Machine Learning in the Cross Section of Asset Returns

Doron Avramov

Updated: October 16, 2024

Machine learning Methods in asset pricing

- Machine learning offers a vast collection of high-dimensional models designed to predict, explain, or classify quantities of interest, while imposing regularization to prevent overfitting.
- In empirical finance, machine learning models have been deployed to predict returns, estimate the stochastic discount factor (SDF), extract common factors, and test asset pricing models.
- There are many other applications including the study of mutual funds.
- Machine learning applies to any asset class, including equities, fixed-income securities, currencies, and cryptocurrencies.
- This set of notes comprehensively covers the architecture of various routines and their broad implementations in empirical asset pricing.
- The routines apply to beta-pricing (IPCA, CA) and pricing kernel (Ridge, GAN) representations.
- The routines can also apply to reduced-form settings (Ridge, LASSO, NN, LSTM, TE, RL), whereas the return-generating process does not conform to any structural representation.
- IPCA, CA, Ridge, GAN, LASSO, NN, LSTM, TE, and RL are all abbreviations for machine learning routines to be explained in detail throughout the notes.
- I will also discuss how ChatGPT works, building on concepts from stock return prediction.

Asset pricing models

- We start with a brief review of asset pricing settings.
- In unconditional asset pricing, stock return moments are fixed.
- That is, all model parameters are time-invariant.
- Conditional models formulate time variation.
- There are different ways to model time-varying moments.
- First, risk and risk premia can vary with macro variables (e.g., the dividend-to-price ratio) or firm characteristics (e.g., size, book-to-market).
- Second, risk and risk premia can follow a latent ARMA process, e.g., AR(1).
- Third, time variation can be captured through high-frequency data in rolling samples.
- Theoretically, beta pricing and pricing kernel representations are equivalent.
- Empirical tests, however, could be different.

Beta pricing representation

- In beta pricing settings, the expected excess return of asset *i* at time *t* is given by

$$\mathbb{E}\left(r_{i,t}^{e}\right) = \alpha_i + \beta_i{}'\mathbb{E}(f_t)$$

  where f_t denotes a set of K portfolio spreads, β_i is a K vector of factor loadings, and α_i reflects the expected return component unexplained by factors, or model mispricing.
- The aim is to identify economic (ICAPM) or statistical (APT) factors that eliminate mispricing.
- Absent alpha, the expected return differential across assets is triggered by factor loadings only.
- The presence of model mispricing could give rise to additional cross-sectional effects.
- The alpha-beta debate: does an anomaly represent a risk factor or instead mispricing?
- An anomaly can also reflect false discovery; hence, t-ratio thresholds should be higher.

Pricing kernel representation

- The pricing kernel representation for asset pricing can be formulated through the Hansen-Jagannathan equation:

$$M_t = 1 - b'(r_t - \mu),$$

where μ is an N–vector of expected return and r_t is an N-vector of realized returns.

- The unobservable pricing kernel is projected on the space of demeaned returns.
- Identification of the projection slopes is feasible through the first-order conditions.
- FOCs are also known as the Euler Equation, $E(M_t\ r_t)=\mathbf{1}$, where $\mathbf{1}$ is an N–vector of ones.
- There are three plausible formulations for the N-vector of slope coefficients.

Pricing kernel parameters

- First, b can be constant – amounting to unconditional models.
- Moreover, b can also vary with macro conditions or firm characteristics.
- To illustrate, consider variation with firm characteristics.
- We have $b_{t-1} = C_{t-1}d$, where C_{t-1} is an $N \times H$ matrix, H characteristics (e.g., size, profitability, past returns) for each of the N stocks, and d is an $H\times 1$ vector.
- Plugging b_{t-1} into the pricing kernel yields

$$M_t = 1 - d'[C'_{t-1}r_t - E_{t-1}(C'_{t-1}r_t)].$$

- The quantity $C'_{t-1}r_t$ can be interpreted as H returns on managed portfolios.
- Conceptually, the pricing kernel is now projected on the space of H demeaned managed portfolio returns.

OLS

- Prior to delving into machine learning methods, we start, for perspective, with ordinary least squares (OLS).
- OLS is the best linear unbiased estimator of the regression coefficients.
- **BLUE=B**est **L**inear **U**nbiased **E**stimate.
- Regression errors do not have to be normal, nor do they have to be independently and identically distributed.
- But errors have to be zero mean, serially uncorrelated, as well as homoscedastic.
- In the presence of heteroskedasticity or autocorrelation, OLS is no longer BLUE.
- We can still use OLS estimators by finding heteroskedasticity-robust estimators of the variance, or we can devise an efficient estimator by re-weighting the data appropriately to incorporate heteroskedasticity.
- Similarly, with autocorrelation, we can find an autocorrelation-robust estimator of the variance, or we can devise an efficient estimator by re-weighting the data appropriately to account for autocorrelation

Is BLUE so promising?

- Requiring linearity is binding since nonlinear estimators do exist.
- This is where nonparametric Lasso and neural networks (NN) come to play.
- Likewise, requiring unbiasedness is crucial since biased estimators do exist.
- This is where shrinkage methods come into play: the OLS estimator's variance can be too large as OLS coefficients are unregulated.
- If judged by Mean Squared Error (MSE), alternative biased estimators could be more effective if they produce substantially smaller variance within the set of linear models.
- Recall that MSE = Variance + Biased Squared.
- Likewise, alternative non-linear estimators could be more effective within the set of unbiased models.
- The MSE of an OLS estimate is computed on the next page.

MSE of OLS

- Let β denote the true regression coefficients and let $\hat{\beta} = (X'X)^{-1}X'Y$, where X is a $T \times M$ matrix of de-meaned predictors and Y is a $T \times 1$ vector of the dependent variable.
- Both β and its estimate $\hat{\beta}$ are vectors of dimension M.
- The mean squared error (MSE) of the OLS estimate is given by

$$\begin{aligned}
\mathrm{MSE}(\hat{\beta}) &= E\left[(\hat{\beta}-\beta)'(\hat{\beta}-\beta)\right] \\
&= E\left\{\mathrm{tr}\left[(\hat{\beta}-\beta)'(\hat{\beta}-\beta)\right]\right\} \\
&= E\left\{\mathrm{tr}\left[(\hat{\beta}-\beta)(\hat{\beta}-\beta)'\right]\right\} \\
&= \mathrm{tr}\left\{E\left[(\hat{\beta}-\beta)(\hat{\beta}-\beta)'\right]\right\} \\
&= \mathrm{tr}[(X'X)^{-1}\,\sigma^2] \\
&= \sigma^2\mathrm{tr}[(X'X)^{-1}].
\end{aligned}$$

- When predictors are highly correlated, the matrix $X'X$ is ill-conditioned and the expression $\mathrm{tr}[(X'X)^{-1}]$ quickly explodes.

Shortcomings of OLS

- In the presence of many predictors, OLS delivers nonzero estimates for all coefficients – thus it is difficult to implement variable selection when the true data-generating process is sparse.
- Interpretation becomes challenging, as even insignificant coefficients contribute to the predicted value.
- The OLS solution is unique only if the design of X is full rank.
- The OLS does not handle potential nonlinearities and interactions between predictors.
- In summary, OLS is restrictive, often yields poor predictions, may overfit, does not penalize model complexity or large coefficients, and can be difficult to interpret.
- **Bayesian perspective**: one can introduce informed priors on regression coefficients to shrink slopes toward zero or values implied by economic theory or sound intuition.
- **Classical perspective**: shrinkage methods penalize complexity and impose regularization.
- Nonlinearities and interactions between predictors can also be accounted for.
- Such objectives are accomplished through an assortment of machine-learning methods.

Stock return Predictability: Economic restrictions on OLS

- Still, there are two easy-going ways to possibly improve OLS estimates.
- Source: Gu, Kelly, and Xiu (2019).
- Base case: the pooled OLS estimator corresponds to a panel (balanced) regression of future returns on firm attributes, where *T* and *N* represent the time-series and the cross-section dimensions.
- The objective is formulated as

$$\mathcal{L}(\theta) = \frac{1}{NT}\sum_{i=1}^{N}\sum_{t=1}^{T}\left(r_{i,t+1} - f\left(x_{i,t};\theta\right)\right)^2$$

where $r_{i,t+1}$ is stock return at time $t+1$ per firm i, $f = x'_{i,t}\theta$ is the corresponding predicted return, $x_{i,t}$ is a set of firm characteristics, and θ stands for model parameters.

- Predictive performance could be improved using optimization that value weights, rather than equal weights, stocks based on market size, inverse volatility (precision), inverse credit risk, etc.

Stock return Predictability: Economic restrictions on OLS

- An alternative optimization takes account of the heavy tail displayed by stocks and the potential harmul effects of outliers.
- The objective is formulated such that squared (absolute) loss is applied to small (large) errors:

$$\mathcal{L}(\theta) = \frac{1}{NT}\sum_{i=1}^{N}\sum_{t=1}^{T} H\left(r_{i,t+1} - f\left(x_{i,t};\theta\right),\xi\right)$$

where ξ is a tuning hyper-parameter and

$$H(y,\xi) = \begin{cases} y^2, & \text{if} \quad |y| \le \xi \\ 2\xi|y| - \xi^2, & \text{if} \quad |y| > \xi \end{cases}$$

- The hyper-parameter ξ is determined by model performance in a validation sample.
- Later, the selection of hyperparameters is described in detail.
- At this point, we are ready to proceed with machine learning methods, starting from Ridge.

Ridge Regression

- Ridge is one of several shrinkage methods.
- Hoerl and Kennard (1970a, 1970b) introduce the Ridge regression

$$\min (Y - X\beta)'(Y - X\beta) \text{ s. t. } \sum_{j=1}^{M} \beta_j^2 \leq c$$

- The minimization can be rewritten as

$$\mathcal{L}(\beta) = (Y - X\beta)'(Y - X\beta) + \lambda(\beta'\beta)$$

- We get

$$\hat{\beta}^{\text{ridge}} = (X'X + \lambda I_M)^{-1} X'Y$$

where I_M is the identity matrix of order M.

- A first-order benefit is that the regression coefficients can be expressed analytically.
- Notice that including λ makes the problem nonsingular even when $X'X$ is noninvertible.
- λ is a hyperparameter that controls for the amount of regularization.

Ridge Regression

- As $\lambda \to 0$, the OLS estimator obtains.
- As $\lambda \to \infty$, we have $\hat{\beta}^{\text{ridge}} = 0$.
- Ridge regressions do not have a sparse representation (dropping irrelevant predictors), so using model selection criteria to pick λ is infeasible.
- Instead, validation methods are employed.
- In particular, split the sample into three intervals: training, validation, and testing.
- The training sample considers various values for λ each of which delivers a prediction.
- The validation sample chooses λ which provides the smallest MSE.
- Hence, both training and validation samples are used to pick λ.
- Then, the experiment is assessed through out-of-sample predictions given the choice of λ.

Ridge Regression

- As shown below, in a Bayesian setting, the parameter λ denotes the prior precision of beliefs that regression slope coefficients are all equal to zero.
- Precision is the inverse of the variance.
- Classical perspective: the ridge estimator is essentially biased:

$$E\left(\hat{\beta}^{\text{ridge}}\right) \neq \beta$$

- There is no bias from a Bayesian perspective because $E\left(\hat{\beta}^{\text{ridge}}\right)$ is the posterior mean of β.
- The Bayesian analysis combines informed prior views with the likelihood function.
- The posterior mean is the weighted average of (i) the prior mean and (ii) the sample mean, with weights reflecting the precisions of (i) the prior and (ii) the sample estimate, respectively.

Interpretations of the Ridge Regression

Interpretation #1: Data Augmentation

- The Ridge-minimization problem can be formulated as

$$\sum_{t=1}^{T}(y_t - x_t'\beta)^2 + \sum_{j=1}^{M}\left(0 - \sqrt{\lambda}\beta_j\right)^2$$

- Thus, the Ridge-estimator is the usual OLS estimator where the data is transformed such that

$$X_\lambda = \begin{pmatrix} X \\ \sqrt{\lambda} I_M \end{pmatrix}, Y_\lambda = \begin{pmatrix} Y \\ 0_M \end{pmatrix}$$

where 0_M is an M-vector of zeros.

- Then, it follows that

$$\begin{aligned} \hat{\beta}^{\text{ridge}} &= (X_\lambda' X_\lambda)^{-1} X_\lambda' Y_\lambda \\ &= (X'X + \lambda I_M)^{-1} X'Y \end{aligned}$$

Interpretations of the Ridge Regression

Interpretation #2: Informative Bayes Prior

- Suppose the prior on β is of the form:

$$\beta \sim N\left(0, \frac{1}{\lambda} I_M\right)$$

- Then, the posterior mean of β is:

$$(X'X + \lambda I_M)^{-1} X'Y$$

- Bayesian methods are quite useful in asset pricing.
- For instance, consider the time-series asset pricing regression

$$r_t = \alpha + \beta r_{mt} + \varepsilon_t$$

- From a Bayesian perspective, we can formulate informed prior on mispricing

$$\alpha | V \sim N\left(0, \frac{\sigma^2}{s^2} V^{\eta}\right)$$

where V is the covariance matrix of the residuals in time-series asset pricing regressions, $s^2 = trace(V) = \sum_{j=1}^{N} \lambda_j$ and the $\lambda_j - s$ are eigenvalues of V.

Interpretations of the Ridge Regression

Interpretation #2: Informative Bayes Prior

- Notice that we can decompose the positive define matrix V as

$$V = Q\Lambda Q'$$

- Q is a matrix of ordered eignevectors that are orthogonal and Λ is a diagonal matrix with the corresponding ordered eigenvalues.
- Then, $trace(V) = trace(Q\Lambda Q') = trace(\Lambda Q'Q) = trace(\Lambda) = \sum_{j=1}^{N} \lambda_j$
- σ^2 controls for the degree of confidence in the prior, tuned by $trace(V)$.
- Limit cases: zero σ^2 means dogmatic beliefs while infinitely large σ^2 amounts to noninformative priors.
- The case $\eta = 1$ resembles the asset pricing prior of Pastor (2000) and Pastor and Stambaugh (2000).
- The PS prior is flexible since factors are prespecified and are not ordered per their importance, yet there are also merits for the $\eta = 2$ case, which applies to factors that are principal components, as discussed below.

Interpretations of the Ridge Regression

- Source: Kozak, Nagel, and Santosh (2020).
- To continue the Bayesian interpretation, consider the Hansen-Jagannathan representation of the pricing kernel

$$\begin{aligned} M_t &= 1 - b'(r_t - \mu) \\ &= 1 - \mu' V^{-1}(r_t - \mu) \\ &= 1 - \mu' Q \Lambda^{-1} Q'(r_t - \mu) \\ &= 1 - \mu_Q' \Lambda^{-1}(Q_t - \mu_Q) \\ &= 1 - b_Q'(Q_t - \mu_Q) \end{aligned}$$

where the second equation follows by the Euler equation -- $\mathrm{E}[M_t(r_t - \mu)]=0$.

- Assuming $\mu \sim N\left(0, \frac{\sigma^2}{s^2} V^{\eta}\right)$, it follows that $\mu_Q = Q'\mu$ has the prior distribution (V is assumed known)

$$\begin{aligned} \mu_Q = Q'\mu &\sim N\left(0, \frac{\sigma^2}{s^2} Q' \boldsymbol{V^{\eta}} Q\right) \\ &\sim N\left(0, \frac{\sigma^2}{s^2} Q' \boldsymbol{Q \Lambda^{\eta} Q'} Q\right) \\ &\sim N\left(0, \frac{\sigma^2}{s^2} \Lambda^{\eta}\right) \end{aligned}$$

Interpretations of the Ridge Regression

- As $b_Q = \Lambda^{-1}\mu_Q$, its prior distribution is formulated as (Λ is assumed known):

$$b_Q = \Lambda^{-1}\mu_Q \sim N\left(0, \frac{\sigma^2}{s^2}\Lambda^{\eta-2}\right)$$

- For $\eta < 2$, the variance of the b_Q coefficients associated with the smallest eigenvalues explodes.
- For $\eta = 2$, the pricing kernel coefficients $b = V^{-1}\mu$ have the prior distribution

$$b \sim N\left(0, \frac{\sigma^2}{s^2} I_N\right)$$

- Picking $\eta = 2$ makes the prior of b independent of V.
- Let us stick to this prior and further formulate the likelihood for b as

$$b \sim N\left(V^{-1}\hat{\mu}, \frac{1}{T}V^{-1}\right)$$

where $\hat{\mu}$ is the sample mean return.

