correct answer, the autoencoder is able to extract information from unlabelled training data and is thus an unsupervised learning model. The goal of the autoencoder is to learn an internal representation of the data, a *latent* representation, which entails projecting the data to a lower dimension, while still keeping most of the information. The idea behind this is that, for many data sources, the data might be a result of a few key variables which lead to the sample at hand, similar to how many theories in asset pricing theorize that the returns of all existing companies is mainly dependent on a few key factors. To find such internal representations in an unsupervised fashion, one can train a network to produce a output similar to the input. This idea was considered early by Rumelhart et al. (1985). While this might seem like a trivial task, it quickly gets complicated if we want the network to use a internal representation of lower dimensionality than the input. Thus, the autoencoder consists of

an encoder, which maps the input to a *latent space* of lower dimensionality, followed by an decoder that aims to find the inverse of that encoder and thus produce an output similar to the input, as can be seen in Figure 3.

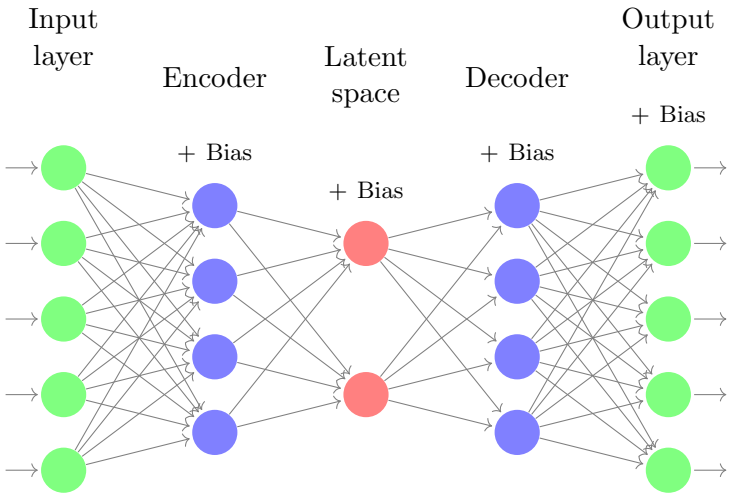

Figure 3. An autoencoder.

The result of interest of this network is thus not its output, but rather the latent representation it creates, which hopefully can "distill" the data into a few key variables.

A multitude of regularization techniques can be used to prevent overfitting when training neural networks. A common approach is to add a penalty term to whatever function is being optimized. Two such penalty terms which are widely used are the L1-norm (LASSO) and the L2-norm (ridge):

$$l_1(\lambda, x) = \underbrace{\lambda}_{regularization\ coefficient} \sum_i |x_i| \ , \tag{6}$$

$$l_2(\lambda, x) = \lambda \sum_i x_i^2 \ . \tag{7}$$

While both methods are popular, LASSO is especially useful for creating sparse models, as it has a tendency to pull parameters to exactly zero, while ridge only pulls them very close to zero (Friedman et al., 2001). Why this is the case can be understood if we instead view the regularization from its equivalent formulation as an optimization under a *constraint*. Figure 4 shows how the LASSO constraint tends to find its optimal solution at a corner, where one parameter is zero. Adding a penalty term, such as the L1-norm, based on the size of the network's weights to the loss function when training a neural network, is called *weight decay* (Bishop, 2006).

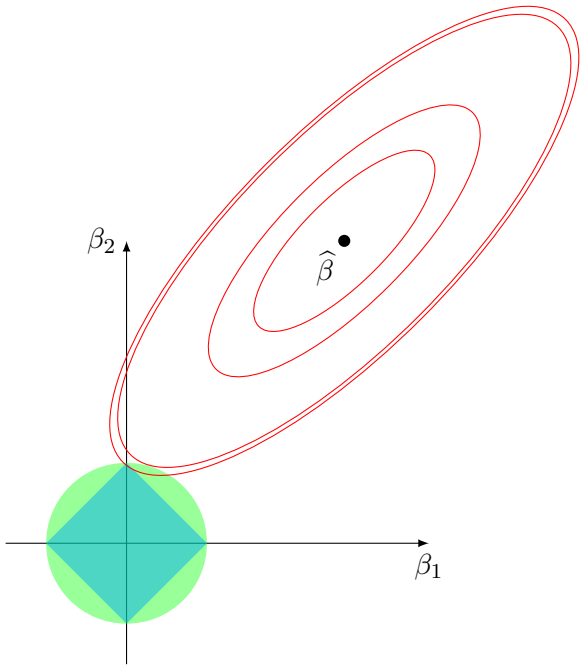

Figure 4. L1 (diamond) vs. L2 (circle) regularization. The filled regions are the respective constraints and the red lines are contour lines of the function being optimized (where contour lines closer to $\hat{\beta}$ are preferred).

III. Method

A. Empirical Data

A.1. Data source

Our empirical data is an updated version of the datasets used in Gu et al. (2020) and Gu et al. (2021). Their original data covers all firms listed in NYSE, AMEX and NASDAQ from March 1957 to December 2020, which contains monthly individual stock returns and 94 firm characteristics commonly used in the return prediction literature. A summary of these firm characteristics is provided in Table VI in the appendix.

By definition, the calculation of most of the characteristics depends on firm filings to the stock exchange. However, there usually exists a delay between the reporting period ending date and the publication date of the financial statement to the general public[3]. To prevent information leakage caused by this delay, GKX assume a one month delay in monthly characteristics, four months delay in quarterly characteristics and six months delay in annual characteristics.

[3]See: *https://www.sec.gov/smallbusiness/goingpublic/exchangeactreporting*
As per SEC regulation, the annual report (Form 10-K) must be submitted within 60 to 90 days of the close of their fiscal year, and the quarterly report (Form 10-Q) must be submitted within 40 days from the end of the quarter.

A.2. Data preprocessing

Constrained by computing capabilities, we take a 41-year slice of the raw data, beginning in January 1980 and ending in December 2020, and based on their frequencies during the sample period we use the top 1000 stocks for our main analysis. We treat the one-month Treasury bill rate from Kenneth R. French's Data Library[4] as the risk-free rate and compute the excess return for all selected stocks.

To control for the impact of outliers in firm characteristics, we follow Kelly et al. (2019) and implement rank normalization on the characteristics data. In particular, for each month and every characteristic, we calculate each stock's cross-sectional percentile rank. We then use equation (8) to transform the range of characteristics values into the range $[-1, 1]$,

$$Z = 2 \times rank\_pct(Z) - 1 \ , \tag{8}$$

where Z denotes the firm characteristics matrix, and $rank\_pct(\cdot)$ is the percentile rank operator.

Apart from normalization, we also tackle the unbalanced panel issue, i.e., that the number of stocks in each month varies greatly. Figure 5 illustrates the number of available stocks over the sampling period. Unlike GKX, we choose to construct a balanced panel by, for each month, generating extra observations for the missing stocks observations. Figure 6 shows the amount of missing values over the sampling period. We fill the excess returns in these generated stock observations with the cross-sectional median of that month's excess returns. For missing values in firm characteristics, we use the same approach, which always takes the value of zero after the rank normalization process.

