Autoencoder Asset Pricing Models

Shihao Gu
Booth School of Business
University of Chicago

Bryan Kelly
Yale University, AQR Capital Management, and NBER

Dacheng Xiu
Booth School of Business
University of Chicago

February 15, 2019

Abstract

We propose a new latent factor conditional asset pricing model. Like Kelly, Pruitt, and Su (KPS, 2019), our model allows for latent factors and factor exposures that depend on covariates such as asset characteristics. But, unlike the linearity assumption of KPS, we model factor exposures as a flexible nonlinear function of covariates. Our model retrofits the workhorse unsupervised dimension reduction device from the machine learning literature—autoencoder neural networks—to incorporate information from covariates along with returns themselves. This delivers estimates of *nonlinear conditional* exposures and the associated latent factors. Furthermore, our machine learning framework imposes the economic restriction of no-arbitrage. Our autoencoder asset pricing model delivers out-of-sample pricing errors that are far smaller (and generally insignificant) compared to other leading factor models.

Key words: Stock returns, conditional asset pricing model, nonlinear factor model, machine learning, autoencoder, neural networks, big data

1 Introduction

A recent asset pricing literature has emerged challenging the "anomaly" view of characteristic-based asset return prediction. The anomaly view suggests that certain asset attributes have the power to forecast returns above and beyond the expected return variation warranted as compensation for aggregate risk exposures. Kelly, Pruitt, and Su (KPS, 2019) provide evidence that these so-called anomaly asset characteristics in fact proxy for unobservable and time-varying exposures to risk factors, and shows that characteristics contain little (if any) anomalous return predictability once their explanatory power for factor exposures has been accounted for. In other words, characteristics appear to predict returns because they help pinpoint compensated aggregate risk exposures.

The asset pricing model proposed by KPS assumes that individual returns $r_{i,t+1}$ possess a K-factor structure:

$$r_{i,t} = \beta(z_{i,t-1})' f_t + u_{i,t}. \tag{1}$$

The factors f_t are treated as latent, and the $K \times 1$ conditional factor exposure $\beta(z_{i,t-1})$ is a function of an $L \times 1$ vector of asset characteristics $z_{i,t-1}$, where L is potentially high dimensional and strictly greater than K. KPS make the simplifying assumption that the map from L characteristics to K betas is linear:

$$\beta(z_{i,t-1})' = z'_{i,t-1}\Gamma, \tag{2}$$

which leads to an especially tractable estimation strategy for both the beta function and the latent factors.

There are no obvious theoretical or intuitive justifications to suggest that the convenient linearity assumption is satisfied in the data. To the contrary, there are many reasons to expect that this assumption is violated. Essentially all leading theoretical asset pricing models predict nonlinearities in return dynamics as a function of state variables; Campbell and Cochrane (1999), Bansal and Yaron (2004), and He and Krishnamurthy (2013) are prominent examples. Theory also predicts complex dynamics in factor risk exposures, as shown for example in the general equilibrium model of Santos and Veronesi (2004).

We generalize the factor model in (1) using models from the autoencoder family. Autoencoders are workhorse dimension reduction models in the field of machine learning. They can be thought of as a nonlinear, neural network counterpart to principal components analysis (PCA). Both autoencoders and PCA are unsupervised methods—they attempt to model the full panel of asset returns using only the returns themselves as inputs. The statistical content of both methods is a bottleneck that enforces a parsimonious representation of the return data set. The PCA bottleneck uses a linear mapping from N individual returns into $K \ll N$ factors, while autoencoders allow for a nonlinear mapping through neural networks.

Neither method, in their standard form, uses information in covariates to guide dimension reduction. KPS propose "instrumented" PCA (IPCA), which allows the information in covariates to guide the reduction via equation (2) but remains reliant on the linear model formulation.

In this paper, we introduce a new conditional autoencoder which, like IPCA, allows covariates

to help guide dimension reduction. As in its standard formulation, our autoencoder uses a neural network-based compression of returns into a low-dimensional set of factors that is not subject to the restrictive linearity assumption of PCA. At the same time, we allow stock characteristic covariates to have nonlinear and interactive effects on factor exposures. Ultimately, ours is a nonlinear conditional asset pricing model, where the nonlinearities manifest both through a flexible neural network mapping of covariates into betas alongside the compression of returns into factors.

Our empirical analysis of a 60-year history of individual equity returns in the US shows that our autoencoder model dominates observable factor models in the tradition of Fama and French (1993) that use static factor betas, as well as the more sophisticated models such as the linear conditional beta specification of KPS. We follow KPS and compare models based on two statistical criteria. The first is a model's "total R^2," which describes the fraction of variation in the full panel of returns explained by contemporaneous factor realizations. This measures the model's ability to describe the riskiness, or joint return covariation, in the set of assets. The second criterion is a model's predictive R^2, which describes the fraction of return variation described by lagged conditional betas. This measures a model's ability to describe differences in risk compensation across assets.

We conduct all of our comparisons on a purely out-of-sample basis. Focusing, for example, on three-factor specifications, we find that monthly total R^2 from our preferred autoencoder, from IPCA, and from the Fama-French three-factor model are 12.6%, 12.3%, and 3.4%, respectively, while the predictive R^2 is 0.50%, and 0.23% for the autoencoder and IPCA, and is negative for the Fama-French model.

We also compare the relative performance of models in economic terms. In particular, we form long-short decile spread portfolios directly sorted on out-of-sample stock return predictions from each model. Portfolios based on the three-factor autoencoder, IPCA, and Fama-French models earn annualized Sharpe Ratios of 2.16, 1.34, and -0.40, respectively, when portfolios are equal weighted. For value weighted portfolios, the respective Sharpe ratios are 0.92, 0.41, and -0.69. In summary, our conditional autoencoder model outperforms its leading competitors by a wide margin.

We contribute to a burgeoning literature using high dimensional statistical methods, including machine learning techniques, to analyze the cross section of risk and return in financial markets. The leading example in this vein is Gu et al. (2018), who conduct a comparison of machine learning methods for predicting the panel of individual US stock returns. That paper outlines a new research agenda marrying machine learning with the study of asset risk premia.[1] Gu et al. (2018) focus on supervised prediction models but take no stand on the risk-return tradeoff. In other words, their approaches do not constitute asset pricing models. In this paper, we develop asset pricing models using unsupervised and semi-supervised learning methods that model the risk-return tradeoff explicitly. Autoencoders are critical tools in the machine learning suite that have enjoyed success in wide range of practical applications.[2] Our contribution links the methodology literature on autoencoders with the large finance literature on factor pricing models.

[1] Other related asset pricing papers include Feng et al. (2019), Freyberger et al. (2017), Kozak et al. (2017), Feng et al. (2017), Giglio and Xiu (2018), and Gagliardini et al. (2016), among others.

[2] See, for example, Gallinari et al. (1987), Bourlard and Kamp (1988), Hinton and Zemel (1994), and Goodfellow et al. (2016).

KPS, who focus specifically on conditional factor pricing models, is the closest predecessor to our analysis.[3] They unify the literature on linear latent factor APT models (starting from Ross, 1976) with the literature on characteristic-based "anomaly" return prediction. One of our contributions is to extend their work by allowing for a general nonlinear specification of the return factor structure. To do so, we augment the traditional autoencoder by embedding a neural network in the specification of conditional betas, which allows characteristics to determine risk exposures through flexible nonlinearities and interactions, generalizing the linear "instrumented" beta specification of KPS.[4]

Another related predecessor is Kozak et al. (2018), who propose an approach to factor analysis for asset pricing using principal components from "anomaly"-sorted portfolios (reviving an earlier literature on this approach exemplified by Chamberlain and Rothschild, 1983; Connor and Korajczyk, 1986). This approach, however, fails to explain the risk-return tradeoff when the test assets are individual stocks, and is thus not generally applicable (see KPS for a discussion). Our conditional autoencoder model does not suffer this shortcoming. It accurately describes the risk and expected return successfully for individual stocks as well as for anomaly or other stock portfolios. More broadly, we show that our conditional autoencoder formulation is a valid asset pricing model. It is equivalent to a nonparametric model for a stochastic discount factor, and imposes the economic restriction of no-arbitrage pricing. One illustration of the conditional autoencoder's empirical success as a pricing model is that it correctly prices so-called anomaly portfolios with small and insignificant pricing errors. In fact, the pricing errors in our model, which are measured on a purely out-of-sample basis, are a fraction of the magnitude of those from traditional Fama-French factor models.

The rest of the paper is organized as follows. In Section 2, we set up the model and present our methodology. Section 3 presents our empirical studies. Section 4 provides Monte Carlo simulations that demonstrate the performance of our procedures. Section 5 concludes. The appendix contains mathematical proofs and detailed algorithms.

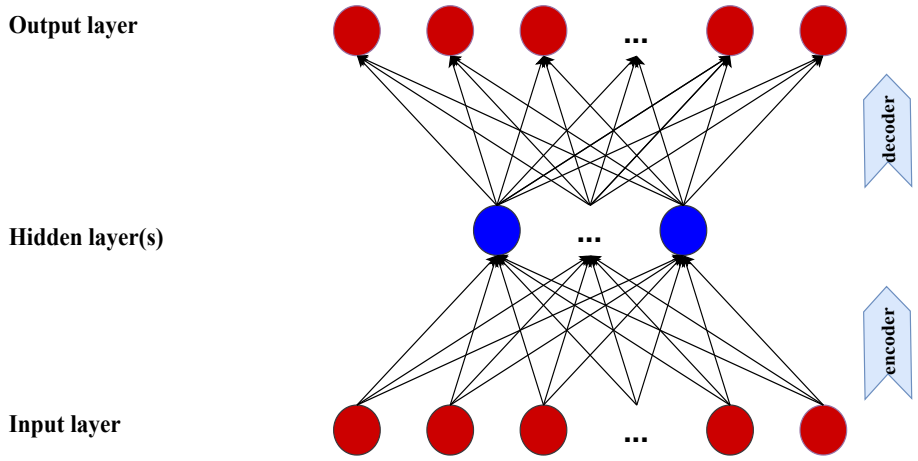
2 Methodology

In this section we first describe standard autoencoders, then we introduce our new conditional autoencoder that leverages covariates as conditioning information. We highlight parallels between autoencoder models and widely studied factor pricing models, and illustrate that linear factor models (and their associated PCA and IPCA estimators) are a special case of autoencoders. Finally, we describe our autoencoder estimation approach.

[3] A related contemporaneous paper that builds on the instrumented beta approach of KPS using kernel methods is Kozak (2019).

[4] Our approach to autoencoder learning for factor models is also related to work relying on kernel or sieve approximations, e.g., Connor et al. (2012) and Fan et al. (2016). Recent work also considers machine learning tools for factor model estimation, such as nuclear norm penalization (Bai and Ng, 2017; Moon and Weidner, 2018) and partial least squares (Kelly and Pruitt, 2015). Our autoencoder approach allows for flexible (and conditional) model structures, and uses stochastic gradient descent to manage the computational complexity in the resulting big data and high parameter setting. That said, a thorough theoretical analysis on the statistical properties of this alternative learning method is left for future research.

Figure 1: Standard Autoencoder Model



Note: This figure describes a standard autoencoder with one hidden layer. The output and input layers are identical, while the hidden layer is a low dimensional compression of inputs variables into latent factors, which can be expressed as weighted linear combinations of input variables.

2.1 Standard Autoencoder

An autoencoder is a special neural network in which the outputs attempt to approximate the input variables. The input variables pass through a small number of neurons in the hidden layer(s), forging a compressed representation of the input (encoding), which is then unpacked and mapped to the output layer (decoding). Because no other variables are used in this model besides the inputs, an autoencoder is an unsupervised learning device.