- Then, the posterior mean of b is given by $E(b) = (V + \lambda I_N)^{-1}\hat{\mu}$ where $\lambda = \frac{s^2}{T\sigma^2}$.
- Ridge regression (pricing kerenl projected on N demeaned returns) with a tuning parameter λ would deliver the same *E(b)* coefficient.
- Or the Ridge (biased) coefficient is equal to the Bayesian posterior mean.

Interpretations of the Ridge Regression

- The prior expected value of the squared Sharpe ratio *(SR)* is given by (V is assumed known):

$$\begin{aligned} E(SR^2) &= E(\mu' V^{-1}\mu) \\ &= E(\mu' Q \Lambda^{-1} Q' \mu) \\ &= E\left(\mu_Q' \Lambda^{-1} \mu_Q\right) \\ &= E\left\{trace\left(\mu_Q' \Lambda^{-1} \mu_Q\right)\right\} \\ &= trace\left[\Lambda^{-1} E\left(\mu_Q \mu_Q'\right)\right] \\ &= \frac{\sigma^2}{s^2} trace[\Lambda^{\eta-1}] \end{aligned}$$

- The Pastor-Stambaugh type prior ($\eta = 1$) tells you that

$$E(SR^2) = \frac{\sigma^2}{s^2} trace(I_N) = N\frac{\sigma^2}{s^2}$$

- Thus, each principal component portfolio has the same expected contribution to the Sharpe ratio.
- If $\eta = 2$, then

$$E(SR^2) = \sum_{j=1}^{N} \frac{\sigma^2}{s^2} \lambda_j = \sigma^2$$

- Thus, the expected contribution of each PC is proportional to its eigenvalue.
- The derivations are based on knowing V, the covariance matrix, while a full Bayesian approach would formulate the inverted Wishart distribution for V. Then, the entire derivations will be different.

Interpretations of the Ridge Regression

Interpretation #3: Eigen-values and Eigen-vectors

- The singular value decomposition (SVD) is the basis for techniques in dimensionality reduction.
- By the unique SVD, we can express X as

$$X_{T\times M} = U_{T\times M}\Lambda^{0.5}_{M\times M}V'_{M\times M}$$

where $U = [U_1, \dots, U_M]$ is a $T \times M$ orthonormal matrix, $\Lambda^{0.5} = \begin{bmatrix} \lambda_1^{0.5} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \lambda_M^{0.5} \end{bmatrix}$ is an $M \times M$ matrix so that $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_M$, and $V = [V_1, V_2, \dots, V_M]$ is an $M \times M$ orthonormal matrix.

- As $X'X = V\Lambda V'$, the columns of V are the eigenvectors of $X'X$ and $(\lambda_1, \dots, \lambda_M)$ are the corresponding eigenvalues.
- As $XX' = U\Lambda U'$, the columns of U are the eigenvectors of XX'.
- The SVD facilitates dimensionality reduction by allowing us to focus on the most significant components.
- By retaining only the largest singular values (and their corresponding singular vectors), we can approximate the original matrix with lower rank, capturing the essential structure while reducing noise and complexity.

OLS – Eigen-values and Eigen-vectors based analysis

- By the singular value theorem, the OLS estimate can be reformulated as

$$\begin{aligned}\hat{\beta}^{\mathrm{OLS}} &= (X'X)^{-1}X'Y \\ &= (V\Lambda V')^{-1}V\Lambda^{0.5}U'Y \\ &= V\Lambda^{-1}V'V\Lambda^{0.5}U'Y \\ &= V\Lambda^{-0.5}U'Y \\ &= V\left[diag\left(\lambda_1^{-0.5},\lambda_2^{-0.5},\ldots,\lambda_M^{-0.5}\right)\right]U'Y\end{aligned}$$

- The fitted value is

$$\begin{aligned}\hat{Y}^{\mathrm{OLS}} &= X\hat{\beta}^{\mathrm{OLS}} \\ &= U\Lambda^{0.5}V'V\left[diag\left(\lambda_1^{-0.5},\lambda_2^{-0.5},\ldots,\lambda_M^{-0.5}\right)\right]U'Y \\ &= U\Lambda^{0.5}\Lambda^{-0.5}U'Y \\ &= UU'Y \\ &= \left[\sum_{j=1}^{M}(U_j\,U_j')\right]Y\end{aligned}$$

- Interpretation: we project Y on ***all*** the M columns of U.
- For comparison, as we show on the next pages, Ridge gives stronger prominence for columns in U associated with higher eigen values, while PCA considers only the first $K<M$ columns.

Interpretations of the Ridge Regression

- We examine how the predicted value of Y is related to the eigen vectors.
- First, we would like to find the eigenvectors and eigenvalues of the matrix Z

$$Z = X'X + \lambda I_M$$

- We know that for every $j=1,2\ldots,M$, the following holds by definition

$$(X'X)V_j = \lambda_j V_j$$

- Thus

$$\begin{aligned}(X'X + \lambda I_M)V_j &= (X'X)V_j + \lambda V_j \\ &= \lambda_j V_j + \lambda V_j \\ &= \left(\lambda_j + \lambda\right)V_j\end{aligned}$$

- Telling you that V still denotes the eigenvectors of Z while $\lambda_j + \lambda$ is the j-th eigenvalue.
- Notice now that if $A = V\Lambda V'$ then $A^L = V\Lambda^L V'$, while L can be either positive or negative.
- Hence, there are the same eigenvectors while eigenvalues are raised to the power of L.

Interpretations of the Ridge Regression

- Then, the inverse of the matrix Z is given by

$$Z^{-1} = V\left[diag\left(\frac{1}{\lambda_1+\lambda}, \frac{1}{\lambda_2+\lambda}, \dots, \frac{1}{\lambda_M+\lambda}\right)\right]V'$$

- The Ridge regression coefficients are

$$\begin{aligned}\hat{\beta}^{\text{ridge}} &= Z^{-1}X'Y \\ &= (X'X+\lambda I_M)^{-1}X'Y \\ &= V\left[diag\left(\frac{\lambda_1^{0.5}}{\lambda_1+\lambda}, \frac{\lambda_2^{0.5}}{\lambda_2+\lambda}, \dots, \frac{\lambda_M^{0.5}}{\lambda_M+\lambda}\right)\right]U'Y\end{aligned}$$

- And the fitted value is

$$\begin{aligned}\hat{Y}^{\text{ridge}} &= X\hat{\beta}^{\text{ridge}} \\ &= \left[\sum_{j=1}^{M}\left(U_j\frac{\lambda_j}{\lambda_j+\lambda}U_j'\right)\right]Y \\ &= U\left[diag\left(\frac{\lambda_1}{\lambda_1+\lambda}, \frac{\lambda_2}{\lambda_2+\lambda}, \dots, \frac{\lambda_M}{\lambda_M+\lambda}\right)\right]U'Y\end{aligned}$$

- Ridge regression projects Y onto components with large λ_j, shrinking the coefficients of low variance components.

PCA Vs. OLS Vs. Ridge Regressions

- In a principal components analysis (PCA) setup, we project Y on a subset of U_j, $j = 1,2,\dots,K < M$.
- Notice that U_j's are ordered per their corresponding eigenvalues in a descending order.
- The $X'X$ expression is approximated by using the K largest eigenvectors and eigenvalues

$$X'X = V\Lambda V' \approx \tilde{V}\tilde{\Lambda}\tilde{V}' = [V_1, \quad \dots, \quad V_K, \quad 0_{M\times(M-K)}]\left[diag\left(\lambda_1, \lambda_2, \dots, \lambda_K, 0_{(M-K)\times 1}\right)\right]\begin{bmatrix} V_1' \\ \vdots \\ V_K' \\ 0_{(M-K)\times M} \end{bmatrix}$$

- Then,

$$\hat{\beta}^{\text{PCA}} = \left(\tilde{V}\tilde{\Lambda}\tilde{V}'\right)^{-1} V\Lambda^{0.5}U'Y$$
$$= \tilde{V}\tilde{\Lambda}^{-1}\tilde{V}'V\Lambda^{0.5}U'Y$$
$$= \tilde{V}\tilde{\Lambda}^{-1}\begin{bmatrix} I_{K\times K} & 0_{K\times M-K} \\ 0_{M-K\times K} & 0_{M-K\times M-K} \end{bmatrix}\Lambda^{0.5}U'Y$$
$$= \tilde{V}\left[diag\left(\lambda_1^{-0.5}, \lambda_2^{-0.5}, \dots, \lambda_K^{-0.5}, 0,0,..\right)\right]U'Y$$

PCA – the predicted value

- The PCA predicted value is

$$\begin{aligned}\hat{Y}^{\text{PCA}} &= X\hat{\beta}^{\text{PCA}} \\ &= U\Lambda^{0.5}V'V\left[diag\left(\lambda_1^{-0.5},\lambda_2^{-0.5},\ldots,\lambda_K^{-0.5},0,0,..\right)\right]U'Y \\ &= U[diag(1,1,\ldots,1,0,0,..)]U'Y \\ &= \left[\sum_{j=1}^{K}(U_j\,U_j')\right]Y\end{aligned}$$

- PCA projects *Y* on the *K* first columns of *U*, rather than the entire *M* columns.
- Later in the notes, I discuss non-linear ways of extracting factors including autoencoder, conditional autoencoder (CA), and LSTM.
- Keep in mind that PCA seeks to **maximize the variance** within the predictors only, without considering the relationship with the response variable, aiming to reduce dimensionality while retaining as much variance as possible from *X*.
- Partial Least Squares (PLS), on the other hand, **maximizes the covariance** between the predictors and the response variables; more on the next slide.

Partial Least Squares(PLS)

- The PLS components are called latent variables.
- They are derived from both *X* and *Y* and are designed to **maximize the correlation between *X* and *Y*.**
- This makes PLS more effective for predictive modeling because it aligns the dimensionality reduction with the prediction task
- PLS Latent Variable Extraction works as follows.
- Let $t_1 = X\, w_1$, where t_1 is the first latent variable and w_1 is the weight vector.
- The first component w_1 maximizes $Cov(Xw_1, Y)$
- It is given by $w_1 = X'Y$; see the solution on the next page.

Derivation of w_i using Lagrange Multiplier

- The objective is to maximize Cov(X w_1, Y) = w_1' X' Y subject to a unity norm constraint.
- The Lagrangian is given by:

$$\mathcal{L}(w_1, \lambda) = w_1' X' Y - \lambda(w_1' w_1 - 1)$$

- Taking the derivative

$$\partial\mathcal{L}/\partial w_1 = X' Y - 2\lambda w_1 = 0$$

- Solving gives: $w_1 \propto X' Y$
- The second component is found similarly with the additional orthogonality constraint w.r.t. the first component.
- The third component has two orthogonality conditions and so on.

PLS Regression and Solution for Slope

- After extracting the latent variables, PLS regresses Y on these components:

  $Y = T\ C + E$, where T is the matrix of latent components, C is the regression coefficients, and E is the error

- The solution for the slope coefficients is given by:

  $$C = (T'T)^{-1}\ T'Y$$

- This is analogous to the Ordinary Least Squares (OLS) solution but uses the latent components instead of the original X matrix.

PLS – Summary

- PLS is a dimensionality reduction technique that projects predictors (independent variables) and responses (dependent variables) onto a new space, maximizing covariance between them.
- **Key Features**:

1. **Correlation Maximization**: PLS finds latent components (factors) that explain the maximum covariance between the predictors and response variables.
2. **Dimensionality Reduction**: Like PCA, PLS reduces the number of predictors, but it focuses on those that are most predictive of the response.
3. **Handling Multicollinearity**: PLS is particularly effective when predictors are highly collinear, as it constructs orthogonal latent variables that summarize the predictor space.
4. **Predictive Power**: By focusing on the relationship between predictors and responses, PLS enhances predictive power compared to methods like PCA that ignore the response.

IPCA: Instrumental Principal Component Analysis

- IPCA, per Kelly, Pruitt, and Su (2017, 2019), is conceptually closer to PLS than to PCA because both IPCA and PLS are **supervised** methods that incorporate the relationship between the predictors and the target variables.
- The factor model for excess returns is formulated as

$$r_{i,t+1} = \beta'_{i,t} f_{t+1} + \epsilon_{i,t+1}$$
$$\beta'_{i,t} = x'_{i,t} \Gamma_\beta$$

where f_{t+1} is a K-vector of latent factors.

- The loadings depend on observable asset characteristics contained in the $M \times 1$ vector $x_{i,t}$ (the first element is one for the intercept), while Γ_β is an $M \times K$ matrix.
- Motivation: Gomes, Kogan, and Zhang (2003) formulate an equilibrium model where beta varies with firm level predictors, such as size and book-to-market.
- Avramov and Chordia (2006) show empirically that conditional beta that varies with firm characteristics improves model pricing abilities.
- Characteristics can simply reflect covariances, or risk sources.

IPCA

- Rewriting the model in a vector form (collecting all the assets at time t):

$$r_{t+1} = X_t \Gamma_\beta f_{t+1} + \epsilon_{t+1}$$

where r_{t+1} is an $N \times 1$ vector of excess returns (number of assets can be time varying), X_t is $N \times M$ matrix of the characteristics, and ϵ_{t+1} is an $N \times 1$ vector of residuals.

- The estimation objective is to minimize

$$\min_{\Gamma_\beta, F} \sum_{t=1}^{T-1} (r_{t+1} - X_t \Gamma_\beta f_{t+1})' (r_{t+1} - X_t \Gamma_\beta f_{t+1})$$

- From the first order condition, we get that for $t = 1, 2, \ldots, T-1$

$$\hat{f}_{t+1} = (\hat{\Gamma}_\beta' X_t' X_t \hat{\Gamma}_\beta)^{-1} \hat{\Gamma}_\beta' Z_t' r_{t+1}$$

and

$$vec(\hat{\Gamma}_\beta') = \left(\sum_{t=1}^{T-1} X_t' X_t \otimes \hat{f}_{t+1} \hat{f}_{t+1}'\right)^{-1} \left(\sum_{t=1}^{T-1} \left[X_t \otimes \hat{f}_{t+1}'\right]' r_{t+1}\right).$$

IPCA

- Estimating the IPCA parameters can efficiently be conducted through Bayesian Gibbs Sampling.
- The Bayesian approach is extremely convenient and moreover you can adopt economically meaningful prior beliefs on the various coefficients.
- Kelly, Pruitt, and Su (2017) propose ways of solving the system as well as they give a plausible managed portfolio-based interpretation to the problem.
- Notice that in a static latent factor model, stock returns are formulated as

$$r_t = \beta f_t + \epsilon_t$$

and the PCA factor solution is

$$\hat{f}_t = (\beta'\beta)^{-1}\beta' r_t$$

- IPCA is analogous, while it accounts for dynamic instrumented betas.

IPCA

- One can also allow for mispricing, where the intercepts in the unrestricted factor model vary with the same firm characteristics.
- Then, the model is formulated as

$$r_{i,t+1} = x'_{i,t}\Gamma_\alpha + x'_{i,t}\Gamma_\beta f_{t+1} + \epsilon_{i,t+1}$$

where Γ_α is an $L \times 1$ vector.

- Let $\tilde{\Gamma} = [\Gamma_\alpha, \Gamma_\beta]$ and let $\tilde{f}_{t+1} = [1, f_{t+1}']'$.
- The model can be rewritten in a matrix form

$$r_{t+1} = X_t \tilde{\Gamma} \tilde{f}_{t+1} + \epsilon_{t+1}$$

- From the first-order minimization condition, we get for $t = 1, 2, \ldots, T-1$

$$\hat{\tilde{f}}_{t+1} = \left(\hat{\tilde{\Gamma}}_\beta' X_t' X_t \hat{\tilde{\Gamma}}_\beta\right)^{-1} \hat{\tilde{\Gamma}}_\beta' X_t' \left(r_{t+1} - X_t \hat{\tilde{\Gamma}}_\alpha\right)$$

and

$$vec\left(\hat{\tilde{\Gamma}}'\right) = \left(\sum_{t=1}^{T-1} X_t' X_t \otimes \hat{\tilde{f}}_{t+1} \hat{\tilde{f}}_{t+1}'\right)^{-1} \left(\sum_{t=1}^{T-1} \left[X_t \otimes \hat{\tilde{f}}_{t+1}'\right]' r_{t+1}\right).$$

Lasso(Least Absolute Shrinkage and Selection Operator)

- Tibshirani (1996) was the first to introduce Lasso.
- Lasso simultaneously performs variable selection and coefficient estimation via shrinkage.
- While the ridge regression implements an l_2-penalty, Lasso is an l_1-optimization:

$$\min (Y - X\beta)'(Y - X\beta) \text{ s. t. } \sum_{j=1}^{M} |\beta_j| \leq c$$

- The l_1 penalization approach is called basis pursuit in signal processing.
- There is, again, a non-negative tuning parameter λ that controls for the amount of regularization:

$$\mathcal{L}(\beta) = (Y - X\beta)'(Y - X\beta) + \lambda \sum_{j=1}^{M} |\beta_j|$$

- Both Ridge and Lasso have solutions even when $X'X$ may not be of full rank (e.g., when there are more explanatory variables than time-series observations) or ill conditioned.