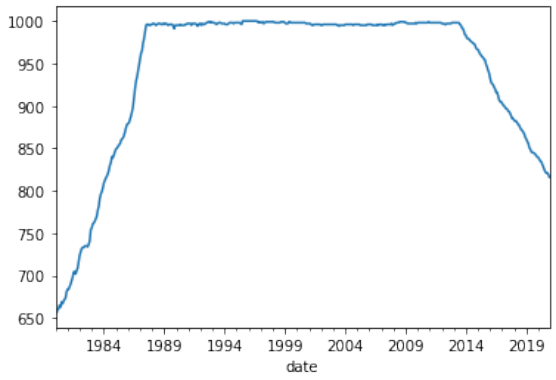

Figure 5. Number of stocks over the sampling period

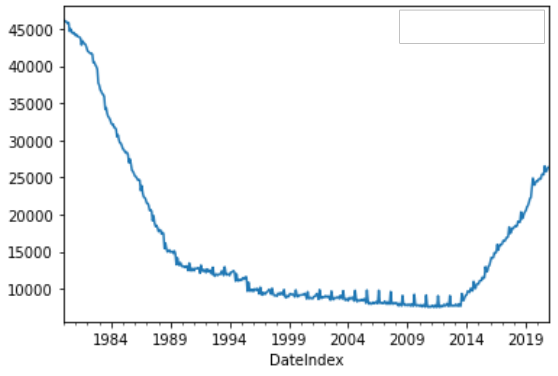

Figure 6. Missing values over the sampling period

[4]See: *http://mba.tuck.dartmouth.edu/pages/faculty/ken.french/data_library.html*

A.3. Sample split

Our data is split into three different subsamples: the training set, the validation set and the test set. The training set is used to fit the model and obtain parameter values and the validation set is used for early stopping, to prevent overfitting. In the end, the test set presents out-of-sample evaluation of the model's performance. Worth noting is that since we use a rolling window approach, the sample split will be different for each iteration.

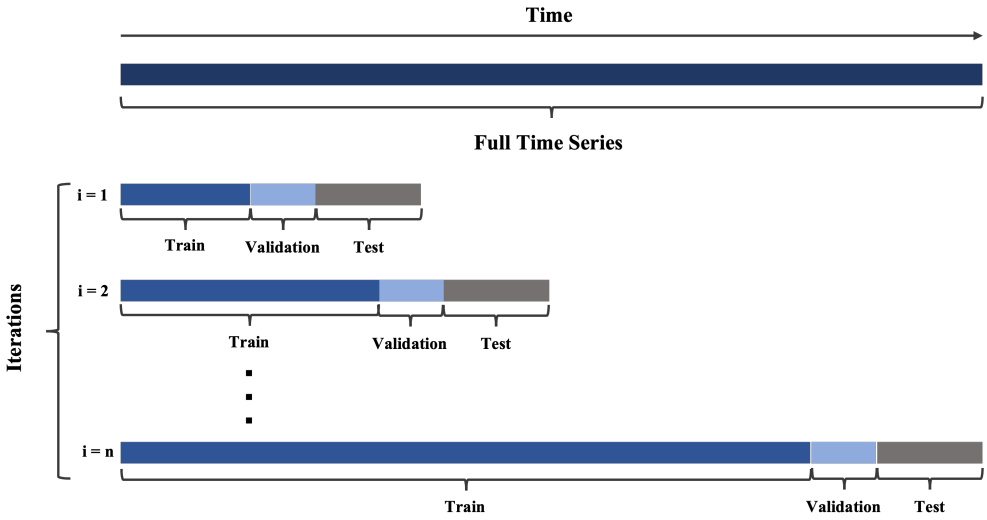

Figure 7. Iterations over entire sample

To initialize our model, we fit it with the training set which ranges from January 1980 to December 1988, totaling 108 months. To avoid large-scale recursive computation, we refit our model every 12 months, using the previous model for initialization of the weights. Figure 7 demonstrates the dynamics of our data split over time. Specifically, in each iteration, we expand our training set by 12 months of data. The size of the validation set remains the same, but it moves forward to instead include the most recent 6 months of data. Accordingly, our test set also moves forward to contain the next 12 months of data.

B. The conditional autoencoder model

Following Gu et al. (2021), we implement the *conditional autoencoder* (CAE), illustrated in Figure 8, to accommodate conditional risk exposures and capture latent risk factors.

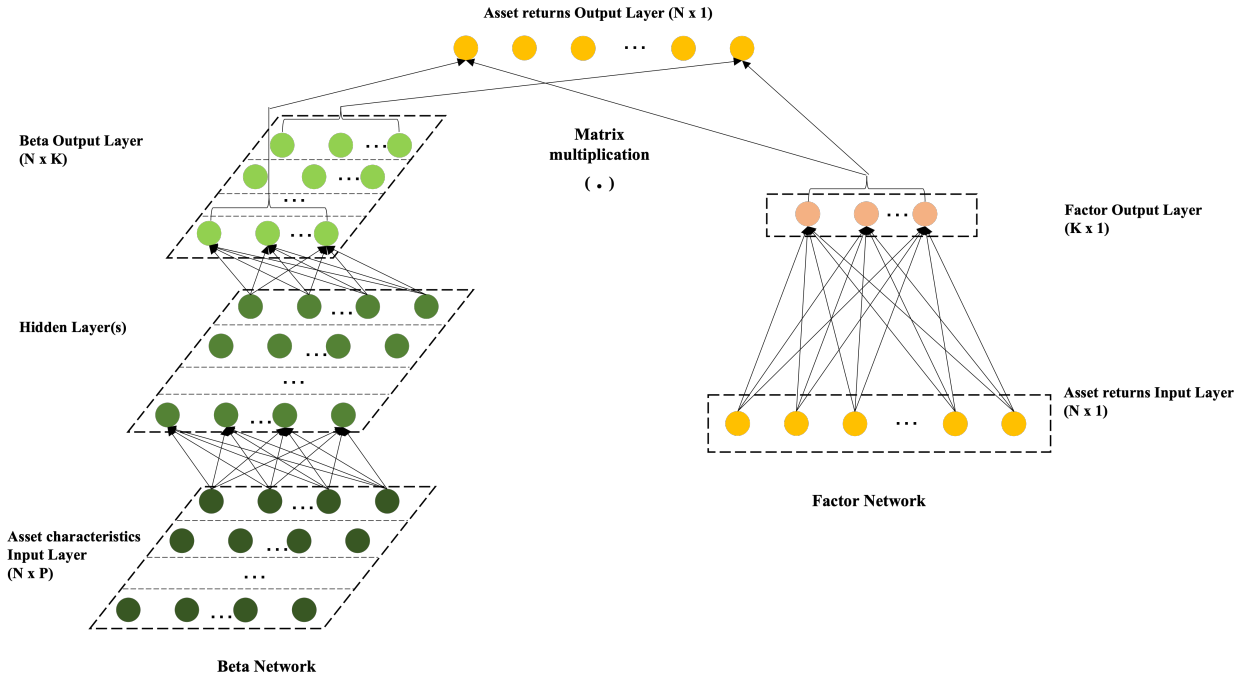

Figure 8. The Conditional Autoencoder (CAE)

