

Nonlinear IPCA with Precision Weighting

Common Task Framework Submission

Szymon Kubiak*

City St George's, University of London, London, UK

Tillman Weyde

City St George's, University of London, London, UK

Oleksandr Galkin

City St George's, University of London, London, UK

Dan Philps

University of Warwick, Coventry, UK

City St George's, University of London, London, UK

Ram Gopal

University of Warwick, Coventry, UK

July 19, 2026

1 Overview

This submission extends Instrumented Principal Components Analysis (IPCA; Kelly et al. 2019), a factor model in which a stock's factor exposures are a linear function of its observable characteristics. We change this in two ways: the linear map from characteristics to exposures is replaced by a deep neural network, and a second network estimates each stock's idiosyncratic return variance from the same characteristics. The two networks are trained one after the other, and their outputs are combined into a mean-variance portfolio through a two-stage construction: a factor-space allocation, followed by a precision-weighted projection to individual stock weights in which the variance estimates control how much weight each stock receives.

2 Background

IPCA. IPCA (Kelly et al., 2019) is a latent factor model in which a stock's factor loadings are a linear function of its observable characteristics. Writing $c_{i,t}$ for the vector of characteristics of stock i at the end of month t , its loading vector is

$$\beta_{i,t} = \Gamma^\top c_{i,t},$$

where Γ is a single coefficient matrix shared across all stocks and dates. Returns then follow

$$r_{i,t+1} = \beta_{i,t}^\top f_{t+1} + \varepsilon_{i,t+1} = (\Gamma^\top c_{i,t})^\top f_{t+1} + \varepsilon_{i,t+1},$$

where f_{t+1} is the vector of latent factor returns. Given Γ , the factor returns in each month are recovered by a cross-sectional regression of returns on loadings; Γ and the factor series f_t

*szymon.kubiak@citystgeorges.ac.uk

are estimated together by alternating between updating the factors and updating Γ until they converge.

Nonlinear factor models. The nonlinear loading map places this model within a line of work applying machine learning to cross-sectional return prediction. Gu et al. (2020) find that deep networks outperform linear baselines out of sample, and Chen et al. (2024) and Didisheim et al. (2023) build deep latent factor models with nonlinear exposures. A linear loading map forces every characteristic to have the same marginal effect on a stock’s exposure regardless of its other characteristics; a network relaxes this, letting the map bend and letting characteristics interact. Our model otherwise stays deliberately close to IPCA - factors are still recovered by cross-sectional regression, loadings are still driven by characteristics, and no per-stock parameters are estimated - so it can be read as a direct nonlinear counterpart of the original.

Idiosyncratic variance. Stocks differ in how much of their return is idiosyncratic, and this dispersion is related to observable characteristics such as size and volatility. Factor models often leave this variation unmodelled or handle it only indirectly through the covariance of estimated residuals. We address this by estimating each stock’s idiosyncratic variance as an explicit function of its characteristics, so that the variance can be predicted for any stock from its features and used directly when sizing positions: in the mean-variance portfolio it acts as a precision weight, and among stocks with similar expected returns those with lower estimated variance receive larger positions.

3 Model specification

3.1 Setup and notation

We work with a panel of stocks observed monthly. At the end of each month t , every stock i has a vector of observable characteristics, and we observe the return it goes on to earn over the following month. The goal is to build, at the end of each month, a portfolio of these stocks with a high Sharpe ratio over the following month.

The symbols used throughout are collected here for reference.

Symbol	Meaning
i, t	stock index and month index
n_t	number of stocks in the cross-section in month t
K	number of latent factors
D	number of characteristics per stock
$c_{i,t} \in \mathbb{R}^D$	characteristic vector of stock i at end of month t
$r_{i,t+1} \in \mathbb{R}$	excess return of stock i over month $t \rightarrow t+1$
$h_{i,t} \in \mathbb{R}^K$	factor loadings of stock i (its factor exposures)
$f_{t+1} \in \mathbb{R}^K$	latent factor returns over month $t \rightarrow t+1$
$\sigma_{i,t}^2$	idiosyncratic (stock-specific) return variance
$H_t \in \mathbb{R}^{n_t \times K}$	the loadings of all stocks in month t , stacked in a matrix
$r_{t+1} \in \mathbb{R}^{n_t}$	the returns of all stocks in month t , stacked in a vector
$\bar{f}, \Sigma_f$	long-run mean and covariance of the factor returns
$w_t \in \mathbb{R}^{n_t}$	portfolio weights assigned in month t