Neural network models (including autoencoders) with L hidden layers can be written using the following recursive formula. Let $K^{(l)}$ denote the number of neurons in each layer $l = 1, ..., L$. Define the output of neuron k in layer l as $r_k^{(l)}$, and the vector of all outputs for this layer as $r^{(l)} = (r_1^{(l)}, \ldots, r_{K^{(l)}}^{(l)})'$. In each hidden layer, inputs from the previous layer are transformed according to a nonlinear activation function $g(\cdot)$ before being passed to the next layer. To initialize the network, the input layer uses the cross section of returns as raw predictors, $r^{(0)} = r = (r_1, \ldots, r_N)'$. The recursive output formula for the neural network at each neuron in layer $l > 0$ is then

$$r_k^{(l)} = g\left(b^{(l-1)} + r^{(l-1)'} W^{(l-1)}\right), \tag{3}$$

with final output

$$G(r, b, W) = b^{(L-1)} + r^{(L-1)'} W^{(L-1)}, \tag{4}$$

where $W^{(l-1)}$ is a $K^{(l)} \times K^{(l-1)}$ matrix of weight parameters, and $b^{(l-1)}$ is a $K^{(l)} \times 1$ vector of so-called bias parameters. Throughout this paper we use as our nonlinear activation function the rectified linear unit (ReLU), $g(y) = \max(y, 0)$. The number of parameters in each hidden layer l is

$K^{(l)}(1+K^{(l-1)})$, plus another $1+K^{(L-1)}$ weights for the output layer. In summary, an autoencoder employs $G(r,b,W)$ to approximate r itself. Figure 1 illustrates the architecture of a simple linear autoencoder with a single hidden layer.

2.1.1 Static Linear Factor Models as a Special Case

The autoencoder is a dimension reduction device. Its objective is to learn a low dimensional representation of the inputs by passing them through a central hidden layer with many fewer neurons than the dimension of r. It thus shares the same spirit as PCA, but is more flexible in that it allows for nonlinear compression of the inputs. In this section we explore the connection between autoencoders and PCA.

In the finance literature, some of the most commonly studied models of asset returns assume a linear latent factor specification with static loadings:[5]

$$r_t = \beta f_t + u_t, \tag{5}$$

where r_t is a vector of returns in excess of the risk free rate, f_t is a $K \times 1$ vector of factor returns, u_t is a $N \times 1$ vector of idiosyncratic errors (uncorrelated with f_t), and β is an $N \times K$ matrix of factor loadings. This resembles that standard factor model setting whose econometric properties are studied in Bai and Ng (2002), Bai (2003), and Giglio and Xiu (2018), among others.

Stacking the time series vectors, the matrix form of the factor model is

$$R = \beta F + U.$$

Following Stock and Watson (2002) and Bai and Ng (2002), a factor model can be estimated with PCA on the covariance matrix of returns.[6] Equivalently, a singular value decomposition (SVD) of R yields estimates of factors and factor loadings directly.[7] That is,

$$R = P\Lambda Q + \widehat{U}, \tag{6}$$

where P and Q are, respectively, the $N \times K$ and $K \times T$ matrices of left and right singular vectors, and $\widehat{U}$ is a $N \times T$ matrix of residuals.

The machine learning literature has long recognized the close connection between autoencoders and PCA (e.g., Baldi and Hornik, 1989) and, by extension, between autoencoders and latent factor asset pricing models. When the autoencoder has one hidder layer and a linear activation function, it is equivalent to the PCA estimator for linear factor models described above.

[5] For example, Connor and Korajczyk (1986) and Kozak et al. (2017).

[6] A subtle consideration in estimating asset pricing models with PCA is choosing whether to impose the zero intercept no-arbitrage restriction. Imposing the restriction amounts to applying PCA to the uncentered second moment matrix of excess returns, rather than to the (centered) covariance matrix.

[7] This approach is applicable with any number of factors $K < N$. An extensive literature studies methods for choosing K, including Bai and Ng (2002), Hallin and Liška (2007), Amengual and Watson (2007), Alessi et al. (2010), Kapetanios (2010), Onatski (2010), Ahn and Horenstein (2013), and Aït-Sahalia and Xiu (2017). Throughout the paper, we will treat K as a tuning parameter, which in principle can be estimated via cross validation.

More specifically, we can write the one-layer, linear autoencoder with K neurons as:

$$r_t = b^{(1)} + W^{(1)}(b^{(0)} + W^{(0)} r_t) + u_t, \tag{7}$$

where $W^{(0)}$, $W^{(1)}$, $b^{(1)}$ and $b^{(0)}$ are $K \times N$, $N \times K$, $N \times 1$, and $K \times 1$ matrices of parameters, respectively. The model can be estimated by solving the following optimization problem:

$$\begin{aligned} &\min_{b,W} \sum_{t=1}^{T} \left\| r_t - \left(b^{(1)} + W^{(1)}(b^{(0)} + W^{(0)} r_t) \right) \right\|^2 \\ =&\min_{b,W} \left\| R - \left(b^{(1)}\iota' + W^{(1)}(b^{(0)}\iota' + W^{(0)} R) \right) \right\|_{\mathrm{F}}^2, \end{aligned} \tag{8}$$

where the F subscript denotes the Frobenius norm. Next, we establish the link between this simple autoencoder and the PCA estimator.

Proposition 1. *The optimal solution to (8) is given by,*

$$\widehat{W}^{(1)} = PA, \quad \widehat{W}^{(0)} = (\widehat{W}^{(1)\prime}\widehat{W}^{(1)})^{-1}\widehat{W}^{(1)\prime}, \quad \widehat{b}^{(1)} = \bar{r} - \widehat{W}^{(1)}\widehat{b}^{(0)} - \widehat{W}^{(1)}\widehat{W}^{(0)}\bar{r}, \quad \widehat{b}^{(0)} = a,$$

where A is any $K \times K$ non-singular matrix, a is a constant scalar, and P is from equation (6).

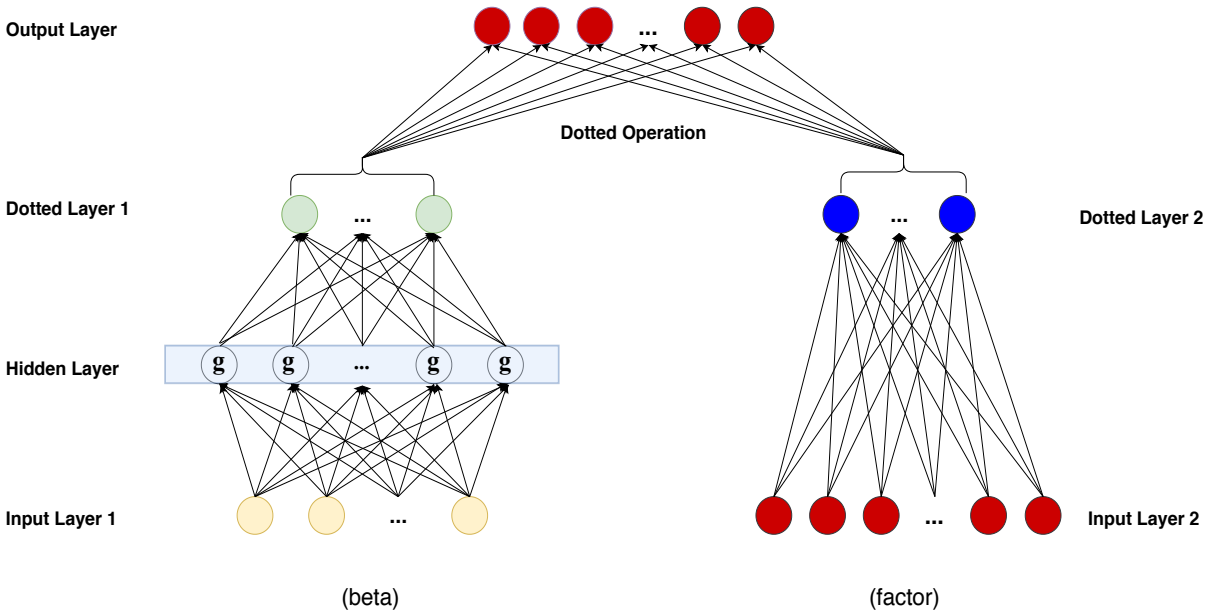
Proposition 1 shows that the linear autoencoder with a hidden layer of K neurons is a linear factor model with K latent factors. The estimated factor loadings are $\widehat{W}^{(1)}$, and the estimated factors are $\widehat{W}^{(0)}R$. These span the same spaces as P and Q in equation (6). Needless to say, autoencoder models are more general than the linear factor model as they allow for dimension reduction via nonlinear transformations of r_t. This additional flexibility has proven valuable in a variety of applications outside of finance. For example, Hinton and Salakhutdinov (2006) show that deep autoencoders handily outperform shallow or linear autoencoders for image recognition.

2.2 Extending the Autoencoder Model to Include Covariates

The static linear factor model has been an extremely productive tool for studying asset returns, but recent research highlights a number of its limitations. The distribution of asset returns is well known to be highly time-varying, and static factor models abstract from a wealth of relevant conditioning information. For example, KPS demonstrate large empirical gains from incorporating asset-specific covariates in the specification of factor loadings. Not only do these covariates improve the estimation of loadings, they also indirectly improve estimates of the latent factors themselves. Their model formulation amounts to a combination of two linear models: One linear specification for the latent factors, shown in equation (1), and another linear specification for conditional betas, shown in (2).

While the standard autoencoder in (5) is a powerful tool for dimension reduction, it shares the same limitation as PCA that it does not leverage conditioning variables to identify the factor structure, and instead relies only on returns themselves. To overcome this limitation, we design a new neural network structure by augmenting the standard autoencoder model to incorporate covariates.

Figure 2: Conditional Autoencoder Model



Note: 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 (in green) depend on firm characteristics (in yellow) of the input layer 1 through an activation function g on neurons of the hidden layer. The right-hand side describes the corresponding factors (in blue), which are weighted combinations of input layer 2.

Figure 2 illustrates the basic structure of our conditional autoencoder. The left side of the network models factor loadings as a nonlinear function of covariates (e.g., asset characteristics), while the right side network models factors as portfolios of individual stock returns. At the highest level, the mathematical representation of the model is identical to equation (1):

$$r_{i,t} = \beta'_{i,t-1} f_t + u_{i,t}. \tag{9}$$

The first key difference between our model and IPCA is in the formulation of conditional betas. We specify the $K \times 1$ vector $\beta_{i,t-1}$ as a neural network model of lagged firm characteristics, $z_{i,t-1}$. The recursive formulation for the nonlinear beta function is:

$$z^{(0)}_{i,t-1} = z_{i,t-1}, \tag{10}$$

$$z^{(l)}_{i,t-1,k} = g\left(b^{(l-1)} + W^{(l-1)} z^{(l-1)}_{i,t-1}\right), l = 1, ..., L_\beta - 1, \tag{11}$$

$$\beta_{i,t-1} = b_0 + W_0 z^{(L_\beta - 1)}_{i,t-1}. \tag{12}$$

Equation (10) initializes the network as a function of the baseline characteristic data, $z_{i,t-1}$. The equations in (11) describe the nonlinear (and interactive) transformation of characteristics as they

propagate through hidden layer neurons.[8] Equation (12) describes how a set of K-dimensional factor betas emerge from the terminal output layer. This formalizes the left side of Figure 2.

On the right side of Figure 2, we see an otherwise standard autoencoder for the factor specification. The recursive mathematical formulation of the factors is:

$$r_t^{(0)} = r_t, \tag{13}$$

$$r_{t,k}^{(l)} = g\left(b^{(l-1)} + r^{(l-1)'} W^{(l-1)}\right), l = 1, ..., L_f - 1, \tag{14}$$

$$f_t = b^{(L_f-1)} + r_t^{(L-1)'} W^{(L_f-1)}. \tag{15}$$

Equation (13) initializes the network with the vector of individual asset returns, r_t. Equations in (14) transform and compress the dimensionality of returns as they propagate through hidden layers. Equation (15) describes the final set of K factors at the output layer.

At last, the "dotted operation" multiplies the $N \times K$ matrix output from the beta network with the $K \times 1$ output from the factor network to produce the final model fit for each individual asset return.