The CAE consists of two parts:

1. The *beta network*, which is a deep neural network with the objective of calculating the factor loadings.

2. The *factor network*, which is a linear autoencoder with the objective of calculating the latent factors.

In the beta network, we employ a deep neural network to model the interactions between observable firm characteristics and factor exposures. This is an extension of the MSCI Barra model [5] and IPCA, where non-linearity is introduced into the setup of the time-varying factor loadings. The beta network takes the characteristics matrix at time t as input, which contains all the ranked characteristics for all the selected companies. Equations (9)-(11) shows how the beta network works on a company basis, indicated by the subscript i. Thus, the real network performs the same function but for all the companies in parallel. The input layer, called the asset characteristics input layer, shown in equation (9) (where $z_{i,t-1}$ is a vector), is followed by a number of hidden layers with nonlinear activation functions, as shown in equation (10). These culminate in the beta output layer, which contains each company's K factor loadings (for the K factors created by the factor

[5]MSCI Barra model is a popular method for practitioners to forecast stock returns. See: *https://www.msci.com/www/research-paper/the-barra-us-equity-model-use4/014291992*

network), as seen in equation (11). As the beta network is a stacked MLP, each company's row in each layer (except of course in the asset characteristics input layer) is similar to the layer described in equation (4).

$$z_{i,t-1}^{(0)} = \underbrace{z_{i,t-1}}_{P\times 1} \tag{9}$$

$$z_{i,t-1}^{(l)} = g(b^{(l-1)} + \mathbf{W}^{(l-1)} z_{i,t-1}^{(l-1)}) \quad l = 1, \ldots, L_\beta \tag{10}$$

$$\underbrace{\beta_{i,t-1}}_{K\times 1} = b^{(L_\beta)} + \mathbf{W}^{(L_\beta)} z_{i,t-1}^{(L_\beta)} \tag{11}$$

The risk premia specification is built upon a linear autoencoder, whose encoder we use to extract latent factors. Being an autoencoder, it has the same input and output during training, that is each company's excess returns at time t, the vector r_t. As the factor output layer, shown in equation (13), immediately follows the asset returns input layer, shown in equation (12), without an activation function, the factor output layer is a linear function of the input. This linear function is the network's encoder.

$$r_t^{(0)} = r_t \tag{12}$$

$$\underbrace{f_t}_{K\times 1} = \tilde{b}^{(1)} + \tilde{\mathbf{W}}^{(1)} r_t^{(0)} \tag{13}$$

The network's decoder is also a linear function, as it should "mirror" the encoder. However, this decoder is not made up of a typical neural network layer. Instead, the decoder consists of a matrix multiplication of the factor loadings and the latent factors, similar to the classic multi-factor model approach. This operation merges the two networks into one and calculates the asset returns output layer, seen in equation (14) and (15).

$$\hat{r}_{i,t} = \beta_{i,t-1}^{'} f_t \tag{14}$$

$$\underbrace{\hat{r}_t}_{N\times 1} = \underbrace{\boldsymbol{\beta}_{t-1}}_{N\times K} \underbrace{f_t}_{K\times 1} \tag{15}$$

Thus, the overarching model is identical to the general multi-factor model described earlier in equation (2); stocks expected excess return can be explained by the compensation of holding risks associated with factors, as seen in equation (16).

$$r_{i,t} = \beta_{i,t-1}^{'} f_t + \epsilon_{i,t} \tag{16}$$

Therefore, the CAE enforces a *no-arbitrage constraint*, i.e., $\alpha = 0$ in equation (2); all returns are assumed to be compensation for risk exposures.

In addition to the CAE, we also implement the *arbitrage conditional autoencoder* (ACAE), which is our own extension of the CAE. The ACAE structure is identical to the CAE's, with one key addition: it adds an intercept to the model. This breaks the assumption that stock excess returns only stem from the compensation for its risk exposures, as it lets $\alpha \neq 0$ in equation (2). Thus, the ACAE also learns to identify mispricings, based on the inputted stock characteristics. The intercept is implemented by letting the beta network learn one more parameter, the intercept, and concatenating the factors with a constant scalar, 1, which gets multiplied with the learnt intercept. Therefore, for the ACAE, equations (11) (13) (15) are replaced by equations (17) (18) (19).

$$\underbrace{\beta_{i,t-1}}_{(K+1)\times 1} = \underbrace{b^{(L_\beta)} + \mathbf{W}^{(L_\beta)}}_{\text{these also expand}} z_{i,t-1}^{(L_\beta)} \tag{17}$$

$$\underbrace{f_t}_{(K+1)\times 1} = \begin{bmatrix} \tilde{b}^{(1)} + \tilde{\mathbf{W}}^{(1)} r_t^{(0)} \\ 1 \end{bmatrix} \tag{18}$$

$$\underbrace{\hat{r}_t}_{N\times 1} = \underbrace{\boldsymbol{\beta}_{t-1}}_{N\times(K+1)} \underbrace{f_t}_{(K+1)\times 1} \tag{19}$$

If we then break out the intercept from the matrix multiplication in equation (19), we get that

$$\underbrace{\hat{r}_t}_{N\times 1} = \underbrace{\boldsymbol{\beta}_{t-1}}_{N\times K} \underbrace{f_t}_{K\times 1} + \underbrace{\alpha_t}_{N\times 1},$$

where α_t is the intercept generated by the beta network.

Inspired by Gu et al. (2021), we use the Adam algorithm to train the networks (for 200 epochs, with a batch size of 32), combined with the following regularization techniques:

- Weight decay (using the L1-norm and $\lambda = 0.0001$), explained in section II.B.
- Batch normalization, i.e., normalizing the output from the hidden layer after each batch.
- Early stopping. If the validation loss does not decrease for 5 epochs, we stop training.

Furthermore, we employ ensemble learning to decrease the importance of the random initialization of the weights, by training multiple networks in parallel (10 in the main experiment and 5 in the

robustness check and the Monte Carlo simulation) and using their average output as our prediction. All of our empirical analysis is implemented in Python[6] and Keras[7], using the Tensorflow backend[8].

We implement four different network setups:

1. CAE0, a CAE with 0 hidden layers in the beta network, which means that both the beta network and the factor network is linear. Thus, the whole network is theoretically similar to IPCA.

2. ACAE0, same as CAE0 but with an intercept.

3. CAE1, a CAE with one hidden layer of 32 nodes in the beta network, which allows for nonlinear factor loadings.

4. ACAE1, same as CAE1 but with an intercept.

We operate each network in two different modes. When evaluating the model's ability to capture the realized riskiness in individual stock returns, see section III.D.1, we use the model as described above, inputting the returns in the asset returns input layer. This is similar to how we train the model. However, when we evaluate the model's ability to explain *expected* returns, i.e., when we try to predict future returns, see section III.D.2, we use another approach, as we in that case cannot input the returns (since they are not available). Instead, we bypass the asset return input layer and directly input the latent factors into the factor output layer, which allows us to predict future, still unknown, returns. The value used for these latent factors is the mean of the estimated latent factors up to time $t-1$.