Lasso – Sparsity

- Unlike Ridge, the Lasso coefficients cannot be expressed in closed form.
- However, Lasso generates sparse solutions, retaining the variables that matter most.
- This improves the interpretability of regression models.
- Large enough λ will set some coefficients exactly to zero.
- To understand why, notice that LASSO can be casted as having a Laplace prior on β

$$P(\beta|\lambda) \propto \left(\frac{\lambda}{2\sigma}\right) \exp\left(-\frac{\lambda|\beta|}{\sigma}\right)$$

- Lasso obtains by combining Laplace prior and normal likelihood.
- Like the normal distribution, Laplace is symmetric.
- Unlike the normal distribution, Laplace spikes at zero (first derivative is discontinuous) and it concentrates its probability mass closer to zero than does the normal distribution.
- This explains why Lasso sets some coefficients to zero, while Ridge (normal prior) does not.

Lasso – picking the shrinkage intensity

- Because LASSO imposes sparsity, you can use model selection criteria to pick λ
- Can also use a validation sample, as in Ridge.
- Examples of model selection criteria include AIC, BIC, FIC, and PIC.
- Model selection criterion consists of (i) goodness-of-fit and (ii) a penalty factor that gets larger as the number of retained variables increases.
- Occam's razor: the law of parsimony – thinner models are preferred.
- Bayesian information criterion (BIC) is often used:

$$BIC = T \times log\left(\frac{RSS}{T}\right) + l \times log(T), \text{ where } l \text{ is the number of variables retained.}$$

- Different values of lambda affect the optimization in a way that a different set of characteristics is retained.
- You choose λ as follows: initiate a range of values, compute BIC for each value, and pick the one that minimizes BIC.

The Spike and Slab Regression

- In all fields of econometrics, there is large uncertainty about whether the true underlying model is sparse (as in Lasso) or dense (as in Ridge) —whether only a few predictors truly matter, or if many variables have non-trivial influence.
- A spike-and-slab prior introduces a Bayesian framework that addresses this uncertainty by learning from the data whether the model is sparse or dense.
- This dual prior allows the combination of sparse and non-sparse representations.
- The steps are provided below, assuming *M* candidate predictors of future returns.
- Consider a binary inclusion vector $\gamma = (\gamma_1, \gamma_2, \ldots \gamma_M)'$ where $\gamma_i = 1$ indicates the inclusion of the i-*th* variable in the model, and $\gamma_i = 0$ indicates its exclusion.
- In the absence of compelling prior information, we assign a Bernoulli prior with probability *p* to each variable, determining the subset of variables retained in the model and the subset that is excluded.

The Slack and Slab Regression

- The 'spike' component of the prior concentrates mass at zero for coefficients associated with $\gamma_i = 0$.
- Conditional on γ, we draw the regression coefficients β from a multivariate normal prior, typically with zero mean and a large variance ('slab'), which allows for non-zero effects when variables are included.
- The posterior distribution of both the inclusion vector γ and the coefficients β is updated using Gibbs Sampling, a type of Markov Chain Monte Carlo (MCMC) method.
- This algorithm iterates between updating γ and β in a stepwise manner based on their conditional posterior distributions.

Step-by-Step Procedure

- Sample γ_i: For each variable, given the current value of β and the observed data, the probability that γ_i is updated using the Bernoulli distribution:

$$P(\gamma_i = 1 \mid \beta, y, X) \propto P(y \mid X, \beta)\, P(\beta \mid \gamma)$$

- Sample β: Conditional on the sampled γ, the regression coefficients are drawn from a multivariate normal distribution:

$$\beta \mid \gamma, y, X \sim N(\mu_\beta, \Sigma_\beta)$$

- These steps are repeated iteratively across thousands of iterations, yielding the posterior distributions of γ and β, approximating the true posterior.

The Spike and Slab Prior Explained

- The spike component enforces sparsity by placing mass on zero for coefficients where the inclusion indicator $\gamma_i = 0$.
- It models the probability that a given variable does not influence the dependent variable.
- For variables with $\gamma_i = 1$, the corresponding β coefficients are drawn from a normal distribution.
- The variance of this slab prior reflects our belief that included variables can have significant but varying effects.
- The combination of spike and slab allows the model to infer which variables should be included (sparsity) and which should have non-zero effects (inclusion) based on the posterior probability of the coefficients.

The Spike and Slab - Advantages

- The spike-and-slab prior adapts to both sparse and dense models, allowing for efficient variable selection in high-dimensional settings where the true model structure is uncertain.
- Using Gibbs sampling, we can draw inferences from the posterior distributions of the model parameters, accounting for uncertainty in both inclusion (via γ) and coefficients (via β).
- Bayesian econometrics provides a natural framework for incorporating prior beliefs about model structure, whether based on theoretical considerations or previous empirical studies.
- Prior knowledge about the importance of certain variables can be encoded into the model via the prior distribution.
- The spike-and-slab prior helps avoid overfitting by penalizing the inclusion of irrelevant variables, improving the model's inferences.

The Adaptive Lasso

- Revisiting LASSO, the approach forces coefficients to be equally penalized.
- One modification is to assign different weights to different coefficients:

$$\mathcal{L}(\beta) = (Y - X\beta)'(Y - X\beta) + \lambda \sum_{j=1}^{M} w_j |\beta_j|$$

- It can be shown that if the weights are data driven and are chosen in the right way, the weighted LASSO can have the so-called ***oracle*** properties even when the LASSO does not have the oracle property.
- This is the adaptive LASSO.
- For instance, w_j can be chosen such that it is equal to one divided by the absolute value of the corresponding OLS coefficient raised to the power of $\gamma > 0$. That is, $w_j = \frac{1}{|\beta_j|^{\gamma}}$ for $j = 1, \ldots, M$, where β_j comes from unconstrained optimization (OLS).
- The adaptive LASSO estimates are given by

$$\mathcal{L}(\beta) = (Y - X\beta)'(Y - X\beta) + \lambda \sum_{j=1}^{M} w_j |\beta_j|$$

- Hyper-parameters λ and γ can be chosen using model selection criteria.
- The adaptive LASSO is a convex optimization problem and thus does not suffer from multiple local minima.
- Later, I describe how adaptive LASSO has been implemented in asset pricing through an *RFS* paper.

Bridge Regression

- Frank and Friedman (1993) introduce the bridge regression.
- This specification generalizes for ℓq penalty.
- The optimization is given by

$$\mathcal{L}(\beta) = (Y - X\beta)'(Y - X\beta) + \lambda \sum_{j=1}^{M} |\beta_j|^q$$

- Notice that $q = 0, 1, 2$, correspond to OLS, LASSO, and Ridge, respectively.
- Moreover, the optimization is convex for $q \geq 1$ and the solution is sparse for $0 \leq q \leq 1$.
- Eventually, q is a hyperparameter to be selected.
- So, there are two hyperparameters.
- When the solution is sparse – use model selection criteria to pick hyperparameters.
- Otherwise, use a validation sample.

The Elastic Net

- The elastic net is yet another regularization and variable selection method.
- Zou and Hastie (2005) describe it as stretchable fishing net that retains "all big fish."
- Using simulation, they show that it often outperforms Lasso in terms of prediction accuracy.
- The elastic net encourages a grouping effect, where strongly correlated predictors tend to be in or out of the model together.
- The elastic net is particularly useful when the number of predictors is much larger than the number of observations.
- The naïve version of the elastic net is formulated through

$$\mathcal{L}(\beta) = (Y - X\beta)'(Y - X\beta) + \lambda_1 \sum_{j=1}^{M} |\beta_j| + \lambda_2 \sum_{j=1}^{M} \beta_j^2$$

- Thus, the elastic net combines l_1 and l_2 norm penalties.
- It still produces sparse representations.
- Thus, can use model selection to pick the hyperparameters.

The Group Lasso

- Suppose the M predictors can be classified into L-representing groups.
- In financial economics, you can divide predictive characteristics into accounting versus market or technical versus fundamental.
- Or you can divide the characteristics into valuation ratios, profitability, investment, and liquidity.
- Let M_l denote the number of predictors per group l.
- Let X_l represent the predictors corresponding to the l-th group, while β_l is the corresponding coefficient vector.
- Then, $\beta=[\beta_1', \beta_2', \ldots, \beta_L']'$, which amounts to a restricted regression with M coefficients but L ***distinct*** coefficients.

The Group Lasso

- The group Lasso solves the convex optimization problem.

$$\mathcal{L}(\beta) = \left(Y - \sum_{l=1}^{L} X_l \beta_l\right)' \left(Y - \sum_{l=1}^{L} X_l \beta_l\right) + \lambda \left(\sum_{l=1}^{L} \beta_l' \beta_l\right)^{\frac{1}{2}}$$

- The group Lasso yields sparsity across the groups, in that some groups are excluded.
- However, there is no sparsity within a group: if a group of parameters is nonzero, all the group members are nonzero.
- The ***sparse*** group Lasso criterion yields sparsity

$$\mathcal{L}(\beta) = \left(Y - \sum_{l=1}^{L} X_l \beta_l\right)' \left(Y - \sum_{l=1}^{L} X_l \beta_l\right) + \lambda_1 \left(\sum_{l=1}^{L} \beta_l' \beta_l\right)^{\frac{1}{2}} + \lambda_2 \sum_{j=1}^{M} |\beta_j|$$

Nonparametric-nonlinear methods

- Lasso, Adaptive Lasso, Group Lasso, Ridge, Bridge, and Elastic net are all linear or parametric approaches for shrinkage.
- Some other parametric approaches (uncovered here) include the smoothed clip absolute deviation (SCAD) penalty of Fang and Li (2001) and Fang and Peng (2004) and the minimum concave penalty of Zhang (2010).
- In many applications, however, there is little a priori justification to assume that the effects of covariates take a linear form or belong to other known parametric families.
- Huang, Horowitz, and Wei (2010) thus propose to use a nonparametric approach: the adaptive group Lasso for variable selection.
- This approach is based on a spline approximation to the nonparametric components.
- To achieve model selection consistency, they apply Lasso in two steps.
- First, they use group Lasso to obtain an initial estimator and reduce the dimension of the problem.
- Second, they use the adaptive group Lasso to select the final set of nonparametric components.

Nonparametric Models in asset pricing

- Cochrane (2011) notes that portfolio sorts are equivalent to nonparametric cross section regressions.
- Following Huang, Horowitz, and Wei (2010), Freyberger, Neuhier, and Weber (2017) study this equivalence formally.
- The cross section of stock returns is modelled as a nonlinear function of firm characteristics:

$$r_{it} = m_t\left(C_{1,it-1}, \dots, C_{S,it-1}\right) + \epsilon_{it}$$

- Notation:
  - r_{it} is the return on firm *i* at time *t*.
  - m_t is a function of *S* firm characteristics $C_1, C_2, \dots, C_S$.
  - Notice, m_t itself is not stock specific but firm characteristics are, just like slopes in cross section regressions
- Consider an additive model of the following form

$$m_t(C_1, \dots, C_S) = \sum_{s=1}^{S} m_{t,s}(C_s)$$

- As the additive model implies that $\frac{\partial^2 m_t(c_1,\dots,c_S)}{\partial c_s \partial c_{s'}}$=0 for $s \neq s'$, apparently there should not be no cross dependencies between characteristics.
- Such dependencies can still be accomplished through producing more predictors as interactions between characteristics.

Nonparametric Models

- For each characteristic *s*, let $F_{s,t}(\cdot)$ be a strictly monotone function and let $F_{s,t}^{-1}(\cdot)$ denote its inverse.
- Define $\tilde{C}_{s,it-1} = F_{s,t}(C_{s,it-1})$ such that $\tilde{C}_{s,it-1} \in [0,1]$.
- That is, characteristics are monotonically mapped into the [0,1] interval.
- An example for $F_{s,t}(\cdot)$ is the rank function: $F_{s,t}(C_{s,it-1}) = \frac{rank(C_{s,it-1})}{N_t}$, where N_t is the total number of firms at time *t*.
- The aim then is to find $\widetilde{m}_t$ such that

$$m_t(C_1, \ldots, C_S) = \widetilde{m}_t(\tilde{C}_{1,it-1}, \ldots, \tilde{C}_{s,it-1})$$

- In particular, to estimate the $\widetilde{m}_t$ function, the normalized characteristic interval $[0,1]$ is divided into *L* subintervals (*L+1* knots): $0 = x_0 < x_1 < \cdots < x_{L-1} < x_L = 1$.
- To illustrate, consider the equal spacing case.
- Then, $x_l = \frac{l}{L}$ for *l=0,…,L-1* and the intervals are:

$$\widetilde{I}_1 = [x_0, x_1),\ \widetilde{I}_l = [x_{l-1}, x_l) \text{ for } l=2,\ldots,L-1, \text{ and } \widetilde{I}_L = [x_{L-1}, x_L]$$

Nonparametric Models

- Each firm characteristic is transformed into its corresponding interval.
- Estimating the unknown function $\tilde{m}_{t,s}$ nonparametricaly is done by using quadratic splines.
- A quadratic spline is a differentiable piecewise quadratic function.
  - The function $\tilde{m}_{t,s}$ is approximated by a quadratic function on each interval $\tilde{I}_l$.
  - Quadratic functions in each interval are chosen such that $\tilde{m}_{t,s}$ is continuous and differentiable in the whole interval $[0,1]$.
  - $\tilde{m}_{t,s}(\tilde{c}) = \sum_{k=1}^{L+2} \beta_{tsk} \times p_k(\tilde{c})$, where $p_k(\tilde{c})$ are basis functions and β_{tsk} are estimated slopes.
  - In particular, $p_1(y) = 1, p_2(y) = y, p_3(y) = y^2,$ and $p_k(y) = \max\{y - x_{k-3}, 0\}^2$ for $k = 4, \ldots, L+2$
  - In that way, you can get a continuous and differentiable function.
- To illustrate, consider the case of two characteristics, e.g., size and book to market (BM), and 3 intervals.
- Then, the $\tilde{m}_t$ function is:

$$\tilde{m}_t(\tilde{c}_{i,size}, \tilde{c}_{i,BM}) =$$

$$= \beta_{t,size,1} \times 1 + \beta_{t,size,2} \times \tilde{c}_{i,size} + \beta_{t,size,3} \times \tilde{c}_{i,size}^2 + \beta_{t,size,4} \times \max\{\tilde{c}_{i,size} - 1/3, 0\}^2 + \beta_{t,size,5} \times \max\{\tilde{c}_{i,size} - 2/3, 0\}^2$$
$$+ \beta_{t,BM,1} \times 1 + \beta_{t,BM,2} \times \tilde{c}_{i,BM} + \beta_{t,BM,3} \times \tilde{c}_{i,BM}^2 + \beta_{t,BM,4} \times \max\{\tilde{c}_{i,BM} - 1/3, 0\}^2 + \beta_{t,BM,5} \times \max\{\tilde{c}_{i,BM} - 2/3, 0\}^2$$

Adaptive group Lasso

- The estimation of $\widetilde{m}_t$ is done in two steps:
- First step, estimate the slope coefficients b_{sk} using the group Lasso routine:

$$\widehat{\beta}_t = \underset{b_{sk:s=1,\ldots,S;k=1,\ldots,L+2}}{\operatorname{argmin}} \sum_{i=1}^{N_t} \left(r_{it} - \sum_{s=1}^{S} \sum_{k=1}^{L+2} b_{sk} \times p_k\left(\tilde{C}_{s,it-1}\right) \right)^2 + \lambda_1 \sum_{s=1}^{S} \left(\sum_{k=1}^{L+2} b_{sk}^2 \right)^{1/2}$$

- Altogether, the number of b_{sk} coefficients is $S \times (L+2)$.
- The second expression is a penalty term applied to the spline expansion.
- λ_1 is chosen such that it minimizes the Bayesian Information Criterion (BIC).
- Each characteristic represents a group.
- The essence of group Lasso is to either include or exclude all $L+2$ spline terms associated with a given characteristic.
- While this optimization yields a sparse solution there are still many characteristics retained.
- To include only characteristics with strong predictive power the adaptive Lasso is then employed.