3.2 Return generating process

We assume returns are driven by a set of common factors plus stock-specific noise. Each stock’s next-month return is its factor exposures times the factor returns, plus a mean-zero error whose

size can differ across stocks:

$$r_{i,t+1} = \underbrace{h_{i,t}^\top f_{t+1}}_{\text{common factor part}} + \underbrace{\varepsilon_{i,t+1}}_{\text{stock-specific noise}}, \quad \varepsilon_{i,t+1} \sim \mathcal{N}(0, \sigma_{i,t}^2).$$

Here the loadings $h_{i,t}$ say how strongly stock i moves with each of the K factors, and f_{t+1} are the (unobserved) factor returns realized over the next month. The noise variance $\sigma_{i,t}^2$ is allowed to differ from stock to stock, rather than being assumed equal across the cross-section.

In this model, both the loadings and the noise variance are functions of the stock's characteristics, learned by two neural networks:

$$h_{i,t} = \text{LoadingNet}_\theta(c_{i,t}), \quad \log \sigma_{i,t}^2 = \text{VarianceNet}_\phi(c_{i,t}),$$

where θ and ϕ are the trainable weights of the two networks. Modelling loadings as a function of characteristics (rather than estimating a separate loading per stock) is the defining idea of IPCA; the contribution here is to make that function nonlinear, and to add a second network for the variance.

The factor returns f_{t+1} are not produced by a network. Instead, once the loadings are fixed, each month's factor return is read off from that month's cross-section by ordinary least squares - a simple linear regression of the stocks' realized returns on their loadings (detailed in Section 5.1). This mirrors standard IPCA, where the factors are likewise recovered by cross-sectional regression.

Like any latent factor model, the loadings and factors are identified only up to a rotation: they can be transformed together, in offsetting ways, without changing any fitted return - so the data alone cannot single out one version. IPCA removes this ambiguity by imposing a normalization on its loading map. We do not, because we never need it: rotating the loadings and factor moments together leaves the max-Sharpe portfolio weights of Section 7 exactly unchanged, so any version of the factors gives the same portfolio. The specific construction we actually use (also Section 7) is only approximately invariant to such rotations, but the dependence is weak and plays no role in anything downstream.

4 Network architecture

4.1 Shared backbone

The two networks have the same structure, differing only in the size of their final output layer. Each takes a stock's D -dimensional characteristic vector as input and passes it through the following stages. An input layer first projects the D characteristics up to a hidden representation of width 256. This is followed by five residual blocks, each of which applies a linear layer, a SiLU nonlinearity (Elfwing et al., 2018), and dropout, and then adds the block's input back to its output. Writing x_ℓ for the input to block ℓ , the block computes

$$x_{\ell+1} = \text{SiLU}(\text{Linear}(x_\ell)) + x_\ell,$$

where the added $+ x_\ell$ is the "residual" or skip connection and SiLU is the activation function. A final linear layer maps the width-256 representation to the network's output: a K -dimensional vector for the loading network, or a single number for the variance network. Counting the input projection and the output layer, each network is seven layers deep.

4.2 LoadingNet

The loading network outputs the K -dimensional loading vector $h_{i,t}$ for each stock, and then subtracts the cross-sectional mean of the loadings across that month's stocks:

$$h_{i,t} \leftarrow h_{i,t} - \frac{1}{n_t} \sum_{j=1}^{n_t} h_{j,t},$$

so that the loadings sum to zero across the month's stocks. The centering is part of the network's computation and is applied every time the network is used, both in training and when generating portfolio weights.