C. Benchmark models

To compare our models' performances, we select the Fama-French factor model and the IPCA equivalent model as our benchmark models.

C.1. Fama-French factor models

In total, we choose six observable factors:

- *market*, *smb*, *hml*, *rmw*, *cma* from Fama and French (2015),

[6] *https://www.python.org/*
[7] *https://keras.io/*
[8] *https://www.tensorflow.org/*

- and the momentum factor *mom*

as predictors in the Fama-French factor model, see Table VII in the appendix.

$$\begin{aligned} E(R_{it}) - R_{ft} = \beta_{iM}[E(R_{Mt}) - R_{ft}] + \beta_{is}E(SMB_t) + h_{ih}E(HML_t) + \beta_{ir}E(RMW_t) \\ + \beta_{ic}E(CMA_t) + \beta_{im}E(MOM_t) \end{aligned} \tag{20}$$

In our comparison, the number of factors in the Fama-French models varies sequentially from 1 to 6, in the same order as in equation (20).

C.2. Instrumented Principal Component Analysis (IPCA)

IPCA is similar to the CAE model in its specification of factor loadings and latent factors. As aforementioned, one of the main differences between IPCA model and our model is that in its beta network, IPCA model assumes a linear relationship between factor loadings and stock characteristics, followed by the principal components analysis in its factor network to estimate factor loadings and associated latent factors simultaneously. Kelly et al. (2019) have shown that there is no closed-form solution to this model and it can only be solved numerically.

In GKX, they have shown that the IPCA model is equivalent to the conditional autoencoder model if we assume only one linear activation function in the beta network and the product of firm characteristics matrix and its transpose is constant for each period[9].

Although the constant matrix assumption is generally not satisfied, GKX have argued that the solutions to these models are still similar, so as their empirical performance. Therefore, we use the zero hidden layer beta network in our conditional autoencoder model to proxy for the IPCA equivalent result in our model comparison.

D. Performance evaluation

We present three out-of-sample measures to evaluate our model's performance in describing realized returns and predicting expected returns.

D.1. Total R^2

Total R^2 represents the proportion of variance of the realized individual stock returns explained by the conditional factor loadings and latent factors extracted from individual stock returns, as well

[9]The latter assumption can be denoted as: $Z_t^{'} Z_t = \Sigma$ for a constant matrix Σ, where Z_t is the stock characteristics matrix at time t. For detailed proof of this argument, please check the appendix in Gu et al. (2021).

as the intercept (if existent), in our model estimation. It evaluates our model's ability to capture the realized riskiness in individual stock returns.

$$R^2_{Total} = 1 - \frac{\sum_{(i,t)\in OOS}(r_{i,t} - \hat{\alpha}_{i,t} - \hat{\beta}'_{i,t-1}\hat{f}_t)^2}{\sum_{(i,t)\in OOS} r^2_{i,t}} \tag{21}$$

where OOS stands for the "out of sample" data set. The evaluation is computed solely on the test sample, ignoring any missing data points that were artificially filled.

D.2. Predictive R^2

Predictive R^2 represents the fraction of variance of the individual stock returns in the next period explained by the current conditional factor loadings and latent factors extracted from individual stock returns, as well as the intercept (if existent), in our model estimation. It evaluates our model's ability to explain expected returns.

$$R^2_{Pred} = 1 - \frac{\sum_{(i,t)\in OOS}(r_{i,t} - \hat{\alpha}_{i,t-1} - \hat{\beta}'_{i,t-1}\hat{\lambda}_{t-1})^2}{\sum_{(i,t)\in OOS} r^2_{i,t}} \tag{22}$$

where λ_{t-1} represents the risk premia associated with the factors. We follow GKX and assume it takes the average of estimated $\hat{f}$ up to month $t-1$. The evaluation is computed solely on the test sample, ignoring any missing data points that were artificially filled.

D.3. Sharpe ratio

To evaluate our model in terms of economic performance, we build up equal-weighted long-short portfolios based on predicted individual stock returns and compute their annualized Sharpe ratios as follows:

$$SR = \sqrt{12} \times \frac{r_p}{\sigma_p} \ , \tag{23}$$

where r_p denotes the average monthly excess return of a portfolio and σ_p stands for the standard deviation of the monthly portfolio returns.

In our portfolio creation process, we assume a non-friction market, meaning that there are no transaction costs or shorting constraints on stocks during the entire time horizon.

First, we implement a "naïve" strategy to construct a benchmark portfolio, which is simply the mean return of all stocks available for purchase. This "naïve" strategy does not depend on any model predictions, hence we use it as a benchmark.

Our model-based portfolios are constructed in the following fashion. At the beginning of month t, we rank stocks according to their *expected* excess returns, ignoring any missing data points that were artificially filled. Then we short the bottom 10% stocks and uses the money to purchase the top 10% stocks. In this way, we have constructed an equally-weighted long-short portfolio and held it during month t. At the beginning of month $t+1$, we then adjust the investment positions based on the same principle as aforementioned. Throughout the whole time horizon, we accumulate the gains and losses solely obtained from this strategy. Based on the accumulated wealth in each period, we calculate the annualized excess returns and standard deviations, and use equation (23) to compute Sharpe ratios for all of the model-based portfolios.

E. Monte Carlo simulation

To examine our model's performance, we create an artificial three-factor model in line with the conditional autoencoder model setup and do a Monte Carlo simulation. We follow GKX's method and assign the same values they use in their simulation for our model's parameters, so as to be comparable to their results. In this model, we assume there are $N = 200$ stocks with $P_c = 50$ observable characteristics per stock, $T = 180$ time periods, and $P_x = 50$ factor components to form latent factors.

First, we create stock characteristics dataset. For the jth stock characteristics, it follows an $AR(1)$ model:

$$c_{ij,t} = \rho_j c_{ij,t-1} + \epsilon_{ij,t} \ ,$$

where $\rho_j \sim \mathcal{U}[0.9, 1]$ measures the persistence in the characteristics time series and $\epsilon_{ij,t} \sim \mathcal{N}(0, 1)$ is white noise. We follow the same data preprocessing approach as in our real-world data analysis and implement cross-sectional rank normalization on the panel of stock characteristics:

$$c_{ij,t} = \frac{2}{N+1} rank(c_{ij,t}) - 1 \ .$$

The stock excess return for firm i at time t is determined in a plain vanilla multi-factor framework

$$r_{i,t} = \beta_{i,t-1}^{'} f_t + \varepsilon_{i,t} \ , \tag{24}$$

where $\beta_{i,t}$ is the factor loadings for stock i at period t, f_t denotes the risk factor vector and $\varepsilon_{i,t} \sim t_5(0, 0.1^2)$ stands for the idiosyncratic errors.