Adaptive group Lasso

- To implement adaptive group Lasso, define the following weights using estimates for b_{sk} from the first step:

$$w_{ts} = \begin{Bmatrix} \left(\sum_{k=1}^{L+2} \tilde{b}_{sk}^2\right)^{-\frac{1}{2}} & if & \sum_{k=1}^{L+2} \tilde{b}_{sk}^2 \neq 0 \\ \infty & if & \sum_{k=1}^{L+2} \tilde{b}_{sk}^2 = 0 \end{Bmatrix}$$

- Then, estimate again the coefficients b_{sk} using the above-estimated weights w_{ts}

$$\widehat{\beta}_t = \underset{b_{sk:s=1,\ldots,S;k=1,\ldots,L+2}}{\operatorname{argmin}} \sum_{i=1}^{N_t} \left(r_{it} - \sum_{s=1}^{S} \sum_{k=1}^{L+2} b_{sk} \times p_k\left(\tilde{C}_{s,it-1}\right) \right)^2 + \lambda_2 \sum_{s=1}^{S} \left(w_{ts} \sum_{k=1}^{L+2} b_{sk}^2 \right)^{1/2}$$

- λ_2 is chosen such that it minimizes BIC.
- The above formulation of weights w_{ts} guarantees that the second step does not pick characteristics that are excluded in the first step.

Regression Trees

- Regression trees are a nonparametric machine learning technique used to model decisions based on input variables, resulting in a tree-like structure where each decision (or split) is made based on specific criteria.
- They are particularly useful when predicting outcomes that depend on multiple interacting variables and when the relationship between predictors and outcomes is nonlinear.
- In the context of asset returns, a regression tree can be used to decide whether to sort stocks by a particular characteristic.
- If sorting by that characteristic is not effective, the tree can then ask whether another characteristic might be more useful.
- At each decision point (or node), the tree asks whether a cut-off at a specific value of the chosen variable would help divide the stocks into two groups—each with similar characteristics.
- For example, the tree might ask if sorting stocks by market capitalization and then applying a cut-off at a particular value could create two distinct portfolios.

Regression Trees

- The tree is built by selecting splits that minimize prediction error.
- Specifically, at each node, the dataset is divided into two subsets that minimize the mean squared error (MSE) of predicted returns:

$$\mathcal{L}(C, C_{left}, C_{right}) = \frac{1}{N_{left}} \sum (r_{i,t+1} - \theta_{left})^2 + \frac{1}{N_{right}} \sum (r_{i,t+1} - \theta_{right})^2$$

- Notation: C denotes the data set from the preceding step, while the new bins are C_{left} and C_{right}, and the $N-$s are the corresponiding number of observations.
- The predicted return is the average of returns of all stocks within the group

$$\theta_{left} = \frac{1}{N_{left}} \sum_{z_{i,t} \in C_{left}} r_{i,t+1} \; ; \; \theta_{right} = \frac{1}{N_{right}} \sum_{z_{i,t} \in C_{right}} r_{i,t+1}$$

Pros of Regression Trees

- **Simplicity and Interpretability:**
- Regression trees are intuitive and easy to interpret. Each split represents a simple decision rule that can be visualized, making it easy to understand the relationship between variables and outcomes.
- **Nonlinearity and Interactions:**
- They naturally capture nonlinear relationships and interactions between predictors without requiring complex transformations or assumptions about the underlying data distribution.
- **Versatility:**
- Regression trees can be used for both regression and classification tasks, making them versatile across different domains.
- **Handle Missing Data:**
- They can handle missing values by splitting the data based on available information, without requiring imputation or removal of missing cases.
- **No Assumptions on Data Distribution:**
- Unlike linear models, regression trees do not require any assumptions about the linearity or normality of the data, allowing them to adapt to complex datasets.
- **Variable Importance:**
- Trees naturally rank variables by importance, as splits are based on the variable that most reduces the prediction error at each step.

Cons of Regression Trees

- **Prone to Overfitting:**
- Without proper tuning, regression trees can easily overfit the training data, capturing noise rather than underlying patterns. This happens when the tree grows too deep with too many nodes.
- **Instability:**
- Small changes in the data can result in entirely different trees being generated, making them unstable. This is because the splits at each step are highly sensitive to variations in the data.
- **Bias Toward Dominant Features:**
- Regression trees can be biased toward features with more levels or categories. Features with more unique values are more likely to be chosen for splits, which may not always reflect true predictive power.
- **Limited Predictive Power:**
- While easy to interpret, single regression trees often lack predictive accuracy, especially when compared to more sophisticated models like random forests, gradient boosting, or neural networks.
- **Difficulty Capturing Smooth Relationships:**
- Since regression trees use step functions to make predictions, they may struggle to capture smooth relationships between the independent and dependent variables.
- **Greedy Algorithm:**
- The algorithm makes locally optimal decisions at each split, without considering the overall structure of the tree. This can lead to suboptimal trees, which might not represent the best global model.
- **Need for Pruning:**
- To combat overfitting, trees often need to be pruned (i.e., removing branches that add complexity without improving accuracy), which can be computationally intensive and adds an additional layer of complexity.

Random Forest

- A random forest is an ensemble learning method that combines multiple decision trees to improve prediction accuracy and control for overfitting.
- While a single decision tree may be prone to overfitting by learning too much from the training data, a random forest builds several trees using random subsets of the data and features, and then aggregates their predictions.
- The process works as follows:

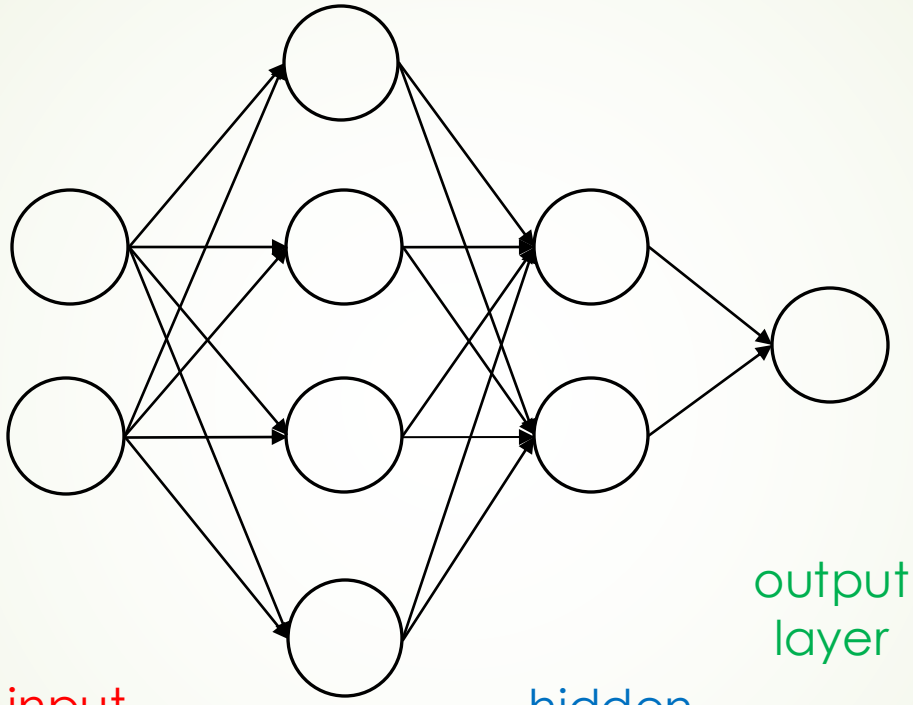
1. **Bootstrap Sampling:** Each tree in the random forest is trained on a randomly sampled subset (with replacement) of the original data.
2. **Feature Randomness:** At each node within a tree, only a random subset of the features is considered for splitting. This increases diversity among the trees and prevents any single variable from dominating the predictions.
3. **Averaging Predictions:** Once all trees are built, the final prediction is made by averaging the predictions of each individual tree (in the case of regression) or by majority voting (in the case of classification).

- Random forests improve prediction performance by reducing variance.
- The individual trees might have high variance (i.e., they could overfit the training data), but when their predictions are averaged together, the variance is reduced, leading to more accurate and robust predictions.

Neural Networks

- All upcoming machine learning routines will utilize Neural Networks (NN) with varying levels of complexity.
- NNs form the foundation of deep learning and are highly capable of approximating complex functions in high-dimensional spaces, as well as capturing intricate time dependencies (e.g., LSTM, Transformer Encoders).
- Inspired by the structure of the human brain, NNs consist of layers of "neurons" connected by "synapses" that transmit signals between layers.
- NN models ingest data, learn to recognize patterns, and produce predictions.
- NNs have diverse applications, including facial recognition, time series forecasting (e.g., stock returns, rainfall), music composition, and self-driving cars.
- The network processes information from an **input layer**, through one or more **hidden layers**, to an **output layer**—a structure often called a **Feed-Forward Network (FFN)**.
- The **output layer** makes predictions similar to fitted values in regression analysis.
- Most data processing occurs in the **hidden layers**, which consist of interconnected neurons that extract complex features.

Fully connected (dense layer) Network



Neural Networks

- Each neuron applies a nonlinear activation function f to its aggregated signal before sending its output to the next layer

$$x_k^l = f(\theta_0 + \sum_j z_j \theta_j)$$

where x_k^l corresponds to neuron $k \in 1,2,\dots,K^l$ in the hidden layer $l \in 1,2,\dots,L$.

- The activation function (or the threshold function) is usually one of the following
  - Sigmoid $\sigma(x) = \frac{1}{(1+e^{-x})}$
  - $\tanh(x) = 2\sigma(x) - 1$
  - $ReLU(x) = \begin{cases} 0 & if\ x < 0 \\ x & otherwise \end{cases}$
- The sigmoid function is between 0 and 1
- The hyperbolic tangent is between -1 and 1
- The result of the activation function determines whether the particular neuron will get activated.
- An activated neuron transmits data to the neuron of the next layer over the channel – forward propagation – data propagates through the network.
- NN is essentially a nonlinear nonparametric regression.
- With a linear activation function, NN boils down to OLS.

Neural Networks

- For the ReLU activation function, we can rewrite the neural network function as:

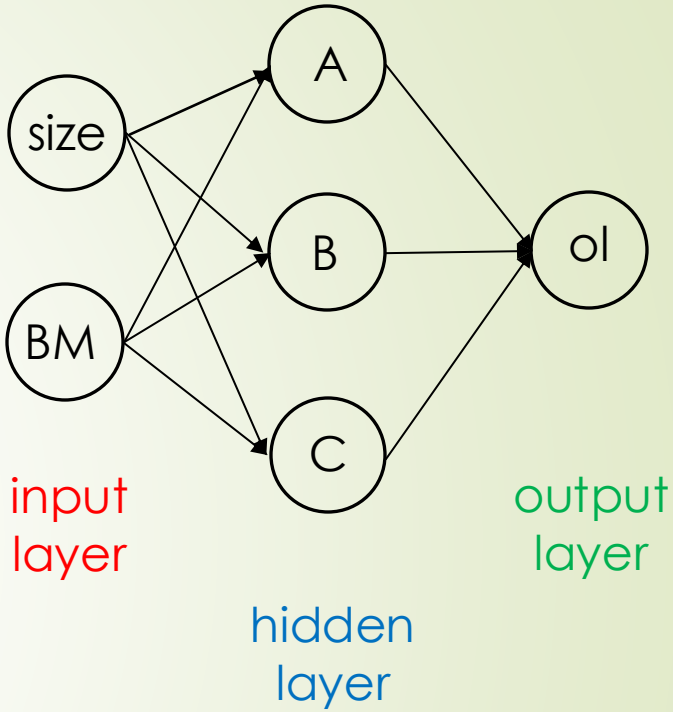
$$output = \max\left(\max\left(\max\left(XW_{\text{hl}}^{1}, 0\right)W_{\text{hl}}^{2}, 0\right) \ldots W_{\text{hl}}^{\text{n}}, 0\right) W_{\text{output}}$$

where X is the input, W_{hl}^{i} are the weight matrix of the neurons in hidden layer $i \in 1, \ldots, n$, n is the number of hidden layers, and W_{output} are the weighs of the output layer.

- In NN, slopes are termed weights while intercepts are termed biases.
- Then, run an optimization to minimize the loss function.
- When the predicted variable is continuous, mean squared errors (MSE) can be used for the loss function.
- To predict probability or for classification purposes, can use Softmax loss function.
- If the activation function is linear – simply ignore the MAX operator in the above equation.
- Then the output is XW – boiling down to OLS.

A simple example with ReLU

- Let us assume two inputs only: market cap and BM (book to market)
- One hidden layer with three neurons: A, B, and C
- W'-s are the slops (weights) while b-s are the intercepts (biases).
- $input^A = size \times W^A_{size} + BM \times W^A_{BM} + b^A$
- $output^A = max(input^A, 0)$
- $input^B = size \times W^B_{size} + BM \times W^B_{BM} + b^B$
- $output^B = max(input^B, 0)$
- $input^C = size \times W^C_{size} + BM \times W^C_{BM} + b^C$
- $output^C = max(input^C, 0)$
- Output layer (ol): $output = output^A \times W^{ol}_A + output^B \times W^{ol}_B + output^C \times W^{ol}_C + b^{ol}$
- The output is the predicted return.
- Implement that procedure for any stock while the parameters are identical across stocks.
- To find the parameters, you minimize the sum squared errors (realized versus predicted returns) by aggregating across all stocks and all months.



The output Layer – Interpretation

- The output layer is interpreted based on the particular experiment.
- In supervised learning, the output tries to come close to the label.
- If the label is return, as in the previous example, the output is predicted return.
- If the label is about identifying the top decile of stocks (ones versus zeros), the output is about the predicted probability of belonging to the top.
- Further, while in the previous example, the output reflects a scalar, it can also be a vector of dimension d.
- For instance, you apply different weights and biases to predict future returns – hence, there are multiple return predictions per stock.
- Then, if there are 1000 stocks, the intermediate output is 1000 by d matrix.
- You have to convert that matrix to a vector of dimension 1000 by multiplying the matrix by a vector of order d, which is yet another set of parameters to estimate.

Hyperparameters

- The number of hidden layers and the number of neurons per layer are both hyperparameters.
- The regularization (next page) parameters are also hyperparameters.
- The learning rate (coming up soon) establishes yet another hyperparameter.
- Can split the sample into three pieces: training, validation, and testing.
- To illustrate, suppose the sample spans the January 1981 till the end of 2024.
- Use the first twenty years as training – January 1981 till December 2000.
- Use the next ten years as validation – January 2001 till December 2010.
- Then, generate one year of monthly predictions – for the year 2011.
- Next…
- Training sample becomes January 1981 till December 2001.
- Validation sample becomes January 2002 till December 2011.
- Generate yet another year of monthly predictions – for the year 2012.
- And so on. ..
- The training sample is expanding.
- The validation sample is rolling.

Regularization

- Machine learning methods are subject to overfitting; hence, regularization is essential.
- You can implement LASSO on the weights and biases when minimizing the loss function.
- Lasso will mute some of the coefficients.
- However, it won't essentially mute a variable or variables, as in Lasso regressions.
- You can also implement Ridge or Elastic Net.
- Batch normalization: Normalizes the inputs to each layer by adjusting the mean and variance of the mini-batch, which helps stabilize training, allows for higher learning rates, and provides some regularization benefits.
- Early stopping: Stops training once the model's performance on a validation set starts to deteriorate, preventing the network from overfitting to the training data.
- There are many more regularization methods.
- One useful and intuitive approach is ***dropout,*** *to be explained on the next page.*

Dropout

- Dropout works by randomly "dropping out" a fraction of the neurons in the network during training.
- This prevents the network from relying too heavily on any one neuron and forces the model to learn more robust features.
- Here's how dropout works in practice:

1. **Training Phase:** During each iteration, which consists of a forward pass followed by a backward pass, a fraction of the neurons (e.g., p=20% or 50%) is randomly selected and dropped out. These neurons are ignored during both the forward pass (when activations are calculated) and the backward pass (when gradients are computed), so their weights do not get updated. The remaining neurons continue to process the data. Each time a new batch of data is processed in an iteration, a different random set of neurons is dropped out.
2. **Inference Phase:** During testing or validation, all the neurons are active, but their outputs are scaled down by the same fraction that was used during training. This adjustment ensures that the output remains consistent and accounts for the randomness introduced during training.