In practice, using the full cross section of individual stock returns in the factor network faces two daunting obstacles. The first is that the number of individual firms in our sample is roughly 30,000, which means that the number of weight parameters in the factor network can be astronomical, while the number of time series observations in our data set is a mere 720. Second, the panel is extremely unbalanced—in any given month, we have on average around 6,000 non-missing stocks, thus most of the stock-level weight parameters would require estimation from very few time series observations.

We therefore make one key modification to the factor side of the model that massively reduces the model's computational cost up front. Instead of initializing the network with the full cross section of stock returns in equation (13), we instead initialize it with a set of portfolios, defined as

$$x_t = (Z_{t-1}' Z_{t-1})^{-1} Z_{t-1} r_t. \tag{16}$$

This $L \times 1$ vector is a set of portfolios that are dynamically re-weighted (or "managed") on the basis of stock-level characteristics. The j^{th} element of x_t is akin to a return on a long-short portfolio constructed by sorting stocks based on the j^{th} characteristic. Initializing the model with $r_t^{(0)} = x_t$ accomplishes three things at once. First, it performs a preliminary reduction of the data that eliminates tens of thousands of parameters from the model. This preprocessing step in (16) can be viewed as adding a new initial layer to the factor neural network that dynamically (as a function of Z_{t-1}) collapses the N returns, r_t, down to L neurons, x_t, before proceeding with the rest of the network propagation. Second, it sidesteps issues of panel incompleteness, since portfolios are formed from the subset of stocks that are non-missing at each point in time. Third, it connects the factor autoencoder to the finance literature on characteristic-managed portfolios, including KPS,

[8] Equations (11) and (14) describe the k^{th} neuron in each hidden layer, and assume the convention that $z_{i,t-1}^{(l)}$ and $r_t^{(l)}$ are vectors that stack the output from all neurons in the l^{th} layer.

Feng et al. (2017), Kozak et al. (2017), and Giglio and Xiu (2018). KPS and Giglio and Xiu (2018), in particular, show that conditional linear factor models for individual stocks can be recast as static factor analysis on characteristic-managed portfolios.

2.2.1 Conditional Linear Factor Models as a Special Case

Just like the autoencoder model nests the static linear factor model, the augmented autoencoder nests the IPCA factor model proposed by KPS as a special case. KPS propose a method called IPCA to estimate their linear, time-varying beta model. IPCA solves the optimization problem:

$$\min_{\Gamma,F} \sum_{t=1}^{T} \sum_{i=1}^{N} \left\| r_{i,t} - z'_{i,t-1}\Gamma' f_t \right\|^2 = \min_{\Gamma,F} \sum_{t=1}^{T} \left\| r_t - Z_{t-1}\Gamma' f_t \right\|^2 . \tag{17}$$

where $F = (f_1, f_2, \ldots, f_T)$ and $Z_t = (z'_{1,t}, z'_{2,t}, \ldots, z'_{N,t})'$, and subject to restrictions that identify a unique rotation of factors.[9]

For comparison, consider a particularly simple version of the conditional autoencoder that uses a linear activation function and one layer of K neurons on both the beta side and the factor side of the network. In this case, $\beta'_{i,t} = Z_{t-1}W'_0$ and $f_t = W_1 x_t$, so the the estimation objective of the conditional autoencoder is:[10]

$$\min_{W_0,W_1} \sum_{t=1}^{T} \left\| r_t - Z_{t-1}W'_0 W_1 x_t \right\|^2 . \tag{18}$$

The next proposition formalizes the equivalence between IPCA and this linear conditional autoencoder.

Proposition 2. *The solution to* (18) *is equivalent to the solution of* (17) *if* $Z'_t Z_t = \Sigma$ *for a constant matrix* Σ.

In the general case where $Z'_t Z_t$ is non-constant, the two estimators are similar but no longer equivalent (see KPS). We find that the empirical performance of (17) and (18) is similar in our data.

2.3 Regularized Autoencoder Learning

Autoencoders, like neural networks more broadly, have many advantages relative to traditional linear factor models. In particular, the high capacity of a neural network model enhances its flexibility to construct the most informative features from data. With enhanced flexibility, however, comes a higher propensity to overfit. We take a variety of measures to alleviate overfitting, including a careful design of the empirical strategy and the extensive use of regularization.

[9] The identifying assumptions are

$$\Gamma\Gamma' = \mathbb{I}_K, \quad FF' \text{ is a diagonal matrix with descending diagonal entries}, \quad F\iota \geq 0.$$

These restrictions place no economic restrictions on the model and solely serve to pin down a uniquely identified solution to the first-order conditions.

[10] Without loss of generality, we suppress the bias term, i.e., b_0, in the neural networks, as it can instead be represented via a constant in the list of conditioning variables.

2.3.1 Training, Validation, and Testing

We divide our sample into three disjoint time periods that maintain the temporal ordering of the data. The first, or "training," subsample is used to estimate the model subject to a specific set of tuning hyperparameter values. These are critical to the performance of machine learning methods as they control model complexity.

The second, or "validation," sample is used for tuning the hyperparameters. We construct fitted values for data points in the validation sample based on the estimated model from the training sample. Next, we calculate the objective function based on errors from the validation sample, and iteratively search for hyperparameters that optimize the validation objective.

Tuning parameters are chosen from the validation sample taking into account estimated parameters, but the parameters are estimated from the training data alone. The idea of validation is to simulate an out-of-sample test of the model. Hyperparameter tuning amounts to searching for a degree of model complexity that tends to produce reliable out-of-sample performance. The validation sample fits are of course not truly out-of-sample because they are used for tuning, which is in turn an input to the estimation. Thus the third, or "testing," subsample, which is used for neither estimation nor tuning, is truly out-of-sample and thus is used to evaluate a method's out-of-sample performance.

2.3.2 Regularization Techniques

The most common machine learning device for guarding against overfitting is to append a penalty to the objective function in order to favor more parsimonious specifications. This "regularization" approach mechanically deteriorates a model's in-sample performance in the hope of improving its stability out-of-sample. This will be the case when penalization manages to reduce the model's fit of noise while preserving its fit of the signal.

Let $\mathcal{L}(\theta;\cdot)$ denote the objective function to optimize the autoencoder (9), where θ summarizes the weight parameters in the loading and factor networks of (10) through (15). We define our estimation objective to be

$$\mathcal{L}(\theta;\cdot) = \frac{1}{NT}\sum_{t=1}^{T}\sum_{i=1}^{N}\left\|r_{i,t} - \beta'_{i,t-1}f_t\right\|^2 + \phi(\theta;\cdot), \tag{19}$$

where $\phi(\theta)$ is a penalty function that regularizes the model. There are many choices for the penalty function $\phi(\cdot)$; we use on LASSO, or "l_1," penalization, which takes the form

$$\phi(\theta;\lambda) = \lambda\sum_{j}|\theta_j|.$$

The fortunate geometry of the LASSO penalty sets coefficients on a subset of covariates to exactly zero. In this sense, the LASSO imposes sparsity on weight parameters, encouraging insignificant weights to vanish. The LASSO penalty involves a non-negative hyperparameter, λ, which is deter-

mined in the validation sample.

In addition to l_1-penalization, we employ a second machine learning regularization tool known as "early stopping." It begins from an initial parameter guess that imposes parsimonious parameterization (for example, setting all θ values close to zero). In each step of the optimization algorithm, the parameter guesses are gradually updated to reduce fitting errors in the training sample. At each new guess, estimates are also constructed for the validation sample, and the optimization is terminated when the validation sample errors begin to increase. This typically occurs before the fitting errors are minimized in the training sample, hence its name (see Algorithm 1). By ending the parameter search early, parameters are shrunken toward the initial guess, and this is how early stopping regularizes against overfit. It is a popular substitute to "l_2"-penalization of θ parameters because it achieves regularization at a much lower computational cost.

As a third regularization technique, we adopt an ensemble approach in training our neural networks. In particular, we use multiple random seeds, say, 10, to initialize neural network estimation and construct model predictions by averaging estimates from all networks. This enhances the stability of the results because the stochastic nature of the optimization can cause different seeds to settle at different optima.

2.3.3 Optimization Algorithms

The high degree of nonlinearity and nonconvexity in neural networks, together with their rich parameterization, make brute force optimization highly computationally intensive (often to the point of infeasibility).

A common solution uses stochastic gradient descent (SGD) to train a neural network. Unlike standard gradient descent that uses the entire training sample to evaluate the gradient at each iteration of the optimization, SGD evaluates the gradient from a small random subset of the data at each iteration. This approximation sacrifices accuracy for enormous acceleration of the optimization routine. A critical tuning parameter in SGD is the learning rate, which controls the step size of the descent. It is necessary to shrink the learning rate toward zero as the gradient approaches zero, otherwise noise in the calculation of the gradient begins to dominate its directional signal. We adopt the adaptive moment estimation algorithm (Adam, Algorithm 2), an efficient version of SGD introduced by Kingma and Ba (2014) that computes adaptive learning rates for individual parameters using estimates of first and second moments of the gradients.

We also adopt "batch normalization" (Ioffe and Szegedy (2015), Algorithm 3), a simple technique for controlling the variability of predictors across different regions of the network and across different datasets. It is motivated by the phenomenon of internal covariate shift in which inputs of hidden layers follow different distributions than their counterparts in the validation sample. This issue is constantly encountered when fitting deep neural networks that involve many parameters and rather complex structures. For each hidden layer in each training step (a "batch"), the algorithm cross-sectionally de-means and variance standardizes the batch inputs to restore the representation power of the unit.

3 An Empirical Study of US Equity

3.1 Data

We analyze the same dataset studied in Gu et al. (2018), which contains monthly individual stock returns from the Center for Research in Securities Prices (CRSP) for all firms listed in the three major exchanges: NYSE, AMEX, and NASDAQ. We use the Treasury bill rate to proxy for the risk-free rate from which we calculate individual excess returns. Our sample begins in March 1957 (the start date of the S&P 500) and ends in December 2016, totaling 60 years.

In addition, we build a large collection of stock-level predictive characteristics based on the cross section of stock returns literature. These include 94 characteristics (61 of which are updated annually, 13 updated quarterly, and 20 updated monthly), see Gu et al. (2018) for a full list. Most of these characteristics are released to the public with a delay. To avoid a forward-looking bias, we assume that monthly characteristics are delayed by at most one month, quarterly releases are delayed with a four month lag, and annual releases with a six month lag. Thus, we match realized returns at month t with the most recent monthly characteristics at the end of month $t-1$, the most recent quarterly data as of $t-4$, and most recent annual data as of $t-6$. Observations are occasionally missing some characteristics. We replace a missing characteristic with the cross-sectional median of that characteristic during that month.

Distributions of some characteristics are highly skewed and leptokurtic. Following a common tack in the literature that avoids undue influence of outlying observations, we rank-normalize all characteristics into the interval $(-1, 1)$ each month t. We then form 94 managed portfolios using (16). We also include one equal-weighted market portfolio that corresponds to a constant regressor in Z_{t-1}.

Unlike the existing literature, we do not impose any filters based on stock prices or share codes, or rule out financial firms. Past literature has imposed these filters in large part because they find it difficult to reconcile the return behavior of low price stocks, uncommon share codes, and financial sector stocks with the rest of the sample. We have no such difficulty thanks to the superior capacity of our model and the rich feature sets we allow for in our framework. The total number of stocks in our sample is nearly 30,000, with the average number of stocks per month exceeding 6,200.

3.2 Models Comparison Set

We compare a range of latent factor models in our empirical analysis. The first model, which we refer to as "PCA," corresponds to specification (5). It assumes a linear functional form, constant betas, no conditioning information, and is estimated via PCA.[11] The second model we refer to as "IPCA" and follows the KPS specification of (1) and (2). In particular, it assumes a linear factor structure and conditional betas that are likewise linear in covariates.

[11] The universe of individual stocks changes over time, so our PCA estimation must cope with unbalanced panels. We use an EM algorithm for PCA (Stock and Watson, 2002). The IPCA algorithm devised by KPS is robust to missing data. Likewise, the SGD algorithm for autoencoder models is not affected by missing data, because individual stock returns across periods are collected in a single pool from which random batches of observations are drawn.