In this model, latent factors f_t are constructed as follows:

$$f_t = Wx_t + \eta_t \ ,$$

where $x_t \sim \mathcal{N}(0.03, 0.1^2 \times \mathbb{1}_{P_x})$ is the factor components at time t, $\eta_t \sim \mathcal{N}(0, 0.01^2 \times \mathbb{1}_3)$ denotes the residuals and W is a fixed weighting matrix which extracts three latent factors from x_t. For simplicity, we assume that the latent factors correspond to the first three factor components and set W as follows:

$$W = \begin{bmatrix} 1 & 0 & 0 & \dots & 0 \\ 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 1 & \dots & 0 \end{bmatrix} \ .$$

Factor loadings β are simulated as follows:

$$\beta_{i,t-1} = g^*(c_{i,t-1}; \theta) \ , \tag{25}$$

where $c_{ij,t}$ is an entry from the $N \times P_c \times T$ stock characteristics matrix C, and $g^*(c_{i,t}; \theta)$ is the nonlinear mapping from characteristics to factor loadings.

Since we assume that only the first three characteristics contribute to beta, $g^*(\cdot)$ includes three arguments.

$$g^*(c_{i,t}; \theta) = (c_{i1,t}^2, 2 \times (c_{i1,t} \times c_{i2,t}), 0.6 \times sign(c_{i3,t}))^{'} \ , \tag{26}$$

Thus, we simulate both returns and stock characteristics with a model without any intercept.

IV. Results and discussion

In this section, we present the main empirical findings of our models and discuss these results in detail.

A. Out-of-sample evaluation

A.1. Total R^2

Table I exhibits the out-of-sample total R^2 results for all the models. Although the number of stocks (1,000) used in our analysis is much smaller than GKX (nearly 30,000), overall, the total R^2 performance in our empirical results is as robust as theirs.

Table I
Out-of-sample total R^2(%) comparison

Models	Number of Factors					
	1	2	3	4	5	6
FF	13.0	14.1	15.3	14.9	13.9	13.2
CAE0	17.0	17.8	18.1	18.1	18.0	18.1
ACAE0	16.3	17.1	17.7	17.8	17.8	18.0
CAE1	17.0	17.0	17.1	17.1	17.0	17.1
ACAE1	17.2	17.2	17.1	17.1	17.3	17.2

Note: In this table, we report the out-of-sample total R^2 for Fama-French factor models (FF), IPCA equivalent models (CAE0/ACAE0), and 1-layer conditional autoencoder models (CAE1/ACAE1). The number of factors in all models varies from 1 to 6.

Our benchmark Fama-French factor model (FF) delivers the worst performance, compared to the other models, and its underperformance is in line with GKX's findings. One possible explanation for the underperformance is infrequent re-estimation of the model. GKX mention that the infrequent re-estimation in their model training process[10] contributes to FF's underperformance. Since the Fama-French factor model assumes a static beta formulation, without a timely re-estimation of model parameters it cannot render a good description of time-varying factor loadings. As our FF estimation uses the same design, this explanation also applies to our analysis. It should be noted that even though we see underperformance, FF still perform better in our experiments than in GKX's. This could be related to our sample size. Compared with datasets typically studied in the literature, GKX handle a much larger dataset[11] and they argue that FF is not suitable to describe datasets of this scale. As our analysis only covers 1,000 stocks, one could expect this effect to be smaller in our experiment.

In comparison, the IPCA equivalent models (CAE0/ACAE0) demonstrate quite robust results. As illustrated in the second row of Table I, for the no-intercept IPCA equivalent model (CAE0), increasing the number of factors generally improves its total R^2 performance, which indicates that the true risk structure may not be one-dimensional and that a multi-dimensional factor structure can better capture information contained in stock characteristics. The level of total R^2 plateaus around 18%, which provides evidence for a parsimonious factor structure instead of a "factor zoo".

[10] GKX implement the same 12-month model refitting frequency as ours. See Section III.A.3.

[11] The number of stocks in GKX dataset (nearly 30,000) far exceeds those typically studied in the literature (e.g., 12,000 in Kelly et al. (2019)).

However, the inclusion of an intercept in the model (ACAE0) does not add extra explanatory power in our sample.

For the nonlinear version of the conditional autoencoder model (CAE1/ACAE1), the out-of-sample total R^2 performance is rather stable (around 17%) for a varying number of factors. It should be noted that we see two differences compared with the IPCA equivalent models' performance. First, higher dimensional factor structures do not generate extra power in explaining contemporaneous stock returns. For example, the total R^2 for the 5-factor CAE5 model (17.1%) is the same as the 1-factor model (17.0%) in the one-factor CAE1 model. This result is not compatible with GKX's or the IPCA equivalent model's findings, both of which see a positive relationship between the number of factors and the corresponding total R^2 value. The other difference is that the introduction of an intercept into the overarching model in equation (16) improves our model's performance in explaining individual stock excess returns. As is shown in the last row of Table I, the value of total R^2 in ACAE1 is always equal or greater than its counterpart in CAE1 for each number of factors, suggesting the existence of compensation for mispricings.

A.2. Predictive R^2

In Table II, we report the out-of-sample predictive R^2 results for all the models. Compared to the Fama-French factor models, the predictive performance of our conditional autoencoder models still prevail.

The linear conditional autoencoder models (CAE0/ACAE0) exhibit robust performance in predicting expected returns. However, the introduction of an intercept exerts different influences on model's predictability depending on the number of factors, which coincides with findings in Kelly et al. (2019). Specifically, it slightly raises the predictive R^2 for lower dimensional factor models (1-2 factor models), but decreases it for models with a number of factors greater than 3. This results may suggest that with the increase of factor dimensions, the risk exposures better align with the information contained in stock characteristics, while the compensation from mispricings is gradually crowded out.

The results demonstrate a lower predictive R^2 level for the no-intercept nonlinear model (CAE1) compared to its linear counterpart (CAE0). However, this inferior performance quickly disappears once we include an intercept into the model (ACAE1), which generates the best performance for every number of factors. The robust out-of-sample predictability of ACAE1 models implies that the stock characteristics not only proxy for compensation of risk exposures, but also contains

information about the existence of mispricings.

Table II
Out-of-sample predictive R^2(%) comparison

Models	Number of Factors					
	1	2	3	4	5	6
FF	0.52	0.47	0.49	0.42	0.43	0.41
CAE0	0.69	0.68	0.77	0.78	0.78	0.76
ACAE0	0.72	0.72	0.72	0.75	0.70	0.70
CAE1	0.69	0.67	0.64	0.66	0.65	0.65
ACAE1	0.76	0.80	0.80	0.83	0.79	0.84

Note: In this table, we report the out-of-sample predictive R^2 for Fama-French factor models (FF), IPCA equivalent models (CAE0/ACAE0), and 1-layer conditional autoencoder models (CAE1/ACAE1). The number of factors in all models varies from 1 to 6.

A.3. Portfolio performance

We examine the economic performance of our models in terms of equal-weighted long-short portfolios' annualized Sharpe ratios and present them in Table III.