4.3 VarianceNet

The variance network is identical to the loading network except that its final layer outputs a single number, the log-variance $\log \sigma_{i,t}^2$ of the stock's idiosyncratic noise. Working in the log domain means the variance, recovered as $\sigma_{i,t}^2 = \exp(\cdot)$, is always positive whatever the network outputs.

5 Training methodology

The two networks are trained one after the other, not together. First the loading network is trained on its own; then it is frozen and the variance network is trained on the leftover errors.

5.1 Stage 1 - LoadingNet

Stage 1 teaches the loading network to produce loadings that explain returns well. The logic each month is: given the loadings the network currently produces, find the factor returns that best fit that month's actual stock returns, and then measure how large the leftover fitting errors are. The network is nudged to make those errors small.

In detail, for each month t in the training window:

1. Pass every stock's characteristics through the network to get the loading matrix H_t (one row of loadings per stock).
2. Find the factor returns $\hat{f}_{t+1}$ that best explain this month's realized stock returns r_{t+1} , by regressing returns on loadings:

$$\hat{f}_{t+1} = (H_t^\top H_t + \lambda I)^{-1} H_t^\top r_{t+1}.$$

This is the ordinary-least-squares formula for regressing r_{t+1} on H_t ; the small term λI (with λ tiny) keeps the matrix invertible.

3. Compute the fitting errors - the part of returns the factors could not explain:

$$\hat{\varepsilon}_{t+1} = H_t \hat{f}_{t+1} - r_{t+1}.$$

4. Use the average squared error as the loss for that month, $\mathcal{L}_t = n_t^{-1} \|\hat{\varepsilon}_{t+1}\|^2$.

This is the same loss used by linear IPCA, with one difference: the loadings here come from a nonlinear network rather than a linear map.

The four steps above define the loss for a single month. Training repeats them across the whole training window many times: one epoch is one pass through all training months, and the network

is trained for 50 such epochs. Within each epoch the months are visited in a fresh random order, so the network does not see them in the same sequence every time. The per-month losses are not applied one at a time; instead the gradients from 12 consecutive months are accumulated and the network weights are updated once per group. Each month's loss is already an average over its cross-section, so this grouping is not about batch size; it gives each update a fixed one-year calendar span and averages over the month-to-month variation in the gradient, making the updates less noisy. The factor returns $\hat{f}_{t+1}$ are computed by a regression rather than output by the network, but that regression is differentiable in the loadings, so gradients still flow back into the network weights through it.

The optimizer is AdamW, with a learning rate of 3×10^{-4} and weight decay of 10^{-4} . The learning rate is decreased smoothly to zero over the 50 epochs following a cosine schedule, which helps the network settle into a good solution toward the end of training. The variance network in Stage 2 is trained with all the same settings.

5.2 Stage 2 - VarianceNet

Once Stage 1 is done, the loading network is frozen and we record each stock's squared fitting error from that month's regression:

$$\hat{\varepsilon}_{i,t+1}^2 = (h_{i,t}^\top \hat{f}_{t+1} - r_{i,t+1})^2.$$

These squared errors are what the variance network learns to predict: a stock whose returns are hard to explain (large errors) should be assigned a large idiosyncratic variance, and vice versa. The variance network is trained by minimizing the Gaussian negative log-likelihood,

$$\mathcal{L}_\phi = \frac{1}{N} \sum_{i,t} \left[\underbrace{\log \sigma_{i,t}^2}_{\text{penalizes over-estimating}} + \underbrace{\hat{\varepsilon}_{i,t+1}^2 / \sigma_{i,t}^2}_{\text{penalizes under-estimating}} \right],$$

where $\sigma_{i,t}^2 = \exp(\text{VarianceNet}_\phi(c_{i,t}))$ and N is the total number of stock-months. The two terms pull in opposite directions: the first grows if the predicted variance is too large, the second if it is too small relative to the squared error. At the minimum, the predicted variance tracks the size of the residuals from Stage 1.