We then consider a range of conditional autoencoder (CA) architectures with varying degrees of complexity. The simplest, which we denote CA_0, uses a single linear layer in both the beta and factor networks as described in (18), making it similar (but not identical) to IPCA. Next, CA_1 adds a hidden layer with 32 neurons in the beta network, but continues to assume a single linear layer on the factor side. CA_2 and CA_3 add a second and third hidden layer, with 16 and 8 neurons respectively, to the beta side.

CA_0 through CA_3 all maintain a one-layer linear specification on the factor side of the model. In these cases, the only variation in factor specification is in the number of neurons, which we allow to range from 1 to 6, and which corresponds to the number of factors in the model.

We also compare the autoencoder specifications against benchmark models with observable factors. We refer to these models collectively as "FF," which possess 1 to 6 factors. The first observable factor is the excess market return, then we add SMB, HML, and UMD, sequentially. The five-factor model is the market, SMB, HML, CMA, and RMW, and the six-factor model again appends UMD.[12]

We divide the 60 years of data into 18 years of training sample (1957 - 1974), 12 years of validation sample (1975 - 1986), and the remaining 30 years (1987 - 2016) for out-of-sample testing. Because machine learning algorithms are computationally intensive, we avoid recursively refitting models each month. Instead, we refit once every year as most of our signals are updated once per year. Each time we refit, we increase the training sample by one year. We maintain the same size of the validation sample, but roll it forward to include the most recent twelve months.

3.3 Statistical Performance Evaluation

We evaluate model performance using the total and predictive R^2 defined by KPS. These pool errors across firms and over time into grand panel-level assessments of each model. The total R^2 quantifies the explanatory power of contemporaneous factor realizations, and thus assesses the model's description of individual stock riskiness:

$$R^2_{\text{total}} = 1 - \frac{\sum_{(i,t)\in OOS}(r_{i,t} - \widehat{\beta}'_{i,t-1}\widehat{f}_t)^2}{\sum_{(i,t)\in OOS} r^2_{i,t}}. \tag{20}$$

The OOS set indicates that fits are only assessed on the testing subsample, whose data never enter into model estimation or tuning.

The predictive R^2 assesses the accuracy of model-based predictions of future individual excess stock returns. This quantifies a model's ability to explain panel variation in risk compensation. It is defined as

$$R^2_{\text{pred}} = 1 - \frac{\sum_{(i,t)\in OOS}(r_{i,t} - \widehat{\beta}'_{i,t-1}\widehat{\lambda}_{t-1})^2}{\sum_{(i,t)\in OOS} r^2_{i,t}}, \tag{21}$$

where $\widehat{\lambda}_{t-1}$ is the prevailing sample average of $\widehat{f}$ up to month $t-1$.

[12] Market, SMB, HML, CMA, RMW, and UMD factor returns are from Ken French's website.

Table 1 reports the out-of-sample total R^2 for individual stocks, r_t. The best overall model in terms of explained out-of-sample return variation is IPCA with six-factors, which delivers a 14.5% total R^2. It is closely followed by the conditional autoencoder with one hidden beta layer of 32 neurons (CA_1) and six factors, which achieves an out-of-sample R^2 of 14.3%. Other CA models are similar but slightly weaker.

Interestingly, and somewhat surprisingly, the worst performing models are those with observable factors. We can trace the poor performance of observable factor models to two features of our empirical design. The first is that we infrequently re-estimate model parameters. At the individual stock level, betas are highly time varying, and infrequent re-estimation clearly reveals a shortcoming of traditional observable factor and their static beta formulations. In contrast, we find that infrequent re-estimation has little effect on the out-of-sample fits for conditional models (both IPCA and CA versions), likewise revealing the comparative strength of these methods—their ability to capture time variation through covariates rather than through ad hoc rolling parameter updates. Second, we analyze a much large cross section of stocks (nearly 30,000) than typically studied in the literature (e.g., roughly 12,000 in KPS). This reveals that observable factors are poorly suited to describe the performance of the much larger universe of stocks that we cover. Note that, even though our current data set is much larger than that of KPS, the performance of IPCA is remarkably robust to this change in environment.

Table 1 also reports the out-of-sample total R^2 at the level of managed portfolios, x_t. These portfolios are large and diversified collections of individual stocks, thus much of the idiosyncratic risk in the data is averaged out. As a result, total R^2 at the portfolio-level tends to be far higher. It is still the case that IPCA provides the best fit, followed by CA_1. The comparative performance of observable factor FF models is much improved at the portfolio level,[13] but nonetheless dominated by conditional latent factor models.

Next, Table 2 compares models in terms of predictive R^2. Whereas IPCA dominated in terms of total R^2, its predictive R^2 of 0.3% per month is nearly doubled by the predictive power of (deep) conditional autoencoders. CA_1, CA_2, and CA_3 generate a predictive R^2 of 0.53%, 0.58%, and 0.57%, respectively. All of conditional models, including IPCA and CA_0, dramatically outperform the static FF and PCA models, which generally fail to produce any out-of-sample predictability whatsoever.

3.4 Economic Performance Evaluation

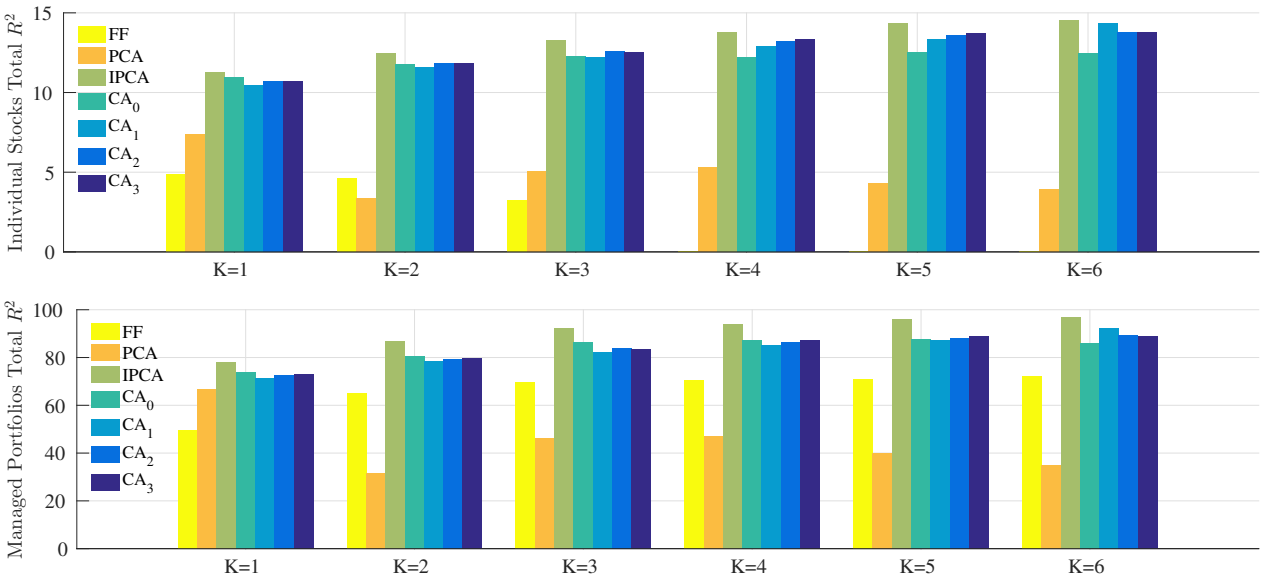
It is difficult to infer the economic contribution of a model from R^2 alone. To assess model performance in economic terms, we evaluate how return predictions from each model translate into Sharpe ratios for portfolios formed based on those predictions.

For each model, we sort stocks into deciles based on the model's out-of-sample return forecasts. We construct a zero-net-investment portfolio that buys the highest expected return stocks (decile 10) and sells the lowest (decile 1). We rebalance portfolios each month, and consider both equal-weighted and value-weighted portfolios.

[13] Intuitively, dynamically reweighting portfolios to maintain roughly constant characteristic values reduces time variation in portfolio betas and thus gives static factor models (including PCA) a better chance to fit the data.

Table 1: Out-of-Sample R^2_{total} Comparison

Model	Test Assets	K					
		1	2	3	4	5	6
FF	r_t	4.8	4.6	3.4	0.1	-2.3	-6.1
	x_t	49.4	64.8	69.5	70.4	71.1	72.2
PCA	r_t	7.3	3.3	5.0	5.3	4.2	3.9
	x_t	66.6	31.7	46.2	47.1	39.8	34.8
IPCA	r_t	11.2	12.4	13.3	13.7	14.3	14.5
	x_t	78.0	86.8	92.1	93.8	96.0	96.7
CA_0	r_t	10.9	11.8	12.3	12.2	12.5	12.4
	x_t	73.7	80.4	86.2	87.1	87.7	85.9
CA_1	r_t	10.4	11.5	12.2	12.9	13.4	14.3
	x_t	71.4	78.3	82.2	85.2	87.2	92.2
CA_2	r_t	10.7	11.8	12.6	13.2	13.6	13.8
	x_t	72.7	79.5	84.0	86.4	88.2	89.3
CA_3	r_t	10.7	11.8	12.5	13.3	13.7	13.8
	x_t	73.1	79.9	83.6	87.1	88.8	89.0

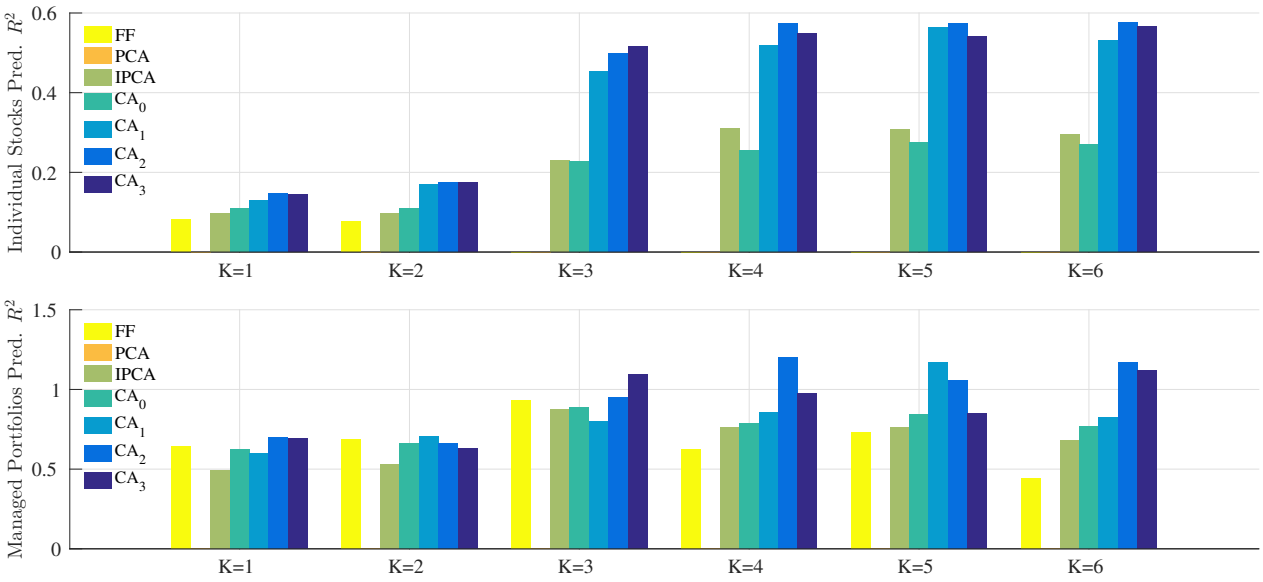


Note: In this table, we report the out-of-sample total R^2 for individual stocks r_t and managed portfolios x_t using observable factor models (FF), PCA, IPCA, and conditional autoencoders CA_0 through CA_3. In all cases, the number of factors K varies from 1 to 6.