Table III
Long-short portfolio Sharpe ratio comparison

Portfolios	Number of Factors					
	1	2	3	4	5	6
Naïve	0.75					
FF	0.20	0.05	<0	<0	<0	<0
CAE0	0.43	0.36	0.99	1.12	1.12	1.16
ACAE0	1.33	1.31	1.33	1.43	1.36	1.34
CAE1	0.42	0.40	0.33	0.33	0.29	0.25
ACAE1	1.27	1.28	1.29	1.39	1.26	1.26

Note: In this table, we report Sharpe ratios for equal-weight long-short portfolios based on the "naïve" strategy, Fama-French factor models (FF), IPCA equivalent models (CAE0/ACAE0), and 1-layer conditional autoencoder models (CAE1/ACAE1). The number of factors in all models varies from 1 to 6.

As is shown in Table III, for the linear factor loading model without intercept (CAE0), we witness steadily rising Sharpe ratios as the number of factors increases. However, this positive

correlation between the number of factors and Sharpe ratios does not hold for the corresponding model with an intercept (ACAE0), in which the Sharpe ratios remain rather stable for portfolios with different number of factors.

Compared to the IPCA equivalent models, the nonlinear conditional autoencoder models (CAE1/ACAE1) do not exhibit better performance with regard to portfolio Sharpe ratios. In particular, portfolio performance based on CAE1 models lags "naïve" strategy by a large margin (as high as 0.5 for the 6-factor model). The performance of the ACAE1 model is more stable and resembles that of the ACAE0 model but with slightly lower Sharpe ratio across the board. Overall, for both IPCA equivalent and conditional autoencoder models, the intercept based versions achieves far better portfolio performance and scores higher Sharpe ratios compared to their no-intercept counterparts.

B. Robustness check

To further explore the robustness of the intercept based models' overperformance, we select a different sample from a larger slice of GKX data. Specifically, we choose top 1,000 stocks based on their frequencies during a 51-year period, which begins in January 1970 and ends in December 2020. The data preprocessing steps are the same as our empirical studies and the number of observations in our robustness check exceeds 500,000.

Beyond selecting a different dataset, we explicitly split the sample stocks into two equal-sized non-overlapping groups based on their PERMNO codes[12]. One group is used for training our models and the other one is used solely for evaluation. In our robustness check, we retrain the one-layer conditional autoencoder models (CAE1/ACAE1) and evaluate their out-of-sample (both in terms of time period and PERMNO code) performance.

The out-of-sample R^2 and Sharpe ratios for equal-weighted long-short portfolio based on the model predictions are reported in Table IV. Despite different stocks being used for training and testing, our models still produce robust out-of-sample results. The total and predictive R^2 and Sharpe ratios of model-based portfolios are as good as those in our main experiment. Furthermore, the intercept based model (ACAE1) still dominates the no-intercept model (CAE1) in terms of all three metrics, further indicating the existence of mispricing information within stock characteristics.

[12]PERMNO is the unique permanent issue identifier used in the Center for Research in Securities Prices (CRSP) database.

Table IV
Robustness check

Models	Out-of-sample Performance		Sharpe ratio	
	Total R^2(%)	Pred R^2(%)	Model-based	Naïve
CAE1	17.1	0.59	0.54	0.71
ACAE1	17.2	0.73	1.42	0.71

Note: In this table, we report the out-of-sample R^2 and Sharpe ratios based on 1-layer conditional autoencoder models with (ACAE1) and without (CAE1) intercept using a 1,000-stock sample, where 500 stocks are used solely for evaluation. The number of factors in all models is 6.

C. Simulation

Since the factor loadings in the simulated data are specified as nonlinear transformations from stock characteristics (equation (26)), it is not surprising to observe a strong performance of nonlinear conditional autoencoder models (CAE1/ACAE1) in Table V. And as expected, the IPCA equivalent models (CAE0/ACAE0) do not achieve a good out-of-sample R^2 in the simulations due to its inability to capture nonlinearities in the factor loadings network. Among these models, CAE1 produces the best results in all three metrics (total R^2, predictive R^2 and Sharpe ratio), which beats the corresponding intercept based model, ACAE1. This result is expected because, in the data generating process, we explicitly exclude the intercept in simulating stock excess returns (equation (24)) and make stock characteristics solely proxy for compensation for factor loadings (equation (26)). This simulation design guarantees the outperformance of no-intercept models over intercept based model.

These simulation results confirm two qualities of our conditional autoencoder model. First, if there exist nonlinearties within the factor loadings network, our model can capture them and generate better out-of-sample performance than the IPCA equivalent linear models. Second, the no-intercept models dominates intercept models if there does not exist compensations for mispricings in stock excess returns.

Table V
Simulation Result

Models	Out-of-sample Performance		Sharpe ratio	
	Total R^2(%)	Pred R^2(%)	Model-based	Naïve
CAE0	4.8	0.9	0.94	0.98
ACAE0	<0	0.7	0.99	0.98
CAE1	12.3	2.2	1.41	0.98
ACAE1	9.1	1.3	1.41	0.98

Note: In this table, we report the out-of-sample R^2 and Sharpe ratios based on IPCA equivalent models (CAE0/ACAE0) and 1-layer conditional autoencoder models (CAE1/ACAE1) in simulation dataset. The number of factors in all models is specified to be 3.

D. Discussion about the empirical findings

Two themes we find in our empirical study are that

1. introducing non-linearity in the factor loading networks (as in CAE1/ACAE1) does not necessarily improve models' out-of-sample performance

2. and that the inclusion of an intercept improves our models' performance, especially in terms of predictive R^2 and portfolio Sharpe ratios.

D.1. Underperformance of the nonlinear models

While the intercept based nonlinear conditional autoencoder (ACAE1) is the best performing model in terms of predictive R^2, the nonlinear methods did not outperform the linear methods across the board. The dominance of IPCA equivalent models (CAE0/ACAE0) over shallow one-layer conditional autoencoder models (CAE1/ACAE1) can also be found in GKX's findings, in which they study a much larger sample dataset than ours. In the particular case of out-of-sample total R^2, IPCA with 6 factors is the best overall model in GKX, and for predictive R^2 and Sharpe ratio, GKX's nonlinear conditional autoencoder model with shallow structure (e.g., one hidden layer in beta network) also tends to perform less effectively.

There are several possible explanations for the underperformance of CAE1/ACAE1 in our case. On one hand, due to computational constraints, we only implement a shallow one-hidden-layer factor loadings network. Chances are that the nonlinear interactions between stock characteristics and factor loadings are not well reflected by this simple structure, and higher degrees of complexity

(e.g., more hidden layers in the beta network) need to be introduced in the model. This argument is supported by empirical results in GKX, in which their 2-layer/3-layer models achieve a better out-of-sample performance than their 1-layer model. On the other hand, the risk exposures information contained in stock characteristics in our sample dataset could potentially better depicted by linear transformations. In this case, nonlinear models would not be suitable, as they are less parsimonious than the IPCA equivalent linear models and add no extra benefit. Our simulation results provide empirical support for this argument. As is shown in Table V, we have created a simulated dataset with pre-specified nonlinear beta network, and our nonlinear conditional autoencoder model (CAE1) succeeds in capturing the nonlinearities and outperforming the IPCA equivalent linear model (CAE0). Therefore, the poor performance of nonlinear models in our empirical findings suggest a linear relationship between stock characteristics and factor loadings in our real-world dataset.