6 Factor estimation

After Stage 1 training, we compute the factor return for every month in the training window using the same regression as in training,

$$\hat{f}_{t+1} = (H_t^\top H_t + \lambda I)^{-1} H_t^\top r_{t+1},$$

which gives a time series of K -dimensional factor return vectors $\hat{f}_t$. From this series we estimate the average factor return $\bar{f}$ and the factor covariance matrix Σ_f :

$$\bar{f} = \frac{1}{T} \sum_{t=1}^T \hat{f}_t, \quad \Sigma_f = \frac{1}{T-1} \sum_{t=1}^T (\hat{f}_t - \bar{f})(\hat{f}_t - \bar{f})^\top + \lambda I_K,$$

where T is the number of training months and the small λI_K term again just keeps the matrix safely invertible. These two quantities, $\bar{f}$ and Σ_f , summarize the factors and feed directly into the portfolio formula.

Note that these factor estimates use plain (unweighted) regression: the per-stock variances from the variance network are not used here. They enter only at the portfolio step (Section 7).

7 Portfolio construction

The portfolio weights are built from a mean-variance objective: the optimal weights are proportional to $\Sigma^{-1}\mu$, where μ is the vector of expected stock returns and Σ their covariance matrix, and we form these two inputs from the model's own estimates.

From the factor model, the expected return of the stocks and their covariance are

$$\mu_{t+1} = H_t \bar{f}, \quad \Sigma_{t+1} = H_t \Sigma_f H_t^\top + \Omega_t^{-1}.$$

The expected return of each stock is its loadings times the average factor returns. The covariance has two parts: a common part $H_t \Sigma_f H_t^\top$ coming from shared factor exposure, and a stock-specific part Ω_t^{-1} , where $\Omega_t = \text{diag}(\sigma_{1,t}^{-2}, \dots, \sigma_{n_t,t}^{-2})$ is the diagonal matrix of the variance network's precisions (a precision is one over a variance). A stock with low idiosyncratic variance has high precision.

Applying the Sherman–Morrison–Woodbury identity to this covariance (a low-rank factor term plus a diagonal) gives the mean-variance-optimal weights in closed form, requiring only a $K \times K$ inverse rather than an $n_t \times n_t$ one:

$$w_t \propto \Omega_t H_t (H_t^\top \Omega_t H_t + \Sigma_f^{-1})^{-1} \Sigma_f^{-1} \bar{f}.$$

Reading this from right to left: $\Sigma_f^{-1} \bar{f}$ is the mean-variance-optimal position in factor space; the middle $K \times K$ inverse rescales it for how the loadings and precisions interact; H_t turns that factor-space quantity into per-stock tilts through each stock's loadings; and the leading Ω_t scales each stock by its precision, so lower-noise stocks get proportionally larger weight. This last step is where the variance network enters the allocation.

The portfolio is built from a variant of this formula that separates two roles the factor covariance plays in it. In the exact formula, Σ_f^{-1} appears twice: once forming the factor-space target $\Sigma_f^{-1} \bar{f}$, and once inside the bracket, where it shapes the geometry of the projection onto stocks. We keep the first and drop the second, replacing Σ_f^{-1} in the bracket with a small ridge λI . The weights then solve $(H_t^\top \Omega_t H_t + \lambda I) x = \Sigma_f^{-1} \bar{f}$ and set $w_t \propto \Omega_t H_t x$. Equivalently, this is a two-stage construction: first choose a factor-space allocation $\phi = \Sigma_f^{-1} \bar{f}$, then map it to stock weights by a precision-weighted projection $w_t \propto \Omega_t H_t (H_t^\top \Omega_t H_t)^{-1} \phi$. The estimated factor covariance thus shapes only what the portfolio targets in factor space, not the geometry by which that target is spread across stocks.

Separating the two roles this way regularizes the construction. The estimated factor covariance Σ_f is noisy, and inverting it amplifies error in its smallest eigenvalues, so Σ_f^{-1} is the noisiest ingredient in the weight formula. The exact formula uses it twice - to set the factor-space target and to shape the projection onto stocks - and each use exposes the weights to that noise. The two-stage form keeps Σ_f^{-1} where it is most informative, in the target $\Sigma_f^{-1} \bar{f}$ that sets the direction of the factor bet, and drops it from the projection, replacing it with a ridge. Between this construction and the exact formula, we chose the one that performed better in pre-1990 walk-forward testing.