Table 3 reports the annualized Sharpe ratios of these 10-1 spread portfolios over our 30-year out-of-sample period. The results essentially recast the model comparison of predictive R^2 in terms of economic magnitudes. The overall best performing portfolio is that based on the conditional autoencoder with two hidden beta layers, CA_2. This model achieves an Sharpe ratio of 2.63 for the equal-weighted portfolio, and 1.53 with value weights. The performance of CA_1 and CA_3 is only slightly lower. Following the nonlinear conditional autoencoders, the best model is IPCA, which delivers Sharpe ratios of 2.25 and 0.96 with equal and value weights, respectively (and which

Table 2: Out-of-Sample R^2_{pred} Comparison

Model	Test Assets	K					
		1	2	3	4	5	6
FF	r_t	0.08	0.08	< 0	< 0	< 0	< 0
	x_t	0.65	0.69	0.93	0.62	0.73	0.45
PCA	r_t	< 0	< 0	< 0	< 0	< 0	< 0
	x_t	< 0	< 0	< 0	< 0	< 0	< 0
IPCA	r_t	0.10	0.10	0.23	0.31	0.31	0.30
	x_t	0.49	0.53	0.88	0.76	0.76	0.68
CA_0	r_t	0.11	0.11	0.23	0.25	0.27	0.27
	x_t	0.63	0.66	0.89	0.79	0.84	0.77
CA_1	r_t	0.13	0.17	0.45	0.52	0.56	0.53
	x_t	0.60	0.70	0.80	0.85	1.17	0.83
CA_2	r_t	0.15	0.17	0.50	0.57	0.57	0.58
	x_t	0.70	0.66	0.95	1.20	1.06	1.17
CA_3	r_t	0.14	0.17	0.52	0.55	0.54	0.57
	x_t	0.69	0.63	1.10	0.97	0.85	1.12



Note: In this table, we report the out-of-sample predictive R^2 for individual stocks r_t and managed portfolios x_t using observable factor models (FF), PCA, IPCA, and conditional autoencoders CA_0 through CA_3. In all cases, the number of factors K varies from 1 to 6.

outperforms the linear conditional autoencoder CA_0). Finally, corroborating the R^2 results above, static linear models FF and PCA broadly exhibit poor out-of-sample portfolio performance.

To evaluate the multi-factor mean-variance efficiency of each model, we report the ex ante unconditional tangency portfolio Sharpe ratio among factor portfolios. We calculate out-of-sample factor returns following the same re-estimation approach described earlier. The tangency portfolio return for a set of factors is constructed on a purely out-of-sample basis by using the mean and covariance matrix of estimated factors through t and tracking the post-formation $t + 1$ return.

Table 3: Out-of-Sample Sharpe Ratios of Long-Short Portfolios

	K					
Equal-Weight	1	2	3	4	5	6
FF	-0.66	-0.85	-0.40	-0.30	0.36	-0.21
PCA	0.28	0.09	0.13	-0.08	-0.12	0.15
IPCA	0.20	0.19	1.26	2.16	2.31	2.25
CA_0	0.23	0.32	1.34	1.87	2.10	2.18
CA_1	0.30	0.39	2.12	2.63	2.67	2.60
CA_2	0.30	0.38	2.16	2.64	2.68	2.63
CA_3	0.31	0.38	2.19	2.57	2.57	2.59
	K					
Value-Weight	1	2	3	4	5	6
FF	-0.82	-1.13	-0.69	-0.60	0.18	-0.53
PCA	0.12	-0.18	0.05	-0.10	-0.30	-0.08
IPCA	-0.15	-0.07	0.59	0.81	1.05	0.96
CA_0	-0.11	-0.03	0.41	0.81	0.83	0.88
CA_1	-0.03	0.11	0.91	1.30	1.48	1.40
CA_2	-0.03	0.08	0.92	1.39	1.45	1.53
CA_3	-0.02	0.08	1.09	1.41	1.34	1.51

Note: In this table, we report annualized out-of-sample Sharpe ratios for long-short portfolios using Fama-French models (FF), a vanilla factor model (5), and a variety of autoencoders, A_0, A_1, A_2, A_3, based on (9), respectively, where the number of factors in (5) or the number of neurons in the hidden layer on the right-hand side of (9), K, varies from 1 to 6.

We report results in Table 4. All conditional factor specifications (IPCA and CA_0 through CA_1) produce high unconditional Sharpe ratio statistics, consistent with the findings of KPS. The most dominant overall model on this dimension is CA_3 with five factors, though performance is broadly similar for CA_1 through CA_3. Static factor models perform markedly worse than conditional models. Results in Table 4 reflect the fact that conditional models capture extensive comovement among assets while, at the same time, reconciling their differences in average returns with their factor loadings. These results should not be viewed as performance of implementable trading strategies. The factor tangency portfolios describe the mean-variance efficiency of models without considering practical frictions such as trading costs. Instead, they should be viewed as providing a non-implementable but nonetheless helpful quantitative comparison of models' mean-variance efficiency in economic terms.

3.5 Risk Premia vs. Mispricing

An important implication emerges from a comparison of Table 2 versus the return prediction analysis of Gu et al. (2018). In their paper, the best performing machine learning model forecasts monthly individual stock returns (in the exact same data set as ours) with an R^2 of 0.58%. Yet theirs are pure prediction models—there is no factor structure or risk-return tradeoff—and thus they make no distinction between predictability coming through compensation for risk exposure, and compensation from mispricing (i.e., alpha). In contract, the nonlinear factor models in this paper force all the characteristic-based predictability to come solely through factor risk exposures. That is, the conditional autoencoder models are all specified without an intercept, thus *they impose the economic*

Table 4: Out-of-Sample Factor Tangency Portfolio Sharpe Ratios

	K					
	1	2	3	4	5	6
FF	0.51	0.41	0.53	0.71	0.71	0.82
PCA	0.35	0.23	0.25	0.38	0.48	0.55
IPCA	0.39	0.44	1.81	3.14	3.71	3.72
CA_0	0.42	0.48	1.47	1.76	1.94	1.97
CA_1	0.56	0.91	3.18	3.82	3.63	4.58
CA_2	0.54	0.75	3.56	4.26	4.72	2.77
CA_3	0.54	0.77	3.94	4.75	4.94	4.37

Note: In this table, we report annualized out-of-sample sharpe ratios for the mean-variance efficient portfolio of factors using observable factor models (FF), PCA, IPCA, and conditional autoencoders CA_0 through CA_3. In all cases, the number of factors K varies from 1 to 6. We scale the tangency weights each month by targeting 1% monthly volatility based on historical estimates of factor risk premium and covariance matrix.

restriction of no-arbitrage. Despite this restriction, the conditional autoencoder model achieves nearly identical predictive power for monthly stock returns, 0.58% for the CA_2 specification. This is a significant result—it suggests that stock characteristics predict returns not because they capture "anomalous" compensation without risk, but rather because the characteristics proxy for (and help identify) compensated factor risk exposures.

In this section, we directly test whether the zero-intercept no-arbitrage restriction is satisfied in the data. If it is, the time series average of model residuals for each asset—that is, the pricing errors in our model—should be statistically indistinguishable from zero. We focus this analysis on unconditional pricing errors, defined as:

$$\alpha_i := \mathrm{E}(u_{i,t}) = \mathrm{E}(r_{i,t}) - \mathrm{E}(\beta'_{i,t-1} f_t).$$

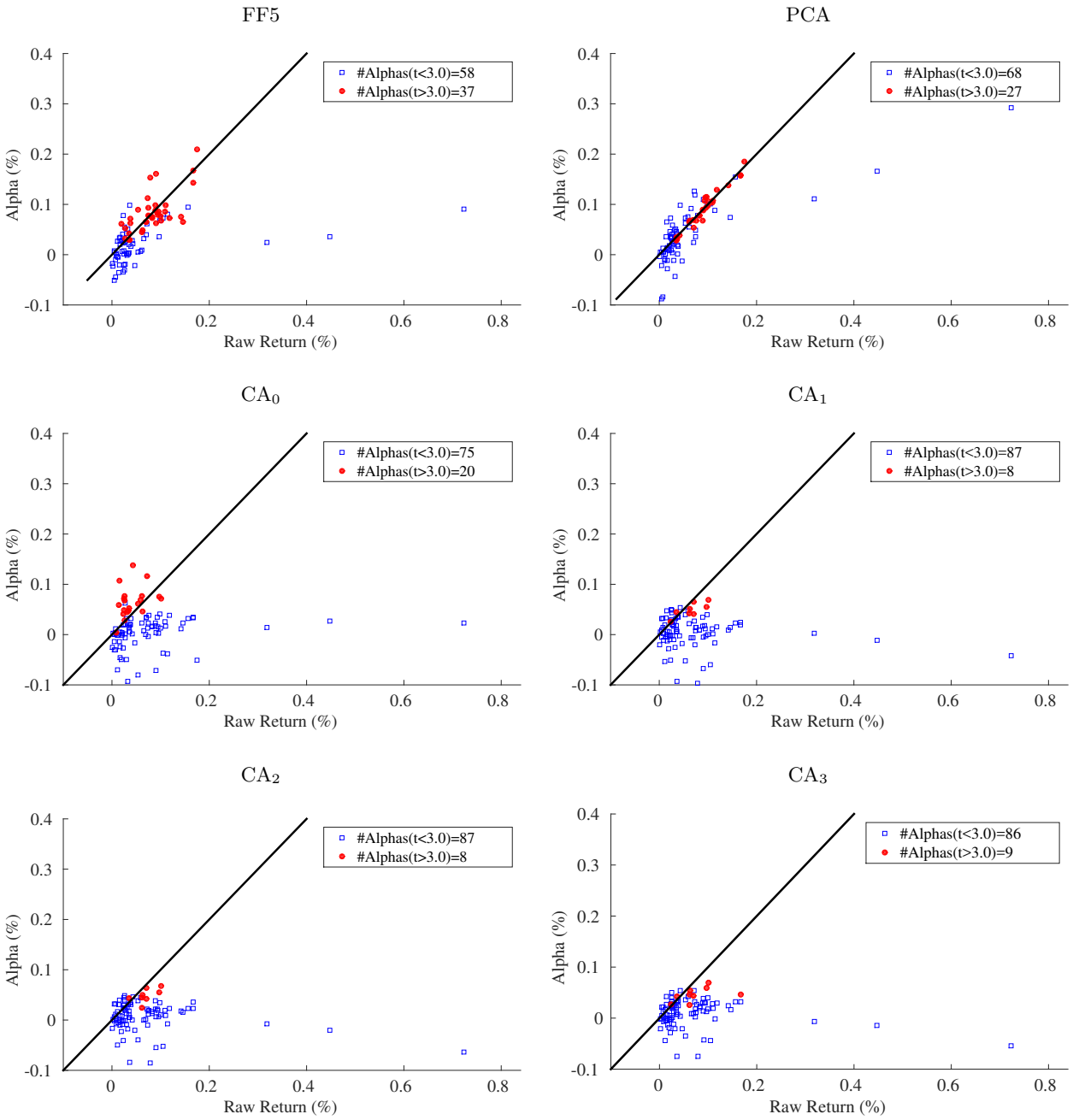
Alphas for the managed portfolios x_t are defined analogously.

We focus our pricing error tests on x_t, whose comparatively low dimensionality avoids inferential difficulties that arise with r_t.[14] To construct estimates of the out-of-sample pricing error, we calculate the average difference between x_t and its out-of-sample model fit. These pricing errors can be interpreted as the average gain of a hedging portfolio that has a zero-exposure on any systematic factors. In a no-arbitrage model, zero-exposure assets should earn zero excess return.

Figure 3 scatters the estimated out-of-sample pricing errors for each model against the average returns of x_t. The figure also reports the number of alphas whose t-statistics exceed 3.0. The overall magnitude of alphas shrinks as we move from static linear models to nonlinear conditional autoencoders. For the five-factor Fama-French model, 37 of the 95 managed portfolios have alpha t-statistics in excess of 3.0. For CA_2, that number drops to 8 out of 95. Furthermore, those that remain significant are economically small (below 7 basis points per month) compared to alphas from the Fama-French model.