The underperformance of more complex models could also be of a more technical nature. First, the results could be explained by the bias-variance tradeoff (see section II.B); the nonlinear model might very well have too high variance, which would impact the out-of-sample performance. Second, more complex models face a higher risk of overfitting (see section II.B), which would also impact the out-of-sample performance. Finally, training machine learning models is not trivial and training more complex models is inherently harder, as the optimization contains more parameters (see section II.B), which could impact the performance of the deeper nonlinear models.

D.2. Arbitrage versus no-arbitrage

GKX explicitly specify their conditional autoencoder models without an intercept. In doing so, they assume a *no-arbitrage* constraint in their model, i.e., the predictive power of stock characteristics results solely from them being a proxy for factor risk exposures and not for mispricings. To support this argument, GKX compare their two-layer conditional autoencoder model's out-of-sample predictive R^2 result (0.58%) with that (0.40%) of deep neural network model from Gu et al. (2020). Since the deep neural network model in Gu et al. (2020) does not distinguish compensations from risk exposures and those from mispricings, GKX claim that the similarity between these two results implies that the predictability most probably comes from risk exposures and that their no-arbitrage assumption holds.

Our empirical findings for the no-intercept model (CAE1) and the intercept based model (ACAE1), our best performing model in terms of predictive R^2, clearly reject the feasibility of

the no-arbitrage assumption for our dataset. As shown in Table I, II, and III, in terms of total R^2, predictive R^2 and Sharpe ratios results, ACAE1 dominates CAE1 in every single model for real-world sample data. Furthermore, in our robustness check, we train and test the nonlinear models on non-overlapping subsets and the intercept based model still prevails in terms of out-of-sample performance. Finally, the outcome of the Monte Carlo simulation provides further evidence for the existence of arbitrage in real-world data. In the stock return simulation (equation (24)), we explicitly exclude intercept in the data generation model and thus the ACAE model does not outperform its no-arbitrage counterpart. This result suggests the existence of arbitrage within the real-world data used in the main experiment, as we would otherwise not witness a dominance of intercept based models in out-of-sample performance.

In conclusion, the inclusion of an intercept improves the performance of our nonlinear conditional autoencoder model both in explaining contemporaneous stock returns and predicting future returns. This indicates that stock characteristics not only proxy for compensation of risk exposures, but also provide information of mispricings.

However, the results are not as clear for the linear models. But even there, we see a higher Sharpe ratio across the board for the intercept based model.

V. Conclusion

In this thesis, we have applied a conditional autoencoder model to describe and capture the interactions between stock characteristics and factor loadings, while simultaneously extracting latent factors from stock returns. Compared with traditional factor models, our machine learning model has exhibited superior out-of-sample performance. Furthermore, we have shown that adding an intercept to the model improves performance. This goes against Gu et al. (2021)'s argument that stock characteristics proxy for factor risk premia and not mispricing. Thus, we believe that stock characteristics proxy not only for factor risk premia, but also compensation for mispricing. In our robustness check, we argue for the reliability of our results by using different stocks for training and testing. Furthermore, we have shown, via a Monte Carlo simulation, that adding an intercept to the machine learning model does not increase performance if stock returns are completely determined by risk exposures, further strengthening our argument about the presence of mispricing in real-world data.

With more available computational capacity, one can implement our study in a greater scale

with a larger sample size, deeper network structures and better hyperparameter search, which could potentially improve performance and the reliability of the results.

VI. References

Bansal, R. and Yaron, A. (2004). Risks for the long run: A potential resolution of asset pricing puzzles. Journal of Finance, 59(4):1481–1509.

Bellman, R. E. (1961). Adaptive control processes : a guided tour. Princeton Univ. Press, Princeton.

Bishop, C. M. (2006). Pattern Recognition and Machine Learning. Springer.

Black, F. (1972). Capital market equilibrium with restricted borrowing. Journal of Business, 45(3):444–455.

Campbell, J. Y. and Cochrane, J. H. (1999). By force of habit: A consumption-based explanation of aggregate stock market behavior. Journal of Political Economy, 107(2):205–251.

Carhart, M. M. (1997). On persistence in mutual fund performance. Journal of Finance, 52(1):57–82.

Chamberlain, G. and Rothschild, M. (1983). Arbitrage, factor structure, and mean-variance analysis on large asset markets. Econometrica, 51(5):1281–1304.

Chu, Y., Hirshleifer, D., and Ma, L. (2020). The causal effect of limits to arbitrage on asset pricing anomalies. The Journal of Finance, 75(5):2631–2672.

Cochrane, J. H. (2009). Asset pricing: Revised edition. Princeton university press.

Fama, E. F. and French, K. R. (1993). Common risk factors in the returns on stocks and bonds. Journal of Financial Economics, 33:3–56.

Fama, E. F. and French, K. R. (2015). A five-factor asset pricing model. Journal of Financial Economics, 116(1):1–22.

Fama, E. F. and MacBeth, J. D. (1973). Risk, return, and equilibrium: Empirical tests. Journal of Political Economy, 81(3):607–636.

Friedman, J., Hastie, T., Tibshirani, R., et al. (2001). The elements of statistical learning, volume 1. Springer series in statistics New York.

Gardner, M. W. and Dorling, S. (1998). Artificial neural networks (the multilayer perceptron)—a review of applications in the atmospheric sciences. Atmospheric environment, 32(14-15):2627–2636.

Giglio, S. and Xiu, D. (2021). Asset pricing with omitted factors. Journal of Political Economy, 129(7):000–000.

Gu, S., Kelly, B., and Xiu, D. (2020). Empirical asset pricing via machine learning. Review of Financial Studies, 33(5):2223–2273.

Gu, S., Kelly, B., and Xiu, D. (2021). Autoencoder asset pricing models. Journal of Econometrics, 222(1):429–450.

Israel, R., Kelly, B. T., and Moskowitz, T. J. (2020). Can machines "learn" finance? Available at SSRN 3624052.

Jegadeesh, N., Noh, J., Pukthuanthong, K., Roll, R., and Wang, J. (2019). Empirical tests of asset pricing models with individual assets: Resolving the errors-in-variables bias in risk premium estimation. Journal of Financial Economics, 133(2):273–298.

Kelly, B. T., Pruitt, S., and Su, Y. (2019). Characteristics are covariances: A unified model of risk and return. Journal of Financial Economics, 134(3):501–524.

Kelly, B. T., Pruitt, S., and Su, Y. (2020). Instrumented principal component analysis. Available at SSRN 2983919.

Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980.

Le Cun, Y., Jackel, L. D., Boser, B., Denker, J. S., Graf, H. P., Guyon, I., Henderson, D., Howard, R. E., and Hubbard, W. (1989). Handwritten digit recognition: Applications of neural network chips and automatic learning. IEEE Communications Magazine, 27(11):41–46.

Lee, J. A. and Verleysen, M. (2007). Nonlinear dimensionality reduction. Springer Science & Business Media.

Lintner, J. (1965). Security prices, risk, and maximal gains from diversification. Journal of Finance, 20(4):587–615.

Maki, A. (2020). Lecture 3: Challenges in machine learning, dd2421. KTH Royal Institute of Technology.