The resulting weights are rescaled so their absolute values sum to one, $w_t = w_t^{\text{raw}} / \sum_i |w_{i,t}^{\text{raw}}|$, fixing the portfolio's gross exposure at one dollar.

8 Data preprocessing

Within each monthly cross-section, every characteristic is rank-transformed: non-missing values are ranked (with ties taking the maximum rank) and divided by the cross-section size to give a

percentile in $[0, 1]$, which is then shifted to $[-0.5, 0.5]$ by subtracting 0.5. Exact zeros and missing entries are both mapped to the median rank of 0.5, i.e. treated as neutral. No winsorization or feature selection is applied; all features in `ctff_features` enter both networks. The daily-return series provided with the dataset is not used by this model.

9 Walk-forward protocol

The model is refit once per year on an expanding window. For each year T , the training window is all months up to the last month-end before the first month of T ; the two networks and the factor moments $(\bar{f}, \Sigma_f)$ are estimated on this window from scratch, and portfolio weights are then generated for each month of T .

All training and inference was performed on a single NVIDIA GeForce RTX 5090 (32GB VRAM).

10 Hyperparameters and selection

The table below lists the full configuration.

Parameter	Value	Notes
Latent factors K	32	Output dimension of LoadingNet
Hidden width	256	Both networks
Depth	7 layers	Input projection + 5 residual blocks + output head
Activation	SiLU	
Dropout	0.1	Between linear and activation in each block
Optimizer	AdamW	Decoupled weight decay
Learning rate	3×10^{-4}	Both stages
Weight decay	10^{-4}	Both stages
Epochs per stage	50	Stage 1 and Stage 2 each
LR schedule	Cosine annealing	Single cycle per stage
Gradient accumulation	12 months	One optimizer step per 12 cross-sections

Table 1: Hyperparameter configuration.

These were selected using the pre-1990 training period. For each candidate configuration we ran the same annual walk-forward procedure described above and compared configurations by their walk-forward Sharpe over that period. The search was not a systematic grid; different combinations were tried in an exploratory fashion.

Several patterns emerged from this exercise.

- **Depth and width.** Shallower networks and narrower hidden dimensions were tested and performed worse than the reported configuration.
- **Number of epochs.** Training for longer than 50 epochs per stage generally reduced performance, though the effect was not fully consistent across variants. 50 epochs was chosen as a balance between reliable convergence and avoiding this deterioration.
- **Dropout.** Moderate dropout (around 0.1) slightly improved performance; larger values consistently made results worse.
- **Number of factors.** $K = 32$ is larger than typical in the linear IPCA literature (which usually reports diminishing returns past $K \approx 5$ to 6). In this model, performance kept improving as K was increased, and $K = 32$ was selected on that basis. A plausible reading

is that the nonlinear loading map spans a richer set of exposures than a linear map, so more latent factors carry usable structure before noise dominates; we did not investigate this beyond the walk-forward comparison.

- **Training window.** Both expanding windows and 120-month rolling windows were tested. The expanding window performed better and was adopted.

References

- Luyang Chen, Markus Pelger, and Jason Zhu. Deep learning in asset pricing. *Management Science*, 70(2):714–750, 2024.
- Antoine Didisheim, Shikun Barry Ke, Bryan T Kelly, and Semyon Malamud. Complexity in factor pricing models. Technical report, National Bureau of Economic Research, 2023.
- Stefan Elfving, Eiji Uchibe, and Kenji Doya. Sigmoid-weighted linear units for neural network function approximation in reinforcement learning. *Neural networks*, 107:3–11, 2018.
- Shihao Gu, Bryan Kelly, and Dacheng Xiu. Empirical asset pricing via machine learning. *The Review of Financial Studies*, 33(5):2223–2273, 2020.
- Bryan T Kelly, Seth Pruitt, and Yinan Su. Characteristics are covariances: A unified model of risk and return. *Journal of Financial Economics*, 134(3):501–524, 2019.