[14] For example, stock-level idiosyncratic risk is so large that stock-level alpha estimates tend to be extremely noisy.

Figure 3: Out-of-sample Pricing Errors Across Models

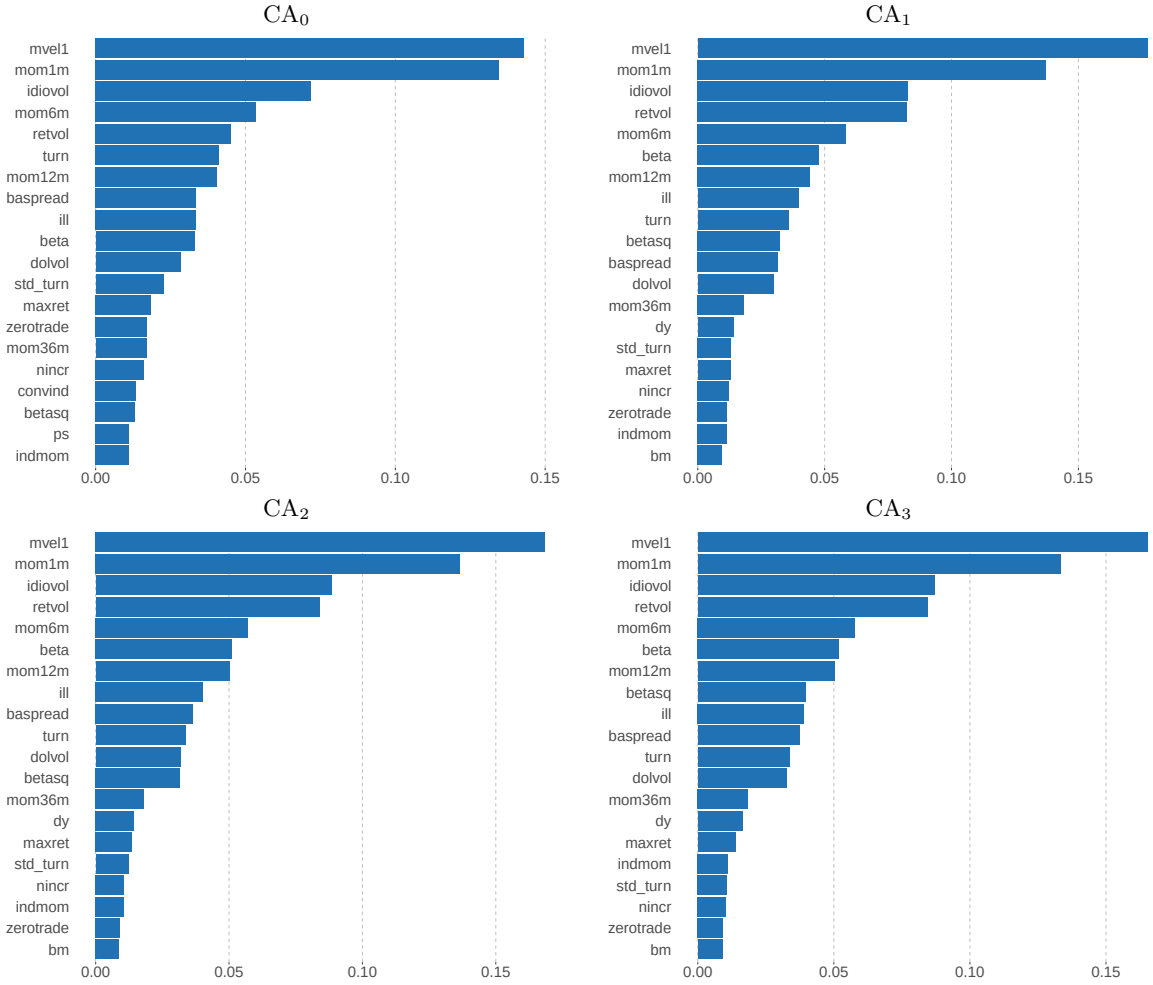


Note: The figure reports out-of-sample pricing errors (alphas) for 95 characteristic-managed portfolios x_t, relative to the Fama-French five-factor model (FF5), the static linear latent five-factor model (PCA), and conditional autoencoders (CA_0 through CA_3). Alphas with t-statistics in excess of 3.0 are shown in red dots, while insignificant alphas are shown in hollow squares.

3.6 Characteristics Importance

Following Gu et al. (2018), we identify influential covariates by ranking them according to a notion of variable importance, defined as the reduction in total R^2 resulting from setting all values of a

Figure 4: Top Twenty Characteristics by Variable Importance



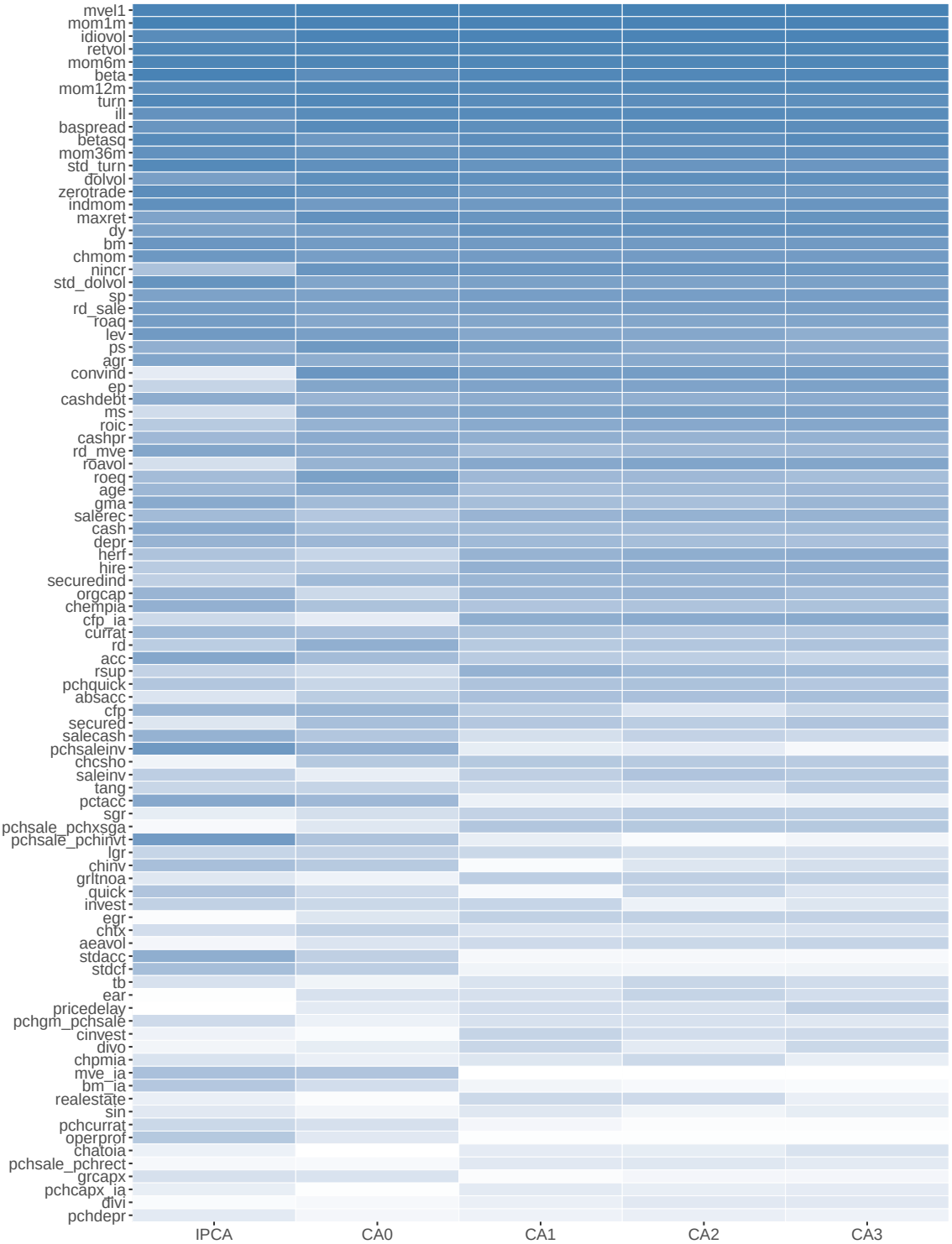
Note: This figure compares variable importance for the top twenty most influential variables in each model, based on an average over all training samples. The variable importances within each model are normalized to sum to one. All models fix $K = 5$.

given characteristic to zero while holding the remaining model estimates fixed. For this analysis, we focus on the five-factor specification of each model.

Figures 4 illustrates characteristic importance for each conditional autoencoder specification. It focuses on the top 20 characteristics for each model. Beyond these, variable importance hovers near zero (we show importance for the full list of characteristics in Figure 5). The total contribution by the top twenty characteristics is around 80% for CA_0, and 90% for CA_1 through CA_3.

Three categories of characteristics stand out as the most influential. The first is a price trend category, which includes short-term reversal (mom1m), stock momentum (mom12m), momentum change (chmom), industry momentum (indmom), recent maximum return (maxret), and long-term reversal (mom36m). The second category includes liquidity variables, such as turnover and turnover volatility (turn, std_turn), log market equity (mvel1), dollar volume (dolvol), Amihud illiquidity (ill), number of zero trading days (zerotrade), and bid-ask spread (baspread). Risk measures constitute

Figure 5: Importance Rankings of All Characteristics



Note: This figure ranks 94 stock-level characteristics in terms of overall model contribution. Characteristics are ordered based on the sum of their ranks over all models, with the most influential characteristics on top and least influential on bottom. Columns correspond to individual models, and color gradients within each column indicate the most influential (dark blue) to least influential (white) variables.

the third influential group, including total and idiosyncratic return volatility (retvol, idiovol), market beta (beta), and beta-squared (betasq). Interestingly, all variants of the autoencoder model agree on the importance of these three categories. Moreover, these results closely coincide with the findings of Gu et al. (2018), who track variable importance using R^2_{pred} (there is no notion of R^2_{total} in their analysis because they focus solely on prediction and therefore do not consider contemporaneous factor associations). This consistency of variable importance across different objectives is an indication of robustness in our list of key variables, and an indication that these variables matter for understanding variation in both expected returns and realized returns.

4 Monte Carlo Simulations

To demonstrate the finite sample performance of our autoencoder learning method, we simulate a conditional 3–factor model for excess returns r_t, for $t = 1, 2, \ldots, T$:

$$r_{i,t} = \beta_{i,t-1} f_t + \varepsilon_{i,t}, \quad \beta_{i,t-1} = g^*(c_{i,t-1}; \theta), \quad f_t = W x_t + \eta_t$$

where c_t is an $N \times P_c$ matrix of characteristics, f_t is a 3×1 vector of factors, x_t is a $P_x \times 1$ vector of factor components, W is a $3 \times P_x$ factor weighting matrix, and η_t and ε_t are 3×1 and $N \times 1$ vectors of idiosyncratic errors respectively.

We choose $x_t \sim \mathcal{N}(0.03, 0.1^2 \times \mathbb{I}_{P_x})$, $\eta_t \sim \mathcal{N}(0, 0.01^2 \times \mathbb{I}_3)$ and $\varepsilon_{i,t} \sim t_5(0, 0.1^2)$, in which their variances are calibrated so that the average time series R^2 is about 45% and the average annualized volatility is around 60%.

We simulate the panel of characteristics for each $1 \leq i \leq N$ and each $1 \leq j \leq P_c$ from the following model:

$$c_{ij,t} = \frac{2}{n+1} \text{rank}(\bar{c}_{ij,t}) - 1, \quad \bar{c}_{ij,t} = \rho_j \bar{c}_{ij,t-1} + \epsilon_{ij,t}, \tag{22}$$

where $\rho_j \sim \mathcal{U}[0.9, 1]$, and $\epsilon_{ij,t} \sim \mathcal{N}(0, 1)$, so that the characteristics feature some degree of persistence over time, yet is cross-sectionally normalized to be within $[-1, 1]$. This matches our data cleaning procedure in the empirical study.