Mitchell, T. M. et al. (1997). Machine learning. McGraw-hill New York.

Mohri, M., Rostamizadeh, A., and Talwalkar, A. (2018). Foundations of machine learning. MIT press.

Murtagh, F. (1991). Multilayer perceptrons for classification and regression. Neurocomputing, 2(5):183–197.

Nair, V. and Hinton, G. E. (2010). Rectified linear units improve restricted boltzmann machines. In Icml.

Novy-Marx, R. (2013). The other side of value: The gross profitability premium. Journal of Financial Economics, 108(1):1–28.

Pukthuanthong, K., Roll, R., and Subrahmanyam, A. (2019). A protocol for factor identification. Review of Financial Studies, 32(4):1573–1607.

Roll, R. (1977). A critique of the asset pricing theory's tests part i: On past and potential testability of the theory. Journal of Financial Economics, 4(2):129–176.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1985). Learning internal representations by error propagation. Technical report, California Univ San Diego La Jolla Inst for Cognitive Science.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning representations by back-propagating errors. Nature, 323(6088):533–536.

Sharpe, W. F. (1964). Capital asset prices: A theory of market equilibrium under conditions of risk. Journal of Finance, 19(3):425–442.

Wegman, E. J. (1990). Hyperdimensional data analysis using parallel coordinates. Journal of the American Statistical Association, 85(411):664–675.

Appendix A.

Here we present the detailed summary of the input variables used in our conditional autoencoder model.

Table VI
Summary of Stock Characteristics

No.	Acronym	Firm characteristics	Frequency
1	absacc	Absolute accruals	Annual
2	acc	Working capital accruals	Annual
3	aeavol	Abnormal earnings announcement volume	Quarterly
4	age	# years since first Compustat coverage	Annual
5	agr	Asset growth	Annual
6	baspread	Bid-ask spread	Monthly
7	beta	Beta	Monthly
8	betasq	Beta squared	Monthly
9	bm	Book-to-market	Annual
10	bm_ia	Industry-adjusted book to market	Annual
11	cash	Cash holdings	Quarterly
12	cashdebt	Cash flow to debt	Annual
13	cashpr	Cash productivity	Annual
14	cfp	Cash flow to price ratio	Annual
15	cfp_ia	Industry-adjusted cash flow to price ratio	Annual
16	chatoia	Industry-adjusted change in asset turnover	Annual
17	chcsho	Change in shares outstanding	Annual
18	chempia	Industry-adjusted change in employees	Annual
19	chinv	Change in inventory	Annual
20	chmom	Change in 6-month momentum	Monthly
21	chpmia	Industry-adjusted change in profit margin	Annual
22	chtx	Change in tax expense	Quarterly
23	cinvest	Corporate investment	Quarterly
24	convind	Convertible debt indicator	Annual
25	currat	Current ratio	Annual
26	depr	Depreciation / PP&E	Annual
27	divi	Dividend initiation	Annual
28	divo	Dividend omission	Annual
29	dolvol	Dollar trading volume	Monthly
30	dy	Dividend to price	Annual
31	ear	Earnings announcement return	Quarterly
32	egr	Growth in common shareholder equity	Annual

Table VI
Summary of Stock Characteristics (continued)

No.	Acronym	Firm characteristics	Frequency
33	ep	Earnings to price	Annual
34	gma	Gross profitability	Annual
35	grcapx	Growth in capital expenditures	Annual
36	grltnoa	Growth in long term net operating assets	Annual
37	herf	Industry sales concentration	Annual
38	hire	Employee growth rate	Annual
39	idiovol	Idiosyncratic return volatility	Monthly
40	ill	Illiquidity	Monthly
41	indmom	Industry momentum	Monthly
42	invest	Capital expenditures and inventory	Annual
43	lev	Leverage	Annual
44	lgr	Growth in long-term debt	Annual
45	maxret	Maximum daily return	Monthly
46	mom12m	12-month momentum	Monthly
47	mom1m	1-month momentum	Monthly
48	mom36m	36-month momentum	Monthly
49	mom6m	6-month momentum	Monthly
50	ms	Financial statement score	Quarterly
51	mvel1	Size	Monthly
52	mve_ia	Industry-adjusted size	Annual
53	nincr	Number of earnings increases	Quarterly
54	operprof	Operating profitability	Annual
55	orgcap	Organizational capital	Annual
56	pchcapx_ia	Industry adjusted % change in capital expenditures	Annual
57	pchcurrat	% change in current ratio	Annual
58	pchdepr	% change in depreciation	Annual
59	pchgm_pchsale	% change in gross margin - % change in sales	Annual
60	pchquick	% change in quick ratio	Annual
61	pchsale_pchinvt	% change in sales - % change in inventory	Annual
62	pchsale_pchrect	% change in sales - % change in A/R	Annual
63	pchsale_pchxsga	% change in sales - % change in SG&A	Annual
64	pchsaleinv	% change in sales-to-inventory	Annual
65	pctacc	Percent accruals	Annual
66	pricedelay	Price delay	Monthly
67	ps	Financial statements score	Annual

Table VI
Summary of Stock Characteristics (continued)

No.	Acronym	Firm characteristics	Frequency
68	quick	Quick ratio	Annual
69	rd	R&D increase	Annual
70	rd_mve	R&D to market capitalization	Annual
71	rd_sale	R&D to sales	Annual
72	realestate	Real estate holdings	Annual
73	retvol	Return volatility	Monthly
74	roaq	Return on assets	Quarterly
75	roavol	Earnings volatility	Quarterly
76	roeq	Return on equity	Quarterly
77	roic	Return on invested capital	Annual
78	rsup	Revenue surprise	Quarterly
79	salecash	Sales to cash	Annual
80	saleinv	Sales to inventory	Annual
81	salerec	Sales to receivables	Annual
82	secured	Secured debt	Annual
83	securedind	Secured debt indicator	Annual
84	sgr	Sales growth	Annual
85	sin	Sin stocks	Annual
86	sp	Sales to price	Annual
87	std_dolvol	Volatility of liquidity (dollar trading volume)	Monthly
88	std_turn	Volatility of liquidity (share turnover)	Monthly
89	stdacc	Accrual volatility	Quarterly
90	stdcf	Cash flow volatility	Quarterly
91	tang	Debt capacity/firm tangibility	Annual
92	tb	Tax income to book income	Annual
93	turn	Share turnover	Monthly
94	zerotrade	Zero trading days	Monthly

Note: The data is collected in Gu et al. (2020) and it is updated to 2020. See: *https://dachxiu.chicagobooth.edu/#research*

Here we present a summary of factors used in Fama-French factor model.

Table VII
Summary of Fama-French Factors

No.	Acronym	Fama-French Factors	Frequency
1	mkt-rf	Excess return on the market	Monthly
2	smb	Size	Monthly
3	hml	Value	Monthly
4	rmw	Operating profitability	Monthly
5	cma	Investment	Monthly
6	mom	Momentum	Monthly

Note: The data is collected in Professor Kenneth R. French's website. See: *http://mba.tuck.dartmouth.edu/pages/faculty/ken.french/data_library.html*