- The key benefit of dropout is that it reduces the risk of overfitting by preventing the network from relying too much on specific neurons. By forcing the network to use different subsets of neurons in each iteration, dropout helps create a more generalized model.

Training the network

- Training a neural network involves minimizing a loss function, quantifying the sum (across time and assets) differences between predicted and actual values.
- This is achieved through **gradient descent**, an optimization technique that updates the network's weights iteratively.
- Weights are typically initialized and updated using gradient descent.
- In each step, weights are adjusted as follows:
  new weight = old weight - (learning rate × gradient of the loss w.r.t to the weight)
- The **learning rate** is a crucial hyperparameter that controls the step size.
  - If it is too large, the network may overshoot the minimum.
  - If it is too small, training can be slow or get stuck in local minima.
- **AdaTune** is an adaptive learning rate algorithm that adjusts the learning rate dynamically during training, allowing for faster and more stable convergence.
- The training process aims to move in the direction of the steepest descent of the loss function, using the gradient (or its approximation) at each point to guide the updates.

Training the network

- **Backpropagation** is the key algorithm used to train neural networks by calculating the gradients required for updating weights.
- It works by applying the **chain rule** to compute the gradient of the loss w.r.t. each weight, starting from the output layer and propagating backward through the network.
- The chain rule allows efficient computation by breaking the gradient into simpler components at each layer.
- Backpropagation stores intermediate results, so gradients for previous layers don't need to be recalculated, making the process efficient.
- The goal is to minimize the loss by updating the weights based on these gradients.
- However, challenges arise due to:
- **Non-convexity**: The loss function often has many local minima and saddle points, making it hard to find the global minimum.
- **Vanishing gradients**: In deep networks, gradients can become very small as they propagate backward, slowing down learning.
- Common activation functions like **ReLU** (Rectified Linear Unit) help mitigate vanishing gradients by introducing non-linearity and maintaining strong gradient signals.
- Different optimizers can be used for training:
- **Stochastic Gradient Descent (SGD)**: Simple and widely used, but can be slow to converge.
- **Adam**: An adaptive optimizer that combines the benefits of momentum and learning rate adaptation for faster convergence.

Reconciling Classical Theory with Deep Learning

- Traditionally, it is expected that **larger models** should eventually underperform due to overfitting.
- However, modern deep-learning models challenge this conventional wisdom.
- So, which is correct—**common wisdom** or the **empirical evidence** from deep learning?
- The two perspectives can be **reconciled**:

- **Under-Parameterized Models**: As model complexity increases, **test error decreases** (following traditional bias-variance tradeoff).
- **Over-Parameterized Models**: Surprisingly, further increasing complexity still **reduces test error**, defying traditional overfitting concerns.
- **Critically-Parameterized Models**: At critical points, increasing complexity can either **increase or decrease** test error, leading to unpredictable behavior.

- In the **under-parameterized regime**, test error follows a U-shaped curve as model complexity increases, in line with classical bias-variance tradeoff predictions.

Autoencoding: nonlinear dimension reduction

- Autoencoders are a type of unsupervised learning model that perform dimensionality reduction by compressing input data into a lower-dimensional representation, then reconstructing the original input from this compressed version.
- This process is highly useful in extracting latent features from complex data, such as in asset pricing applications.
- Autoencoders consist of two main steps:

- **Encoding:** The input data is compressed into a smaller, lower-dimensional representation. The goal is to capture the most important features of the data using fewer variables.
- **Decoding:** The compressed representation is then transformed back into the original input space. This step evaluates how well the autoencoder can reconstruct the input data, making it useful for tasks like denoising.

Comparision with PCA

- Both autoencoders and Principal Component Analysis (PCA) perform dimensionality reduction, but there are key differences:
- **PCA** is a linear technique that reduces dimensions by projecting data onto orthogonal axes, retaining as much variance as possible.
- **Autoencoders**, by contrast, use nonlinear neural networks. This enables them to capture more complex patterns in the data that PCA may miss.
- **Nonlinearity through Neural Networks**
- The use of **nonlinear activation functions** like ReLU (Rectified Linear Unit) in autoencoders allows them to map input data through complex transformations, making them more versatile than PCA. For example, in financial data where relationships between variables are often nonlinear, autoencoders can uncover hidden factors that explain more variance than traditional linear methods.
- **Loss Function and Reconstruction Error**
- The primary goal of the autoencoder is to minimize the **reconstruction error**, which measures how different the original input is from the reconstructed output. This is achieved by training the network to optimize a loss function, typically the mean squared error between the input and the output.

Applications in Asset Pricing

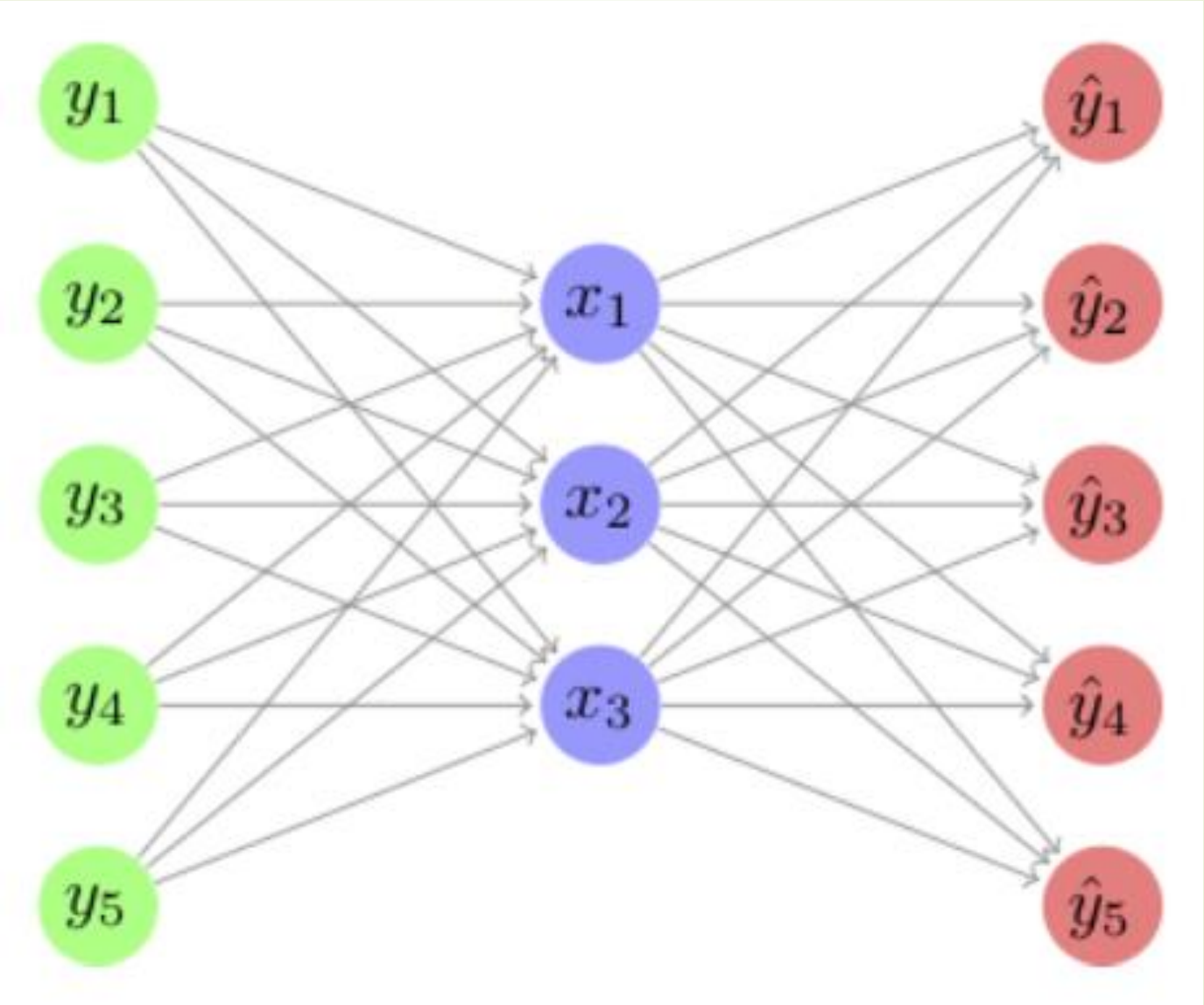
- In asset pricing, autoencoders can be used to extract nonlinear factors that are better suited for complex market behaviors than traditional factor models.
- These latent factors, derived from the encoding process, capture hidden relationships between stock returns and firm characteristics, leading to better predictions and a deeper understanding of asset risk.
- The next slide provides a clear implementation of the routine in a simple one-layer setup.
- The next-next slide provides a color scheme that helps visualize the compression (encoding) and decompression (decoding) processes that are core to the autoencoder structure.

The structure of an autoencoder

- **Green Circles (Input Layer)**: These circles represent the original input variables or features. In the context of asset pricing, these inputs could be the returns or characteristics of multiple assets that are to be compressed.

1. **Purple Circles (Hidden Layer)**: These circles represent the hidden layer neurons. This hidden layer performs the "encoding" step, where the input data is transformed into a lower-dimensional, compressed representation. The use of nonlinear activation functions (like ReLU) allows the autoencoder to capture complex patterns in the data.
2. **Red Circles (Output Layer)**: The red circles represent the "decoded" output, which is a reconstruction of the input variables. The goal is for the output to closely match the original input after passing through the bottleneck (hidden layer), minimizing reconstruction error.
3. **Transition Between Layers**:
    1. The transition from the green to purple layer represents the **encoding** process, where the input is compressed into latent factors.
    2. The transition from purple to red circles represents the **decoding** process, where the compressed representation is expanded back into the original form.

Autoencoding: unconditional asset pricing



Autoencoding with Multiple Layers

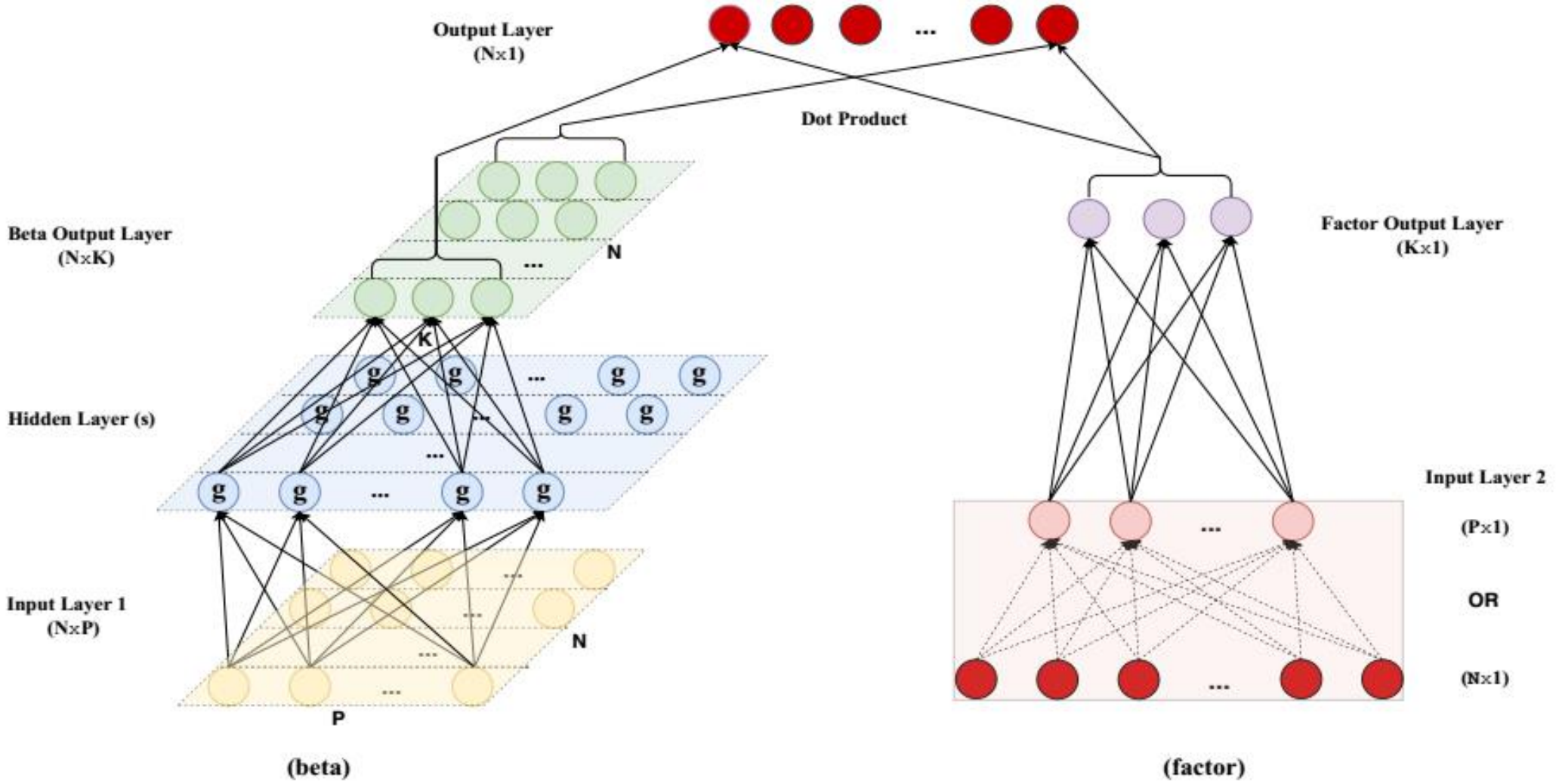
- There may be cases where an autoencoder includes multiple hidden layers. When this happens:
- The **encoding layers** (compressing the data) have a decreasing number of neurons as the model goes deeper, condensing the input data into fewer dimensions.
- Conversely, the **decoding layers** (reconstructing the data) have an increasing number of neurons as they attempt to recover the original data from the compressed representation.
- In this setup, the **encoding phase** identifies the essential lower-dimensional latent factors from the input data. These latent factors are key to understanding the underlying patterns.
- In the particular example provided, there are **three latent factors (K = 3)**, which means the autoencoder has reduced the input to just three key features.
- The model is **unconditional**, meaning that the **factor loadings** (which map input features to latent factors) are fixed and do not change over time.
- The model's parameters are optimized by **minimizing the sum of squared errors** between the actual returns and the reconstructed returns, ensuring that the model accurately captures the key features while reducing dimensionality.

Conditional Autoencoding (CA)

- Gu, Kelly, and Xu (2019) implement autoencoding in a setup where betas vary non-linearly with firm characteristics.
- Beta variations characterize conditional asset pricing models.
- The conditional autoencoder extends IPCA in which loadings are a linear function of firm characteristics.
- So, we have the two pairs {***PCA, autoencoder***} and {***IPCA, CA***) reflecting linear versus nonlinear activation functions, while the first (second) pair refers to unconditional (conditional) asset pricing.
- The figure on the next page describes CA. Source: Gu, Kelly, and Xu (2019).
- The left side of the network models factor loadings as a nonlinear function of predictive characteristics, while the right-side network formulates factors as portfolios of individual stock returns.
- Let us start with the left-hand-side.
- The yellow level describes a panel of predictive characteristics – N stocks P characteristics per stock.
- Characteristics are transformed through s hidden layers to form intermediate outputs – factor loadings.
- The right side is about constructing factors through autoencoder.
- Factors and loadings are interacted to form the eventual output layer – predicted returns.

CA: beta pricing

Figure 2: Conditional Autoencoder Model



Recurrent NN (RNN)

- A significant extension of neural networks (NNs) is the recurrent neural network (RNN).
- To illustrate, imagine observing a snapshot of a flying ball and being asked to predict its future location.
- Without prior information about its motion or history, any prediction would be purely a guess.
- The flying ball is an example of a sequence, where past information influences future outcomes.
- Other common examples include:
- Audio, which is a sequence of sound waves;
- Text, a sequence of characters or words;
- Genetic data and EKG signals are also sequential.