We consider one fixed matrix W and two cases of $g^\star(\cdot)$ functions:

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

(a) Linear factor loadings: $g^*(c_{i,t}; \theta) = (1.2 \times c_{i1,t}, c_{i2,t}, 0.8 \times c_{i3,t})'$.

(b) Non-linear factor loadings: $g^*(c_{i,t}; \theta) = \left(c^2_{i1,t}, 2 \times (c_{i1,t} \times c_{i2,t}), 0.6 \times \text{sgn}(c_{i3,t})\right)'$.

Throughout, we fix $N = 200$, $T = 180$ and $P_c = P_x = 50$. In both cases, $g^\star(\cdot)$ only depends on 3 covariates, so there are only 3 non-zero entries in θ, denoted as θ_0. Case (a) is simple and sparse linear

Table 5: Comparison of Total R^2s and Predictive R^2s in Simulations

Model (a)	K					
Total. R^2	1	2	3	4	5	6
PCA	3.5	4.7	5.5	6.3	7.1	7.8
IPCA	18.6	32.2	40.7	41.0	41.4	41.7
CA_0	15.6	26.7	33.7	33.5	33.4	33.2
CA_1	17.6	30.3	38.1	37.7	37.3	37.1
CA_2	17.7	29.2	36.8	36.5	36.3	35.9
CA_3	17.6	25.6	30.0	29.5	26.3	23.4
Pred. R^2						
PCA	0.17	0.10	0.04	0.01	-0.01	-0.03
IPCA	2.20	2.93	3.33	3.32	3.32	3.32
CA_0	2.04	2.84	3.17	3.14	3.12	3.13
CA_1	2.11	2.93	3.27	3.29	3.26	3.26
CA_2	2.10	2.85	3.22	3.22	3.23	3.22
CA_3	2.06	2.57	2.89	2.86	2.58	2.39

Model (b)	K					
Total. R^2	1	2	3	4	5	6
PCA	3.4	5.1	6.0	6.6	7.3	7.9
IPCA	11.0	11.4	11.9	12.3	12.7	13.1
CA_0	8.5	8.2	7.9	7.6	7.4	7.2
CA_1	15.0	24.6	31.8	32.0	31.9	31.8
CA_2	15.7	23.5	30.9	31.8	30.2	28.2
CA_3	15.9	15.6	14.6	14.0	11.2	9.2
Pred. R^2						
PCA	0.15	0.19	0.15	0.12	0.10	0.09
IPCA	0.84	0.82	0.81	0.80	0.79	0.79
CA_0	0.80	0.76	0.77	0.76	0.72	0.70
CA_1	1.83	2.31	2.70	2.70	2.71	2.73
CA_2	1.95	2.24	2.73	2.80	2.69	2.53
CA_3	1.77	1.43	1.32	1.26	1.06	0.86

Note: In this table, we report the average out-of-sample (OOS) Total R^2s and Predictive R^2s for models (a) and (b) using PCA, IPCA, CA_0, CA_1, CA_2 and CA_3, respectively. We fix $N = 200$, $T = 180$, and $P_c = P_x = 50$. The number of Monte Carlo repetitions is 100.

model. Case (b) involves a nonlinear covariate $c^2_{i1,t}$, a nonlinear and interaction term $(c_{i1,t} \times c_{i2,t})$, and a dummy variable $\text{sgn}(c_{i3,t})$. We calibrate the values of θ_0 such that the total R^2 is around 40%, and the predictive R^2 is 5%.

For each Monte Carlo sample, we divide the whole time series into 3 consecutive subsamples of equal length for training, validation, and testing, respectively. For Vanilla PCA and IPCA, we combine training and validation samples together because they don't have any tuning parameter. For CA_0, CA_1, CA_2 and CA_3 we estimate them in the training sample, then choose tuning parameters for each method in the validation sample, and calculate the prediction errors in the testing sample.

We report the average OOS total and predictive R^2s for each method over 100 Monte Carlo repetitions in Table 5. For model (a), IPCA delivers the best OOS total and predictive R^2s. This is not surprising given that the true model is sparse and linear in the input covariates. More advanced

methods such as CA_1, CA_2 and CA_3 tend to overfit, so their performance is slightly worse. By contrast, for model (b), these methods clearly beat IPCA, because the latter cannot capture the nonlinearity in the model. The Vanilla PCA method is always overfitting in the training sample so that it cannot achieve a good OOS R^2s. The comparison among autoencoder models demonstrates a stark trade-off between model flexibility and implementation difficulty. As shown in the table, shallower conditional autoencoders tend to outperform in our simulation setting, which is consistent with our findings in empirical analysis.

Overall, the simulation results suggest that the conditional autoencoder methods are successful in learning the factor structure in both linear and nonlinear situations. This is not surprising, as these methods are implemented to improve fitting and prediction by allowing more complex functional forms of conditional factor loadings.

5 Conclusion

We propose a new approach to latent factor modeling for asset pricing that draws on autoencoder methods from the machine learning literature. We adapt the standard autoencoder to allow latent factors and factor exposures to depend on asset characteristic conditioning variables. The result is a nonlinear conditional asset pricing model that embeds the economic restriction of no-arbitrage within a broader neural network framework.

In the empirical context of monthly US stock returns, our conditional autoencoder model dominates competing asset pricing models, including Fama-French models, PCA methods, and linear conditioning methods such as IPCA. A long-short decile spread portfolios sorted on stock return predictions from our preferred autoencoder produces an annualized value-weighted Sharpe ratio of 1.53, beating the next closest competitor (IPCA, with Sharpe ratio 0.96) by a wide margin, and on a purely out-of-sample basis. Finally, the pricing errors in our model (likewise measured on an out-of-sample basis) are a fraction of the magnitude of those from traditional Fama-French factor models.

References

Ahn, Seung C., and Alex R. Horenstein, 2013, Eigenvalue ratio test for the number of factors, *Econometrica* 81, 1203–1227.

Aït-Sahalia, Yacine, and Dacheng Xiu, 2017, Using principal component analysis to estimate a high dimensional factor model with high-frequency data, *Journal of Econometrics* 201, 388–399.

Alessi, Lucia, Matteo Barigozzi, and Marco Capasso, 2010, Improved penalization for determining the number of factors in approximate factor models, *Statistics and Probability Letters* 80, 1806–1813.

Amengual, Dante, and Mark W. Watson, 2007, Consistent estimation of the number of dynamic factors in a large N and T panel, *Journal of Business and Economic Statistics* 25, 91–96.

Bai, Jushan, 2003, Inferential Theory for Factor Models of Large Dimensions, *Econometrica* 71, 135–171.

Bai, Jushan, and Serena Ng, 2002, Determining the number of factors in approximate factor models, *Econometrica* 70, 191–221.

Bai, Jushan, and Serena Ng, 2017, Principal components and regularized estimation of factor models, Technical report, Columbia University.

Baldi, Pierre, and Kurt Hornik, 1989, Neural networks and principal component analysis: Learning from examples without local minima, *Neural networks* 2, 53–58.

Bansal, Ravi, and Amir Yaron, 2004, Risks for the long run: A potential resolution of asset pricing puzzles, *The journal of Finance* 59, 1481–1509.

Bourlard, Hervé, and Yves Kamp, 1988, Auto-association by multilayer perceptrons and singular value decomposition, *Biological cybernetics* 59, 291–294.

Campbell, John Y, and John H Cochrane, 1999, By force of habit: A consumption-based explanation of aggregate stock market behavior, *Journal of political Economy* 107, 205–251.

Chamberlain, Gary, and Michael Rothschild, 1983, Arbitrage, factor structure, and mean-variance analysis on large asset markets, *Econometrica* 51, 1281–1304.

Connor, Gregory, Matthias Hagmann, and Oliver Linton, 2012, Efficient semiparametric estimation of the fama–french model and extensions, *Econometrica* 80, 713–754.

Connor, Gregory, and Robert A Korajczyk, 1986, Performance measurement with the arbitrage pricing theory: A new framework for analysis, *Journal of Financial Economics* 15, 373–394.

Fama, Eugene F, and Kenneth R French, 1993, Common risk factors in the returns on stocks and bonds, *Journal of financial economics* 33, 3–56.

Fan, Jianqing, Yuan Liao, and Weichen Wang, 2016, Projected principal component analysis in factor models, *Annals of statistics* 44, 219.

Feng, Guanhao, Stefano Giglio, and Dacheng Xiu, 2017, Taming the factor zoo, Technical report, University of Chicago.

Feng, Guanhao, Nicholas G. Polson, and Jianeng Xu, 2019, Deep learning in asset pricing, Technical report, University of Chicago.

Freyberger, Joachim, Andreas Neuhierl, and Michael Weber, 2017, Dissecting characteristics nonparametrically, Technical report, University of Wisconsin-Madison.

Gagliardini, Patrick, Elisa Ossola, and Olivier Scaillet, 2016, Time-varying risk premium in large cross-sectional equity datasets, *Econometrica* 84, 985–1046.

Gallinari, Patrick, Yann LeCun, Sylvie Thiria, and Francoise Fogelman-Soulie, 1987, Memoires associatives distribuees, *Proceedings of COGNITIVA* 87, 93.

Giglio, Stefano W, and Dacheng Xiu, 2018, Asset pricing with omitted factors, Technical report, University of Chicago.

Goodfellow, Ian, Yoshua Bengio, and Aaron Courville, 2016, *Deep Learning* (MIT Press), http://www.deeplearningbook.org.

Gu, Shihao, Bryan Kelly, and Dacheng Xiu, 2018, Empirical asset pricing via machine learning, Technical report, University of Chicago.

Hallin, Marc, and Roman Liška, 2007, Determining the number of factors in the general dynamic factor model, *Journal of the American Statistical Association* 102, 603–617.

He, Zhiguo, and Arvind Krishnamurthy, 2013, Intermediary asset pricing, *American Economic Review* 103, 732–70.

Hinton, Geoffrey E, and Ruslan R Salakhutdinov, 2006, Reducing the dimensionality of data with neural networks, *science* 313, 504–507.

Hinton, Geoffrey E, and Richard S Zemel, 1994, Autoencoders, minimum description length and helmholtz free energy, in *Advances in neural information processing systems*, 3–10.

Ioffe, Sergey, and Christian Szegedy, 2015, Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift, *International Conference on Machine Learning* 448–456.

Kapetanios, George, 2010, A testing procedure for determining the number of factors in approximate factor models, *Journal of Business and Economic Statistics* 28, 397–409.

Kelly, Bryan, and Seth Pruitt, 2015, The three-pass regression filter: A new approach to forecasting using many predictors, *Journal of Econometrics* 186, 294–316.

Kelly, Bryan, Seth Pruitt, and Yinan Su, 2017, Some characteristics are risk exposures, and the rest are irrelevant, Technical report, University of Chicago.

Kingma, Diederik, and Jimmy Ba, 2014, Adam: A method for stochastic optimization, *arXiv preprint arXiv:1412.6980* .

Kozak, Serhiy, 2019, Kernel trick for the cross section, *SSRN Working Paper* .

Kozak, Serhiy, Stefan Nagel, and Shrihari Santosh, 2017, Shrinking the cross section, Technical report, University of Michigan.

Kozak, Serhiy, Stefan Nagel, and Shrihari Santosh, 2018, Interpreting factor models, *Journal of Finance* 73, 1183–1223.

Moon, Hyungsik Roger, and Martin Weidner, 2018, Nuclear norm regularized estimation of panel regression models, Technical report, University of Southern California.

Onatski, Alexei, 2010, Determining the number of factors from empirical distribution of eigenvalues, *Review of Economics and Statistics* 92, 1004–1016.

Ross, Stephen A, 1976, The arbitrage theory of capital asset pricing, *Journal of economic theory* 13, 341–360.

Santos, Tano, and Pietro Veronesi, 2004, Conditional betas, Technical report, National Bureau of Economic Research.