➢ In all these cases, a sequence is defined by its order: one event follows another.

Recurrent NN (RNN)

- In financial economics, time-series data with short- or long-run dependencies are examples of sequences.
- For instance, asset returns exhibit short- and long-term serial dependencies, such as short-term reversals, long-term reversals, and intermediate-term momentum.
- Can traditional NNs predict the outcome of sequences?
- Standard NNs lose effectiveness when dealing with sequentially dependent inputs.
- This is because they map a fixed and static input into a fixed and static output, ignoring the temporal relationships between data points.
- To address this, deep sequence models like RNNs are used.
- RNNs take into account the temporal dimension by relating the output to both (i) the current input and (ii) the prior history, stored as a latent cell state.

Recurrent NN (RNN)

- The current cell state in an RNN depends on both the input and the past state

$$h_t = G(x_t, h_{t-1})$$

- For instance,

$$h_t = tanh(W_{hx}x_t + W_{hh}h_{t-1} + b_h)$$

- The output is then given by

$$\hat{y}_t = W_{yh}h_t + b_y$$

- In an RNN, only the last hidden state is used to generate the output at time t.
- The loss is the forecast error, or the difference between y_t and $\hat{y}_t$.
- The total loss is the sum of forecast errors squared throughout the sample and experiments (e.g., N stocks).
- Estimate the model parameters by minimizing the loss function.
- The model parameters are ***common*** across the sequence.
- There are three weight matrices

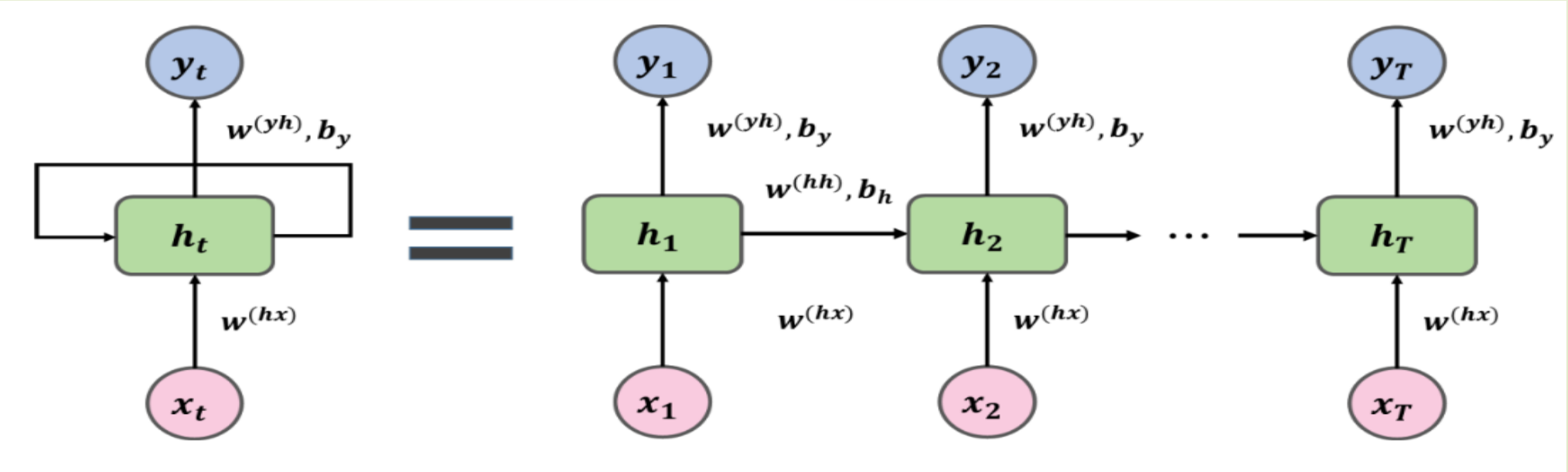
  W_{hx} defines how the inputs at each time step are being transformed

  W_{hh} defines the relationship between the prior and current hidden states

  W_{yh} transforms the hidden state to the output at a particular time step

Simple RNN structure (left side) and its unfolded representation (right side)

- A key feature of RNNs is their ability to handle sequences of arbitrary length.
- For example, consider predicting the next word in a text.
- In a feedforward neural network (FFN), the input text must be of a fixed size (e.g., 30 words), which complicates processing longer or shorter sequences.
- While the last word is critical, its importance depends on its relationship with prior words in the sequence—a relationship that FFNs cannot handle effectively.
- In contrast, RNNs process sequences step-by-step, making them versatile for predicting word sequences, stock returns, and other financial applications.



RNN – summary

- **RNNs** can be thought of as analogous to traditional time-series analysis in econometrics, as both aim to model patterns in sequential data.
- However, RNNs are more flexible, as they uncover these patterns in a data-driven and highly nonlinear manner, often involving many hidden states.
- In summary, RNNs:

a. Handle variable-length sequences, making them versatile for different types of sequential data;

b. Capture long-term dependencies, though this may be challenging for vanilla RNNs, which can suffer from vanishing gradients;

c. Retain information about the order of elements within a sequence, which is essential for tasks requiring sequential context;

d. Share parameters across the sequence, allowing the model to generalize across different parts of the sequence effectively.

RNN – a caveat

- **Feed-forward networks** are trained using the backpropagation algorithm. The process involves taking a set of inputs, making a forward pass through the network (from input to output), and then adjusting the weights through backpropagation. Specifically, the derivative of the loss with respect to each weight parameter is calculated, and the weights are updated to minimize the loss function.
- In **RNNs**, the forward pass occurs through time, and backpropagation is also through time, referred to as **backpropagation through time** (BPTT). This means errors are propagated back from the most recent time step to the beginning of the sequence, tracing through all previous steps. As this happens, the gradients are multiplied by the same weight matrix repeatedly.
- This recursive multiplication introduces a significant challenge: **exploding gradients** or, more commonly, **vanishing gradients**.

RNN – a caveat

- To illustrate:
- If you keep multiplying 0.9 by itself, the sequence eventually approaches zero (vanishing gradient).
- Conversely, multiplying 2 by itself leads to an explosive increase (exploding gradient).
- When the gradients become too small, the network struggles to learn long-term dependencies because the contribution of earlier time steps diminishes rapidly. This results in **biases toward short-term dependencies**, even when long-term ones are important. On the other hand, if the gradients explode, the model becomes unstable and fails to converge.
- Due to this **vanishing gradient problem**, standard RNNs are often ineffective at learning from long-range dependencies in sequences. This issue motivated the development of more sophisticated architectures, like **Long Short-Term Memory (LSTM)** and **Gated Recurrent Units (GRUs)**, which are designed to mitigate these challenges by preserving information over longer periods in a sequence.

LSTM: Maintaining Gradient Value in Backpropagation

- **Problem with RNNs**: As noted, in traditional RNNs, gradients tend to vanish or explode during backpropagation, especially over long sequences, which limits their ability to capture long-term dependencies.
- **Solution – LSTM Cell**:
- **LSTM** replaces the simple RNN cell with a more sophisticated structure that includes multiple gates:
  - **Forget Gate**: Controls how much of the past information to stored.
  - **Input Gate**: Decides which parts of the new input to incorporate into the memory.
  - **Output Gate**: Determines how much of the cell's state to pass to the output.
- This structure allows LSTM to effectively **preserve gradients**, enabling learning over longer periods.

LSTM: Maintaining Gradient Value in Backpropagation

- **Input-Output Transformation**:
- LSTM transforms **time-series inputs** (of dimension *d*) into **time-series outputs** (of dimension *h*).
- This ability to manage dependencies across time makes LSTM especially suited for financial time series with **low signal-to-noise** ratios, which often exhibit both short-term volatility and long-term trends.
- **Recent Advances**:
- **Attention Mechanisms**: More recent models, such as **Transformer architectures**, leverage attention mechanisms to focus on relevant parts of the input sequence, allowing them to outperform LSTMs in many tasks, including natural language processing (NLP) and translation.
- **Relevance to Finance**:
- Despite these advances, LSTM remains a valid approach for predicting stock returns and financial outcomes, especially when signal-to-noise ratios are low.
- For a detailed comparison of deep sequence models, refer to *"Deep Sequence Modeling: Development and Applications in Asset Pricing"* by Cong, Tang, Wang, and Zhang (2020), which compares various deep sequence models in predicting future returns.

Key Parts of an LSTM Cell:

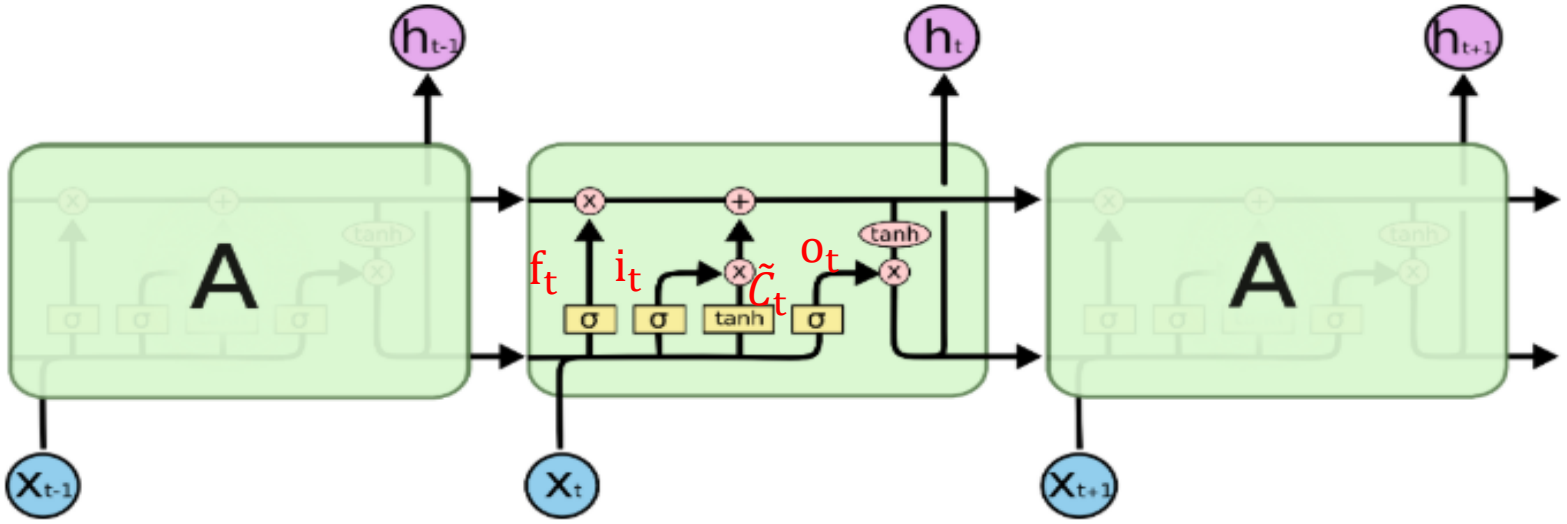
- A **cell state (c)**– Represents long-term memory, holding all accumulated learning from previous steps.
- Three regulators (“gates”) that control the flow of information inside the LSTM unit:
  - **Input gate (i)** -Controls how much of the new information should be allowed into the cell state.
  - **Forget gate (f)** -Decides which parts of the previous cell state should be discarded.
  - **Output gate (o)** - Determines what information should be passed to the next time step as output.

LSTM Unit functionality

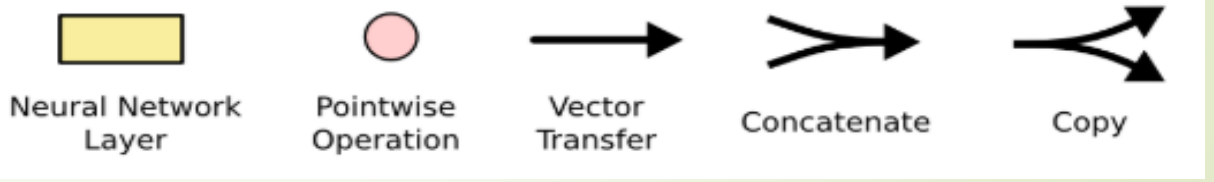
- The gates regulate the flow of information into and out of the cell, ensuring the LSTM can maintain long-term dependencies while filtering out irrelevant details.
- **In particular,**
- **Forget Irrelevant Information**: The forget gate decides how much of the previous memory should be **kept**. It controls what portion of the previous cell state continues to the next step and what portion is discarded.
- **Store Relevant New Information**: The input gate decides what new information to retain.
- **Update the Cell State**: The cell state gets updated with the combination of previous and new information.
- **Generate Output**: The output gate decides what part of the memory and new input should be output at the current time step.

A schematic figure for an LSTM unit

- x_t is the input consisting of time series observations
- h_t is an hidden state
- c_t is the long term memory, maintaning new relevant information while discarding irrelevant information
- i_t, o_t, and f_t are the gates
- $\widetilde{C}_t$ is the ***candidate*** cell state.



The repeating module in an LSTM contains four interacting layers.



The LSTM – Math Representation

- The functions are defined as follows

$$f_t = \sigma_g\left(W_f x_t + U_f h_{t-1} + b_f\right)$$
$$i_t = \sigma_g(W_i x_t + U_i h_{t-1} + b_i)$$
$$\tilde{C}_t = \tanh(W_c x_t + U_c h_{t-1} + b_c)$$
$$\boldsymbol{C_t = f_t \circ C_{t-1} + i_t \circ \tilde{C}_t}$$
$$o_t = \sigma_g(W_o x_t + U_o h_{t-1} + b_o)$$
$$\boldsymbol{h_t = o_t \circ \tanh(C_t)}$$

where $\circ$ is an element-by-element product, and C_0 and h_0 are initial values (can set to zero)

- The new cell state is updated by two main components:

1. A portion of the previous cell state, which is kept or discarded based on the decision of the forget gate. This gate controls how much of the old information should be retained.
2. The new candidate cell state, which is introduced through the input gate. The input gate controls how much of the new information should be added to the cell's memory.

Variables: $x_t \in \mathbb{R}^d$ is the intput vector, $f_t \in \mathbb{R}^h$ is the forget gate, $i_t \in \mathbb{R}^h$ is the input update gate, $o_t \in \mathbb{R}^h$ is the ouptut gate, $\tilde{C}_t$ $(c_t) \in \mathbb{R}^h$ is the cell input (state) vector, $W - s \in \mathbb{R}^{h \times d}$ and $U - s \in \mathbb{R}^{h \times d}$ are weight matrices, and b−s $\in \mathbb{R}^h$ are intercept vectors, all of which are learnt during the training stage, and d is the pre-specified number of input feature. Input and output can have different dimensions.

LSTM dynamics and hyperparameters

- **Role of Gates**:
- **Larger *f***: Indicates a decision to retain more of the previous cell state, giving greater importance to past information.
- **Larger *i***: Implies more weight is given to the new input, allowing the current values to have a stronger impact on the cell state.
- **Independence of *f* and *i***: They function separately, meaning they do not act as explicit complementary weights, and their weights in the cell state do not sum up to one.
- **Hyperparameters**:
- ***h***: Represents the number of hidden units, which controls the output dimension of the LSTM. This is a key hyperparameter that defines the model's capacity to learn patterns.
- **LASSO Regularization**: Adding LASSO during optimization introduces another hyperparameter, which controls how much regularization is applied to prevent overfitting.
- **Hyperparameter Tuning**: validation sample.

Key Aspects of LSTM Architecture

- **Maintain a Separate Cell State:** The cell state is updated across time steps to store long-term information.
- **Gates to Control Information Flow:**
  - **Forget Gate:** Discards irrelevant information.
  - **Input Gate:** Stores relevant information from the current input.
  - **Cell State Update:** Selectively updates the cell state based on the input gate.
  - **Output Gate:** Returns a filtered version of the cell state as output.
- **Backpropagation Through Time:** Ensures uninterrupted gradient flow for learning long-term dependencies.

➨ **Three Important Caveats About LSTM:**

1. **High Memory Requirement:** Due to its complex architecture, LSTMs require more memory.
2. **Training Challenges:** LSTMs face difficulties during training because of long gradient paths, similar to training a 100-layer neural network on a 100-word document.
3. **Activation Functions:**
   1. **Sigmoid & Tanh:** Can be difficult to work with due to saturation issues.
   2. **ReLU (Rectified Linear Unit):** Less sensitive to random initialization, allowing neurons to express strong opinions.

LSTM: Example and factor extraction

- Let firm *i* have *M* characteristics whose time *t* realizations are denoted by $x_t^{(i)}$.
- At time *t*, we use the most recent *K* periods to predict the next period return $\hat{r}_{t+1}^{(i)}$.
- The input is the series $\left\{x_{t-K+1}^{(i)}, \ldots, x_t^{(i)}\right\}$ while the final output, $h_t^{(i)}$ generates the prediction $\hat{r}_{t+1}^{(i)}$.
- LSTM parameters are estimated by minimizing the loss function

$$\mathcal{L} = \sum_{i=1}^{N} \sum_{t=1}^{T} \left(r_{i,t+1} - h_t^{(i)}\right)^2$$

- In that setup, it is assumed that the output is a single number per stock, predicted return.
- If $h_t^{(i)}$ is a vector of length *h*, the output is given by $\tilde{h}_t^{(i)} = W_h h_t^{(i)}$, while $\tilde{h}_t^{(i)}$ is replacing $h_t^{(i)}$ in the loss function above.
- Notice that the estimated parameters are identical across stocks.

LSTM factors

- LSTM can also be applied to a large set of macro variables where lower-dimension state cells summarize the short and long-run dependencies.

Hidden State (h_t) as Dynamic Factors:

- The hidden state in an LSTM cell represents the output at each time step, capturing the most relevant features from the input sequence.
- The hidden states can be interpreted as a set of **dynamic factors** that evolve, reflecting immediate influences on the time-series data, such as market sentiment or short-term economic fluctuations.

Cell State (C_t) as Long-Term Factors:

- The cell state carries the long-term memory of the LSTM, accumulating and retaining information across time steps.
- The cell state can be viewed as encapsulating **long-term factors**, such as persistent macroeconomic trends or underlying market conditions, that affect the sequence over a more extended period.

Factor Extraction through LSTM

- Unlike traditional linear factor models, LSTMs can capture complex, nonlinear dependencies in the data, providing a richer, more refined understanding of the underlying factors.
- By processing sequences of macro predictors or other time-series data, LSTMs dynamically extract and update factors that are crucial for predicting future outcomes.

➨ **Complementing Other Factor Models**:

- **PCA**: Linear extraction of static factors.
- **IPCA**: Instrumental PCA with varying loadings based on characteristics.
- **Autoencoder**: Nonlinear, unsupervised factor extraction.
- **CA**: Conditional Autoencoder capturing nonlinear conditional relationships.
- **LSTM**: Adds the temporal dimension, capturing both short-term and long-term dependencies in a dynamic, data-driven manner.

RNN and LSTM – attention mechanism

- Let us revisit RRN, recall that the latent states are denoted by $h_1, h_2, \ldots, h_T$.
- Notice that h_t is perceived to contain all the abstract features in the entire sequence.
- As all hidden states before t are not involved directly in generating the output, the old information is washed out after being propagated over multiple time steps.
- LSTM has the same drawback.
- To address this issue, the attention mechanism is proposed (Chaudhari et al., 2019).
- With attention, not only h_t but also all the hidden states play a role.
- I will later explain the attention mechanism in the context of TE.

Machine Learning versus Economic Restrictions

- Are machine learning methods lead to better forecasts of future stock returns?
- Avramov, Cheng, and Metzker (2021) show that deep learning signals, such as those described earlier, confront similar caveats as individual anomalies.
- Investment performance considerably deteriorates when distressed stocks are excluded.
- Likewise, performance is vastly stronger during high limits-to-arbtirage market states, such as high market volatility (VIX).
- In addition, the stochastic discount factor estimated by NKS is taking extreme long and short positions that cannot be practically implemented in real-time.
- As noted by Chen, Pelger, and Zhu (2021), it is a natural idea to use machine learning techniques, such as deep neural networks, to deal with the high dimensionality and complex functional dependencies of input data.
- However, machine-learning tools are designed to work well for prediction tasks in a high signal-to-noise environment.
- As asset returns seem to be dominated by unforecastable news, it is hard to predict their risk premia with off-the-shelf methods.
- The presentation of the Avramov et al (2021) paper is in the appendix to the class notes.

Machine Learning versus Economic Restrictions

- On the other hand, Avramov, Kaplanski, and Subrahmanyam (2021) apply LASSO, Ridge, and Elastic Net techniques (shallow learners) to newly defined variables and document robust performance.
- In particular, they consider all COMPUTAT items and compute, per item, the distance between current values and moving averages over past quarters.
- Such deviations predict future stock returns to economically significant degrees.
- The rule based on their Fundamental Deviation Index (FDI) survives recent years, excluding microcaps, long positions only, all market states, and reasonable trading costs.
- They attribute their findings to investor's anchoring.
- Avramov, Cheng, Metzker, and Voigt (2021) show the robust prediction ability of Bayesian Model Averaging (BMA).
- Asset pricing inferences in BMA draw on an integrated model that weights individual models based on posterior probabilities.
- Is there hope for deep learning signals?
- In what follows, two state-of-the-art approaches are explained, Reinforcement Learning (RL) and adversarial GMM – both can be potentially useful.

Reinforcement Learning (RL)

- RL is based on two key ideas: **trial-and-error learning** and **delayed reward**.
- It focuses on **solving problems** rather than using specific methods.
- Any method that fits the problem can be considered RL.
- **Difference from Supervised Learning**:

- In supervised learning, labeled examples are provided by an external supervisor.
- RL involves learning from interaction with the environment, where the agent must explore and exploit based on feedback.

- **Exploration vs Exploitation**:

- RL agents need to balance exploiting known rewards and exploring unknown actions for potentially better outcomes.
- This is especially important in dynamic and uncertain environments

Core Elements of RL

- **Policy**: The agent's strategy for selecting actions based on the current state.
- **Reward Function**: Defines the immediate feedback for an action, signaling good or bad outcomes.
- **Value Function**: Estimates the long-term reward from a state, considering future potential rewards.

Q Learning

- A model-free RL algorithm where the agent learns the optimal action for each state through trial and error.
- The agent updates its "Q-values" (estimates of the action's value) based on feedback from the environment.

➨ **Example**:

- A chess-playing agent improves over time by learning which moves lead to victories (rewards) and which do not, adjusting its policy to maximize long-term success.

Reinforcement Learning in Asset Pricing

- **Reinforcement Learning (RL) in asset pricing** was pioneered by Wang, Zhang, Tang, Wu, and Xiong (2021).
- RL incorporates three components for **return prediction**, while the RL aspect focuses on forming **optimal portfolios** based on these predictions.

1. **Step 1**: Use **LSTM** or **Transformer-Encoder** to generate a representation for each asset based on its historical state.
2. **Step 2**: Introduce a **Cross Asset Attention Network (CAAN)**, which utilizes these asset representations to extract interrelationships among the assets.
3. **Step 3**: Employ a **portfolio generator**, which uses a scalar winner score for each asset from CAAN to derive optimal portfolio weights.

- RL models the joint distribution of asset returns, observing trading actions, testing a range of actions (portfolio weights), and exploring a high-dimensional parameter space to **maximize the Sharpe ratio**.

Transformer Encoder (TE)

- **LSTM** is effective but limited due to its sequential processing and lack of an attention mechanism.
- The **TE** is a deep learning model that incorporates an attention mechanism, which assigns different levels of importance to each part of the input data.
- **Applications**: TE is widely used in fields such as **Natural Language Processing (NLP)** and **Computer Vision (CV)**, including tasks like self-driving cars and interactive gaming.
- Like **RNNs**, Transformer Encoders aim to process sequential input data.
- **Unlike RNNs**, TEs do not process the data in a strict sequence order, allowing for more flexibility.
- The **attention mechanism** provides context to any position within the input sequence, making it highly efficient for complex tasks.
- **Example**: In a sentence, the Transformer does not need to process the beginning before the end—it can analyze the entire input simultaneously.

TE: Key Features

- Instead of sequentially processing the data, the **TE** identifies the **context** that gives meaning to each word or element in the sequence.
- This feature allows for **parallelization**, which significantly reduces training times compared to sequential models like LSTMs.
- **TEs** have become the model of choice for **NLP** tasks, replacing LSTMs due to their efficiency and scalability.
- Parallel processing enables **TEs** to handle **much larger datasets**.
- **TEs** also overcome **vanishing** and **exploding gradient** problems, which are common in RNNs.
- The success of **transformers** has led to the development of **pre-trained models** such as **BERT** (Bidirectional Encoder Representations from Transformers) and **GPT** (Generative Pre-trained Transformer), both trained on massive datasets like Wikipedia and fine-tuned for specific tasks.

TE: Architecture

- The **transformer** uses an **encoder-decoder architecture**.
- The **encoder** consists of multiple layers that process the input data iteratively, one layer at a time. Similarly, the **decoder** processes the encoded output.
- Each encoder layer identifies which parts of the input are most relevant and generates a compressed representation.
- The output of each encoder layer is passed to the next layer for further processing.
- The **decoder layers** operate in reverse: they use contextual information to produce an output sequence from the encoded data.
- Both the **encoder** and **decoder** rely on the **attention mechanism**, which assigns different weights to parts of the input data based on relevance.
- For each input, attention determines which other inputs are most important for producing the final output.
- Each decoder layer incorporates an additional attention mechanism that references the output from previous decoder layers before drawing from the encoder.
- Both the **encoder** and **decoder** layers include feedforward neural networks, residual connections, and layer normalization for processing outputs.

TE in Finance

- **Common Applications in general**

1. Translating from one language to another.
2. Generating an answer (output) for a given question (input).

- In Finance, **Cong et al.** use only the **encoder** part of the TE.
- The authors also implement **LSTM** for time series encoding, using the Ct values to capture the sequence of stock characteristics.
- Both the TE and LSTM models are referred to as **Sequence Representation Extraction Models (SREM)** in the paper.
- The input for TE or LSTM is the **time series of stock characteristics**, which are encoded into a lower-dimensional representation.
- The **encoded data** is used to model the **cross-sectional interactions** between stocks using **CAAN** (Cross Asset Attention Network).
- These interactions are then converted into scores and weights to build portfolios aimed at achieving the highest **ex-post Sharpe ratio**.

TE in Finance

- The first part of the analysis consists of time-series encoding.
- Consider firm i characteristics over past K months

$$X^{(i)} = \left\{x_1^{(i)}, \dots, x_k^{(i)}, \dots, x_K^{(i)}\right\}$$

- In the Cong et al paper, the dimension of the input is K=12, reflecting a one year of monthly observations, while there are 51 firm characteristics.
- Thus, $x_k^{(i)}$ is a vector of dimension $d = 51$, while there are K=12 such vectors.
- Time-series observations are encoded by TE or LSTM into hidden states

$$Z^{(i)} = \left\{z_1^{(i)}, \dots, z_k^{(i)}, \dots, z_K^{(i)}\right\}$$

- It is a sequence-to-sequence encoding for each firm separately.
- The dimension of $z_k^{(i)}$ could be equal to or different from the dimension of $x_k^{(i)}$
- The hidden states attempt to capture long-range dependencies in the data.
- Below, I explain encoding – the transition from $X^{(i)}$ to $Z^{(i)}$ – through TE.
- LSTM encoding was explained earlier, the system cells (C) contain the encoded information.

TE: Self attention

- Again, the inputs for firm i are given by the matrix of characteristics

$$X^{(i)} = \left\{x_1^{(i)}, \ldots, x_k^{(i)}, \ldots, x_K^{(i)}\right\}$$

- The self-attention unit forms, for each $x_k^{(i)}$, **query** - $q_k^{(i)}$, **key** - $k_k^{(i)}$, and **value** - $v_k^{(i)}$ vectors.
- Specifically, the query, key, and value vectors are given by

$$\begin{aligned} q_k^{(i)} &= W^{(Q)} x_k^{(i)} \\ k_k^{(i)} &= W^{(K)} x_k^{(i)} \\ v_k^{(i)} &= W^{(V)} x_k^{(i)} \end{aligned}$$

- The dimensions of $q_k^{(i)}$ and $k_k^{(i)}$ are equal to d_1, which is is a hyperparameter.
- The dimension of $v_k^{(i)}$ is also a hyperparameter, say d_2, could be diffferent from d_1.
- Thus, $W^{(Q)}$ $W^{(K)}$ are $d_1 \times d$ matrices and $W^{(V)}$ is a $d_2 \times d$ matrix (recall, d=*51*).

TE: Self attention

Attention is defined as $Attention(Q,K,V)=softmax(\frac{Q'K}{\sqrt{d_k}})\,V$, where the product of the query and the key represents the relevance and the softmax function is

$$softmax(z_k) = \frac{exp(z_k)}{\sum_{k=1}^{K} exp(z_k)} \quad for\ k = 1, \dots, K$$

Then, the interrelationship between firm i characteristics at different times in the historical period $(k, k') \in 1, \dots, K$ is modeled by the dot product of query $q_k^{(i)}$ and key, $k_{k\prime}^{(i)}$

$$\beta_{k,k\prime} = \frac{q_k^{(i)\prime} k_{k\prime}^{(i)}}{\sqrt{d_1}}$$

The attention score for time k is a softmax of the interrelationship

$$a_k^{(i)} = \frac{\sum_{k\prime=1}^{K} \exp\left(\beta_{k,k\prime}\right) v_{k\prime}^{(i)}}{\sum_{k\prime=1}^{K} \exp\left(\beta_{k,k\prime}\right)}$$

Notice that the attention $a_k^{(i)}$ is a vector of dimension d_2 obtained as the weighted average (sum of weights is equal one) of the value.

Transformer– Self attention

- The output of the self-attention process can also be written using a matrix notation as

$$Z^{(i)} = \text{softmax}\left(\frac{X^{(i)\prime} W^{(Q)\prime} W^{(K)} X^{(i)}}{\sqrt{d_1}}\right) X^{(i)\prime} W^{(V)}$$

- $Z^{(i)}$s a $K \times d_2$ matrix that collects the K values of $a_k^{(i)}$
- The value of each position is calculated by all the positions in the sequence.
- The output is computed as a weighted sum of the values, where the weight assigned to each value is computed by a compatibility function of the query with the corresponding key.
- So, we can compute the attention function on a set of queries simultaneously, rather than period by period.
- The transition from $X^{(i)}$ to $Z^{(i)}$ is complete.
- Should do it stock by stock – query, key, and value matrices (W-s) are identical across stocks.

Transformer– Multi Head Attention

- In order to capture a number of complex interrelations in a sequence, self-attention units are grouped and connected in parallel.
- This connected group is termed multi-head attention.
- For instance, if it is about understanding a sentence, different people could have different perspectives on that sentence.
- The multi-head attention unit is a group of four (hyperparameter) self attention units each with different weights $W^{(Q)}$, $W^{(K)}$, $W^{(V)}$.
- Each self attention unit has an output $Z_h, h = 1, \ldots, 4$, which is a $K \times d_2$ matrix.
- Do the same thing 4 times ,and let the network learn four different items to pay attention to.
- All those matrices are concatenated into a new matrix.
- $\widetilde{Z} = [Z_1, \ldots, Z_4]$ is of dimension $K \times 4d_2$

Transformer– Multi Head Attention

- Then $\widetilde{Z}$ is multiplied from the right by a weight matrix W^o of dimension $4d_2 \times d$.
- The output of the multi-head attention is of dimension $K \times d$, which is (12 $\times$ 51) in our case:

$$head_j = Attention\left(QW_j^Q, KW_j^K, VW_j^V\right)$$

$$MultiHead(Q,K,V) = Concat(head_1, \dots, head_4)W^O$$

- Notice that $\widetilde{Z}$ is computed for each stock.
- Stocks only differ with about their inputs while the parameters are shared.
- The aim is to transform the observed stock characteristics into hidden states by encoding the characteristics through the TE mechanism with multi-head attention.
- Each attention head acts like a "different lens" through which the data is viewed, capturing various aspects of the relationships between the elements in the sequence.