Stock, James H, and Mark W Watson, 2002, Macroeconomic forecasting using diffusion indexes, *Journal of Business & Economic Statistics* 20, 147–162.

Appendix

A Mathematical Proofs

A.1 Notation

We use $(A : B)$ to denote the column concatenation of two matrices A and B. The vector e_i has a value of 1 in the i^{th} position and 0 elsewhere (and whose dimension implicitly conforms to the context). Likewise, the vector ι denotes a conformable vector with all entries being 1. For any time series of vectors $\{a_t\}_{t=1}^T$, we denote $\bar{a} = \frac{1}{T}\sum_{t=1}^T a_t$. In addition, we write $\bar{a}_t = a_t - \bar{a}$. A denotes the matrix $(a_1 : a_2 : \ldots : a_T)$, and $\bar{A} = A - \bar{a}\iota'$ correspondingly. We use $\lambda_j(A)$ to denote the jth largest eigenvalue of A, and $\sigma_j(A)$ the jth largest singular value. We use $\|A\|$ and $\|A\|_{\mathrm{F}}$ to denote the operator norm (or $\mathbb{L}_2$ norm) and the Frobenius norm of a matrix $A = (a_{ij})$, that is, $\sqrt{\lambda_1(A'A)}$ and $\sqrt{\mathrm{Tr}(A'A)}$, respectively.

A.2 Proofs

Proof of Proposition 1. At first, we set the partial derivative with respect to $b^{(1)}$ to zero.

$$\begin{aligned}
\frac{\partial}{\partial b^{(1)}} \left\| R - \left(b^{(1)}\iota' + W^{(1)}(b^{(0)}\iota' + W^{(0)}R) \right) \right\|_{\mathrm{F}}^2 &= 0 \\
\left(R - b^{(1)}\iota' - W^{(1)}(b^{(0)}\iota' + W^{(0)}R) \right)\iota &= 0 \\
\widehat{b}^{(1)} = \frac{1}{T}\left(R\iota - TW^{(1)}b^{(0)} - W^{(1)}W^{(0)}R\iota \right).
\end{aligned}$$

Then we insert the solution into (8).

$$\begin{aligned}
&\min_{b,W} \left\| R - \left(b^{(1)}\iota' + W^{(1)}(b^{(0)}\iota' + W^{(0)}R) \right) \right\|_{\mathrm{F}}^2 \\
=&\min_{W} \left\| \left(R - \frac{1}{T}R\iota\iota' \right) - W^{(1)}W^{(0)} \left(R - \frac{1}{T}R\iota\iota' \right) \right\|_{\mathrm{F}}^2 \\
=&\min_{W} \left\| \bar{R} - W^{(1)}W^{(0)}\bar{R} \right\|_{\mathrm{F}}^2,
\end{aligned}$$

where $\bar{R}$ is a matrix of demeaned returns. Thus, the problem becomes independent of the bias terms. We focus on the weights $W^{(0)}$and $W^{(1)}$.
Next, we set the partial derivative with respect to $W^{(0)}$ to zero and assume $\bar{R}\bar{R}'$ and $W^{(1)\prime}W^{(1)}$ both have full rank.

$$\begin{aligned}
&\frac{\partial}{\partial W^{(0)}} \left\| \bar{R} - W^{(1)}W^{(0)}\bar{R} \right\|_{\mathrm{F}}^2 = 0 \\
&W'^{(1)}W^{(1)}W'^{(0)}\bar{R}\bar{R}' - W'^{(1)}\bar{R}\bar{R}' = 0 \\
&W^{(0)} = (W'^{(1)}W^{(1)})^{-1}W'^{(1)}.
\end{aligned}$$

So, the optimization problem becomes

$$\begin{aligned}
&\min_{W} \left\| \bar{R} - W^{(1)} W^{(0)} \bar{R} \right\|_{\mathrm{F}}^2 \\
=&\min_{W^{(1)}} \left\| \bar{R} - W^{(1)} \left(W'^{(1)} W^{(1)} \right)^{-1} W'^{(1)} \bar{R} \right\|_{\mathrm{F}}^2 \\
=&\min_{W^{(1)}} \left\| \bar{R} - \mathbb{P}_{W^{(1)}} \bar{R} \right\|_{\mathrm{F}}^2 .
\end{aligned}$$

The Eckart-Young-Mirsky theorem for the Frobenius norm states that the best rank K approximation for a matrix is $\sum_{i=1}^{K} s_i p_i q_i' = P \Lambda Q$. Therefore, $W^{(1)}$ is the solution if and only if it satisfies

$$\mathbb{P}_{W^{(1)}} \bar{R} = P \Lambda Q \tag{A.1}$$

It is obvious that $W^{(1)} = P$ is one solution for the above equation, because

$$\mathbb{P}_{W^{(1)}} \bar{R} = P(P'P)^{-1} P' \bar{R} = P(P'P)^{-1} P'(P \Lambda Q + \widehat{U}) = P \Lambda Q.$$

However, the solution of (A.1) is not unique. $W^{(1)} = PA$ is also a solution, where A is any $K \times K$ full-rank matrix. The linear autoencoder is equivalent to PCA since they have the same factor loading matrix P (up to a rotation matrix). □

Proof of Proposition 2. At first, we consider the objective function of IPCA when $Z_{t-1}' Z_{t-1} = \Sigma$. The first-order condition for F is given by

$$\widehat{f}_t \;=\; (\Gamma Z_{t-1}' Z_{t-1} \Gamma')^{-1} \Gamma Z_{t-1}' r_t = (\Gamma \Sigma \Gamma')^{-1} \Gamma Z_{t-1}' r_t.$$

Then we plug in $\widehat{f}_t$ to the objective function of IPCA (17)

$$\min_{\Gamma, F} \sum_{t=1}^{T} \left\| r_t - Z_{t-1} \Gamma' f_t \right\|^2 = \min_{\Gamma} \sum_{t=1}^{T} \left\| r_t - Z_{t-1} \Gamma' (\Gamma \Sigma \Gamma')^{-1} \Gamma Z_{t-1}' r_t \right\|^2 . \tag{A.2}$$

For the two-sided Autoencoder model, we use the managed portfolios $x_t = (Z_{t-1}' Z_{t-1})^{-1} Z_{t-1}' r_t$ as the inputs of the right-hand side. We can rewrite the objective function (18) as

$$\min_{W_0, W_1} \sum_{t=1}^{T} \left\| r_t - Z_{t-1} W_0' W_1 x_t \right\|^2 = \arg\min_{W_0, W_1} \sum_{t=1}^{T} \left\| r_t - (x_t' \otimes Z_{t-1} W_0') \mathrm{vec}(W_1) \right\|^2 . \tag{A.3}$$

Next the first order condition of $\mathrm{vec}(W)$ is

$$\begin{aligned}
\mathrm{vec}(W_1) \;&=\; \left(\sum_{t=1}^{T} (x_t' \otimes Z_{t-1} W_0')' (x_t' \otimes Z_{t-1} W_0') \right)^{-1} \left(\sum_{t=1}^{T} (x_t' \otimes Z_{t-1} W_0')' r_t \right) \\
&=\; \left(\sum_{t=1}^{T} x_t x_t' \otimes W_0 Z_{t-1}' Z_{t-1} W_0' \right)^{-1} \left(\sum_{t=1}^{T} \left(x_t \otimes W_0 Z_{t-1}' r_t \right) \right) .
\end{aligned}$$

We plug in $Z_{t-1}'Z_{t-1} = \Sigma$,

$$\begin{aligned}
\text{vec}(W_1) &= \left(\sum_{t=1}^{T} x_t x_t' \otimes W_0 \Sigma W_0'\right)^{-1} \left(\sum_{t=1}^{T} \left(x_t \otimes W_0 Z_{t-1}' r_t\right)\right) \\
&= \left(\left(\sum_{t=1}^{T} x_t x_t'\right)^{-1} \otimes (W_0 \Sigma W_0')^{-1}\right) \left(\sum_{t=1}^{T} \left(x_t \otimes W_0 Z_{t-1}' r_t\right)\right) \\
&= \sum_{t=1}^{T} \left(\left(\sum_{t=1}^{T} x_t x_t'\right)^{-1} x_t \otimes (W_0 \Sigma W_0')^{-1} W_0 Z_{t-1}' r_t\right).
\end{aligned}$$

Then we recover W_1 from $\text{vec}(W_1)$

$$\begin{aligned}
W_1 &= \sum_{t=1}^{T} \left[(W_0 \Sigma W_0')^{-1} W_0 Z_{t-1}' r_t x_t' \left(\sum_{t=1}^{T} x_t x_t'\right)^{-1}\right] \\
&= \sum_{t=1}^{T} \left[(W_0 \Sigma W_0')^{-1} W_0 \Sigma x_t x_t' \left(\sum_{t=1}^{T} x_t x_t'\right)^{-1}\right] \\
&= (W_0 \Sigma W_0')^{-1} W_0 \Sigma.
\end{aligned}$$

We can plug in the solution of W_1 to the objective function of the two-sided Autoencoder (18),

$$\min_{W_0, W_1} \sum_{t=1}^{T} \left\| r_t - Z_{t-1} W_0' W_1 x_t \right\|^2 = \arg\min_{W_0} \sum_{t=1}^{T} \left\| r_t - Z_{t-1} W_0' (W_0 \Sigma W_0')^{-1} W_0 \Sigma x_t \right\|^2. \tag{A.4}$$

We see IPCA and the two-sided Autoencoder have the same objective functions A.2 and A.4. The Autoencoder solution and the IPCA solution are identical with $\Gamma = W_0$. They give us identical factor estimates $\hat{f}_t$ and factor loading estimates $\hat{\beta}_{i,t-1}$.

□

B Algorithms

Algorithm 1: Early Stopping

Initialize $j = 0$, $\epsilon = \infty$ and select the patience parameter p.
while $j < p$ **do**
  Update θ using the training algorithm (e.g., the steps inside the while loop of Algorithm 2 for h steps).
  Calculate the prediction error from the validation sample, denoted as ϵ'.
  if $\epsilon' < \epsilon$ **then**
    $j \leftarrow 0$.
    $\epsilon \leftarrow \epsilon'$.
    $\theta' \leftarrow \theta$.
    else
    $j \leftarrow j + 1$.
  end
end
Result: The final parameter estimate is θ'.

Algorithm 2: Adam for Stochastic Gradient Descent (SGD)

Initialize the target parameter vector θ_0.
Set $m_0 = 0, v_0 = 0, t = 0, \beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 1e-8$.
Tuning parameters: learning rate α, batch size b.
while θ_t *not converged* **do**
  $t \leftarrow t + 1$.
  $g_t \leftarrow \nabla_\theta \left[\frac{1}{b}\sum_{s \in B_t} \mathcal{L}(\theta; s)\right]\big|_{\theta=\theta_{t-1}}$, where B_t is the set of batch subsamples.
  $m_t \leftarrow \beta_1 m_{t-1} + (1-\beta_1) g_t$.
  $v_t \leftarrow \beta_2 v_{t-1} + (1-\beta_2) g_t \odot g_t$.[15]
  $\hat{m}_t \leftarrow m_t / (1-(\beta_1)^t)$.
  $\hat{v}_t \leftarrow v_t / (1-(\beta_2)^t)$.
  $\theta_t \leftarrow \theta_{t-1} - \alpha \hat{m}_t \oslash (\sqrt{\hat{v}_t} + \epsilon)$.
end
Result: The final parameter estimate is θ_t.

Algorithm 3: Batch Normalization (for one Activation over one Batch)

Input: Values of x for each activation over a batch $\mathcal{B} = \{x_1, x_2, \ldots, x_N\}$.

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{N}\sum_{i=1}^{N} x_i$$

$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{N}\sum_{i=1}^{N}(x_i - \mu_{\mathcal{B}})^2$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}}$$

$$y_i \leftarrow \gamma \widehat{x}_i + \beta := \mathrm{BN}_{\gamma,\beta}(x_i)$$

Result: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i) : i = 1, 2 \ldots, N\}$.