Cross asset attention network (CAAN)

- Up to this stage, we have transformed the real observations $X^{(i)}$ into hidden states $Z^{(i)}$ for each stock.
- LSTM is an an alternative way to make that transformation, while the $Z^{(i)}$ are replaced by the cell states.
- The cross-sectional interrelations among firms' hidden states are modeled by a self attention module, termed cross asset attention network (CAAN), similar to the multi-head attention.
- All the vectors of hidden states for firm i, $Z^{(i)} = \left\{z_1^{(i)}, \dots, z_k^{(i)}, \dots, z_K^{(i)}\right\}$ are concatenated into a vector

$$y^{(i)} = Concat\left(z_1^{(i)}, \dots, z_k^{(i)}, \dots, z_K^{(i)}\right)$$

- $y^{(i)}$ is a **vector** with length $d\,K$ that represents firm i
- The representation $y^{(i)}$ is used to construct three other representation vectors: query - $q^{(i)}$, key - $k^{(i)}$ and value - $v^{(i)}$, using the trainable matrices $W_{\text{CAAN}}^{(Q)}$, $W_{\text{CAAN}}^{(K)}$, $W_{\text{CAAN}}^{(V)}$ (during the training process)

$$q^{(i)} = W_{\text{CAAN}}^{(Q)} y^{(i)}$$
$$k^{(i)} = W_{\text{CAAN}}^{(K)} y^{(i)}$$
$$v^{(i)} = W_{\text{CAAN}}^{(V)} y^{(i)}$$

- These matrices for query, key, and value are identical across assets.
- $W_{\text{CAAN}}^{(Q)}$, $W_{\text{CAAN}}^{(K)}$ are $d_3 \times (dK)$ matrices and $W_{\text{CAAN}}^{(V)}$ is a $d_4 \times (dK)$ matrix

CAAN

- The interrelationship between stock j and stock i is modeled by the dot product of stock i query, $q^{(i)}$ and stock j key,

$$\beta_{i,j} = \frac{q^{(i)\prime} k^{(j)}}{\sqrt{d_3}}$$

- Calculating attention score for stock i is a softmax of the interrelationship of stock i and j multiplied by stock j value

$$a^{(i)} = \frac{\sum_{j=1}^{I} \exp(\beta_{i,j})\, v^{(j)}}{\sum_{j=1}^{I} \exp(\beta_{i,j})}$$

where I is the overall number of stocks and $a^{(i)}$ is a vector with length d_4.

- Finally, the winner score determining the long short position of stock i in the portfolio is

$$s^{(i)} = sigmoid\left(W^{(S)} a^{(i)} + b^{(S)}\right)$$

where $W^{(S)}$ is a $1 \times d_4$ weight vector and $b^{(S)}$ is the bias.

Reinforcement Learning Optimization

- Given the scores of each stock $\{s^{(1)}, \ldots, s^{(i)}, \ldots s^{(I)}\}$ the long and short portfolios are constructed by the G extreme scores.
- Let $o^{(i)}$ be the rank of stock i in a descending order.
- Stock i is in the long portfolio b^{+} if $o^{(i)} \in [1, G]$

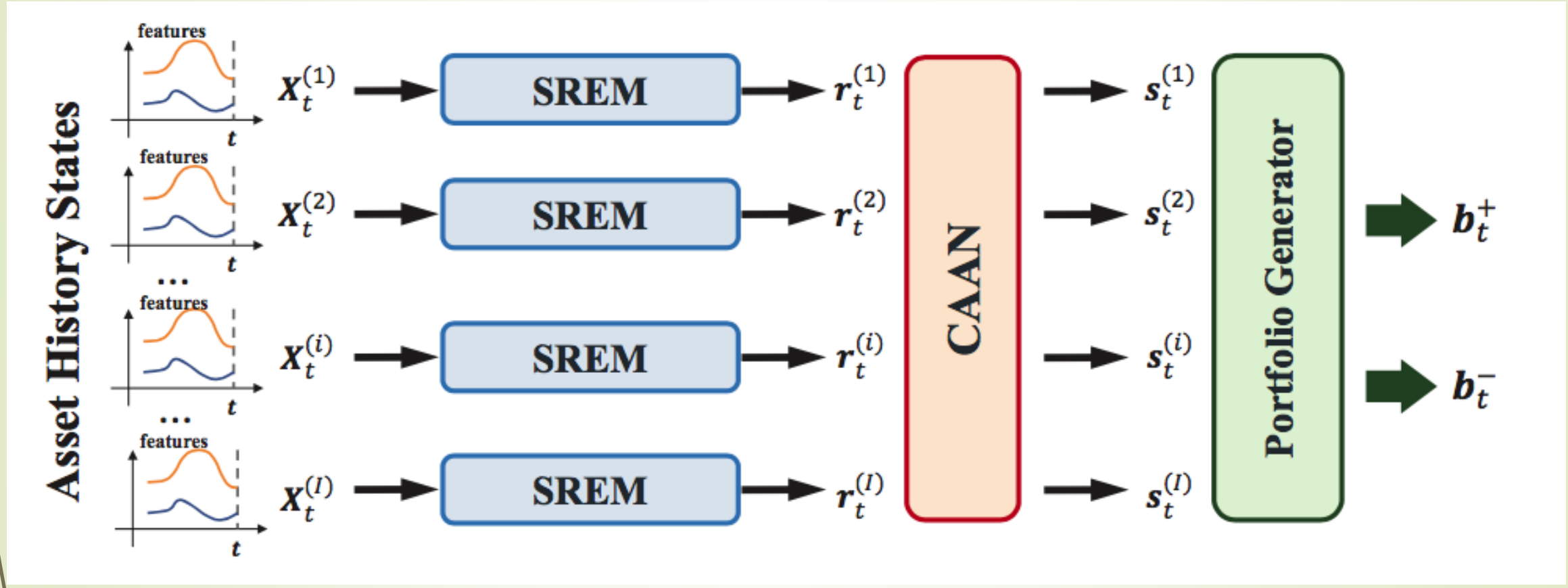
$$b^{+(i)} = \frac{exp\left(s^{(i)}\right)}{\sum_{o^{(i\prime)} \in [1,G]} exp(s^{(i\prime)})}$$

- Stock i is in the short portfolio b^{-} if $o^{(i)} \in [I - G + 1, I]$

$$b^{-(i)} = \frac{exp\left(-s^{(i)}\right)}{\sum_{o^{(i\prime)} \in [I-G+1,I]} exp(-s^{(i\prime)})}$$

- Let denote by b^{c} the vector of weights for all the I stocks.

RL: Forming portfolios through winner scores (s)



Reinforcement Learning

- "Learning what to do – how to map situations into actions – so as to maximize a numerical reward signal" (Sutton and Barto, 2008)
- The characteristics $X_t^{(i)} = \left\{x_1^{(i)}, \ldots, x_k^{(i)}, \ldots, x_K^{(i)}\right\}$ for the past K periods for $i = 1, \ldots, I$ is the state of environment.
- The action at time t is the stock weights in the constructed portfolio b_t^c.
- The reward at time t is the portfolio return at $t+1$: $r_{t+1}^p = r_{t+1}{}' \, b_t^c$
- The value is the Sharpe ratio for a sequence of realized returns, say 12 months

$$J = SR\left\{r_1^p, r_2^p, \ldots, r_{12}^p\right\}$$

Reinforcement Learning

- Let us denote by θ the model parameters that affect b_t^c, then the optimization is to find θ^* such that

$$\theta^* = \text{argmax}_\theta \, J(\theta)$$

- To summarize

$$\theta = \left\{ W_{\text{CAAN}}^{(Q)}, W_{\text{CAAN}}^{(K)}, W_{\text{CAAN}}^{(V)}, W^{(S)}, b^{(S)}, 4 \times \left(W^{(Q)}, W^{(K)}, W^{(V)} \right), W^o \right\}$$

CAAN parameters | TE parameters

- The collection of hyper-parameters $\{d, d_1, d_2, d_3, d_4, h, K, G\}$ is determined in the test sample.
- Because reinforced learning is not supervised, there is no training/validation samples.
- Hyperparameters are determined in the test sample by experimenting on different values.
- The investor is price taker; hence, his action does not affect the evolution of the environment.
- Another reinforcement learning techniques and applications in economics, game theory, operational research and finance can be found in Charpentier, Elie, and Remlinger (2020)

Stock return Prediction versus ChatGPT

- ChatGPT, like many advanced language models, relies on the Transformer architecture described earlier.
- Let's explore how the same principles apply to generating coherent and contextually appropriate responses in a conversation.
- **Understanding Transformers in ChatGPT**

1. **Sequential Data Handling**:
    - Just as the Transformer Encoder processes sequences of stock characteristics over time, ChatGPT processes sequences of words or tokens in a sentence or paragraph. Each word is treated as part of a sequence, with the model considering the context provided by all previous words to generate the next word.
2. **Self-Attention Mechanism**:
    - The self-attention mechanism in ChatGPT, like in predicting stock returns, allows the model to weigh the importance of different words in a sequence. For example, when predicting the next word in a sentence, the model looks at all previous words, determines which ones are most relevant to the context, and uses this information to generate the most appropriate next word.
    - In ChatGPT, the multi-head attention allows the model to understand different nuances of meaning and relationships between words, making it capable of generating contextually accurate and sophisticated language.

Stock return Predictions versus ChatGPT

3. Parallel Processing and Efficiency:

- One of the reasons ChatGPT is so powerful is because of the parallelization capability of transformers. Unlike traditional models that process data sequentially (like RNNs), transformers can process multiple parts of a sequence simultaneously. This is why ChatGPT can generate responses quickly and handle long conversations without losing context.

4. Encoding and Decoding:

- While in stock return prediction, we mainly discussed the encoding part of the Transformer, ChatGPT also relies on a decoding process. The encoded information (which represents the understanding of the input text) is used by the decoder to generate the response. The decoder works similarly by applying attention mechanisms to ensure the generated text is relevant to the input query.

5. Fine-Tuning and Adaptation:

- Just as you might tune a model's parameters to better predict stock returns, ChatGPT is fine-tuned on vast amounts of text data. This allows it to adapt to different types of questions, styles of conversation, and even specific topics, providing more accurate and relevant responses.

Stock return Predictions versus ChatGPT

To sum up, ChatGPT works by leveraging the same principles we've used in stock return prediction with transformers: it processes sequences of data (in this case, words), applies self-attention to understand context, and uses this information to generate coherent and contextually relevant responses.

The power of transformers, with their ability to handle long-range dependencies and process data efficiently, is what makes models like ChatGPT so effective in natural language processing tasks.

Generative Adversarial Network - GAN

- GAN is a setup with two neural networks contesting with each other in a (often zero-sum) game.
- For example, let w and g be two neural networks' outputs.
- The loss function is defined over both outputs, $L(w, g)$.
- The competition between the two neural networks is done via iterating both w and g sequentially:
  - w is updated by minimizing the loss while g is given
$$\hat{w} = \min_{w} L(w|g)$$
  - g is the adversarial and it is updated by maximizing the loss while w is given
$$\hat{g} = \max_{g} L(g|w)$$

Adversarial GMM

- Chen, Pelger, and Zhu (2019) employ an adversarial GMM to estimate the SDF
- The CPZ model is formulated as follows.
- For any excess return, no arbitrage suggests that

  $\mathrm{E}_t\left(M_{t+1}R^e_{t+1,i}\right) = 0$, where $M_{t+1} = 1 - \sum_{i=1}^{N} w_{t,i}R^e_{t+1,i}$ and $w_{t,i}$ is a general function
- It then follows that $\mathrm{E}_t\left(M_{t+1}R^e_{t+1,i}g(I_{t,i}, I_t\ )\right) = 0$
- That is because you can multiply the moment conditions with any time t measurable function of firm characteristics and macro variables.
- The unconditional representation follows from the LIE: $E\left(M_{t+1}R^e_{t+1,i}g(I_{t,i}, I_t\ )\right) = 0$
- The unconditional moment conditions can be interpreted as the pricing errors for a choice of portfolios and times, determined by $g(.)$.
- The challenge is to find the relevant moment conditions to identify the SDF.
- Considering only unconditional moments, the g function is constant.

Adversarial GMM

- Can use the adversarial approach to select the moment conditions that lead to the largest mispricing.
- This is a minimax optimization problem.

$$\min_{w}\max_{g}\frac{1}{N}\sum_{j=1}^{N}\left\|E\left[\left(1-\sum_{i=1}^{N}w\left(I_t,I_{t,i}\right)R^{e}_{t+1,i}\right)R^{e}_{t+1,j}g\left(I_t,I_{t,j}\right)\right]\right\|^{2}$$

where ω and g are normalized functions chosen from a specified functional class.

- These types of problems can be modeled as a zero-sum game, where one player, the asset pricing modeler, aims to choose an asset pricing model, while the adversary searches for conditions under which the asset pricing model performs badly.
- This can be interpreted as first finding portfolios or times that are the most mispriced and then tuning the asset pricing model to also price these assets.
- The process is repeated until the adversary cannot find portfolios with large enough pricing errors.

Adversarial GMM

- Note that this is a data-driven generalization for the research protocol conducted in asset pricing in the last decades.
- To illustrate, assume that the asset pricing modeler uses the Fama-French 5 factor model, spanned by the five factors.
- The adversary might propose momentum sorted test assets, that is g is a vector of indicator functions for different quantiles of past returns.
- As these test assets have significant pricing errors with respect to the Fama-French 5 factors, the asset pricing modeler needs to revise the candidate SDF, for example, by adding a momentum factor.
- Next, the adversary searches for other mispriced anomalies or states of the economy, which the asset pricing modeler will exploit in revising the SDF.

Adversarial GMM

- The adversarial estimation with a minimax objective function is motivated from the insights of Hansen and Jagannathan (1997).
- HJ show that if the SDF implied by an asset pricing model is only a proxy that does not price all possible assets in the economy, then minimizing the largest possible pricing error corresponds to estimating the SDF that is the closest to an admissible true SDF in a least square distance.
- HJ discuss the estimation of the SDF based on the minimax objective function and compare it with the conventional efficient GMM estimation for parametric models with a low dimensional parameter set.
- They conclude that the minimax estimation has desirable properties when models are misspecified and the resulting SDFs have substantially less variation relative to the conventional GMM approach.
- In CPZ, the SDF is implicitly constrained by the fact that it can only depend on stock specific characteristics $I_{i,t}$ but not the identity of the stocks themselves and by a regularization in the estimation.
- Hence, even in-sample, the SDF will have non-zero pricing errors for some stocks and their characteristic managed portfolios, which naturally puts CPZ into the setup of Hansen and Jagannathan (1997).

Adversarial GMM

- Choosing the conditioning function g correspond to finding optimal instruments in a GMM estimation.
- The conventional GMM approach assumes a finite number of moments that identify a finite dimensional set of parameters.
- In CPZ, there are an infinite number of candidate moments without the knowledge of which moments identify the parameters.
- The parameter set is also of infinite dimension, and, hence, there is not an asymptotic normal distribution with a feasible estimator of the covariance matrix.
- The approach thus selects the moments based on robustness.
- By controlling the worst possible pricing error, the approach aims to choose the test assets that can identify all parameters of the SDF and provide a robust fit.
- The conditioning function g generates a very large number of test assets to identify a complex SDF structure.

Adversarial GMM

- The moment conditions are averaged over the sample of all instrumented stocks, that is the loss function is $\frac{1}{N}\sum_{i=1}^{N}\sum_{d=1}^{D}\alpha_{i,d}^2$, where the moment deviation $\alpha_{i,d} = E\left[M_{t+1,}\, R_i g_d(I_t, I_{t,i})\right]$ can be interpreted as the pricing error of stock i instrumented by the element g_d of the vector valued function $g(.)$.
- Note that the instruments g_d are normalized to be in $[-1,1]$.
- In their benchmark model, CPZ consider $N = 10,000$ stocks and $D = 8$ instruments and therefore the total is 80,000 instrumented assets.
- Hence, the SDF depends only on information that affects a very large proportion of the stocks, amounting to systematic mispricing.
- This also implies that the adversarial approach will only select instruments that lead to mispricing for most stocks.

Adversarial GMM

- Once CPZ obtain the SDF factor weights, the loadings are proportional to the conditional moments $E_t\left[F_{t+1,}\, R^e_{t+1,i}\right]$.
- A key element of their approach is to avoid estimating directly conditional means of stock returns.
- CPZ show that they can better estimate the conditional co-movement of stock returns with the SDF factors, which is a second moment, than the conditional first moment.

Adversarial GMM

- The empirical loss function of the model minimizes the weighted sample moments which can be interpreted as weighted sample mean pricing errors:

$$L(\omega|\hat{g}, I_t, I_{t,i}) = \frac{1}{n}\sum_{i=1}^{N}\frac{T_i}{T}\left\|\frac{1}{T_i}\sum_{t\in T_i} M_{t+1}R^e_{t+1,i}\hat{g}(I_t, I_{t+1})\right\|^2$$

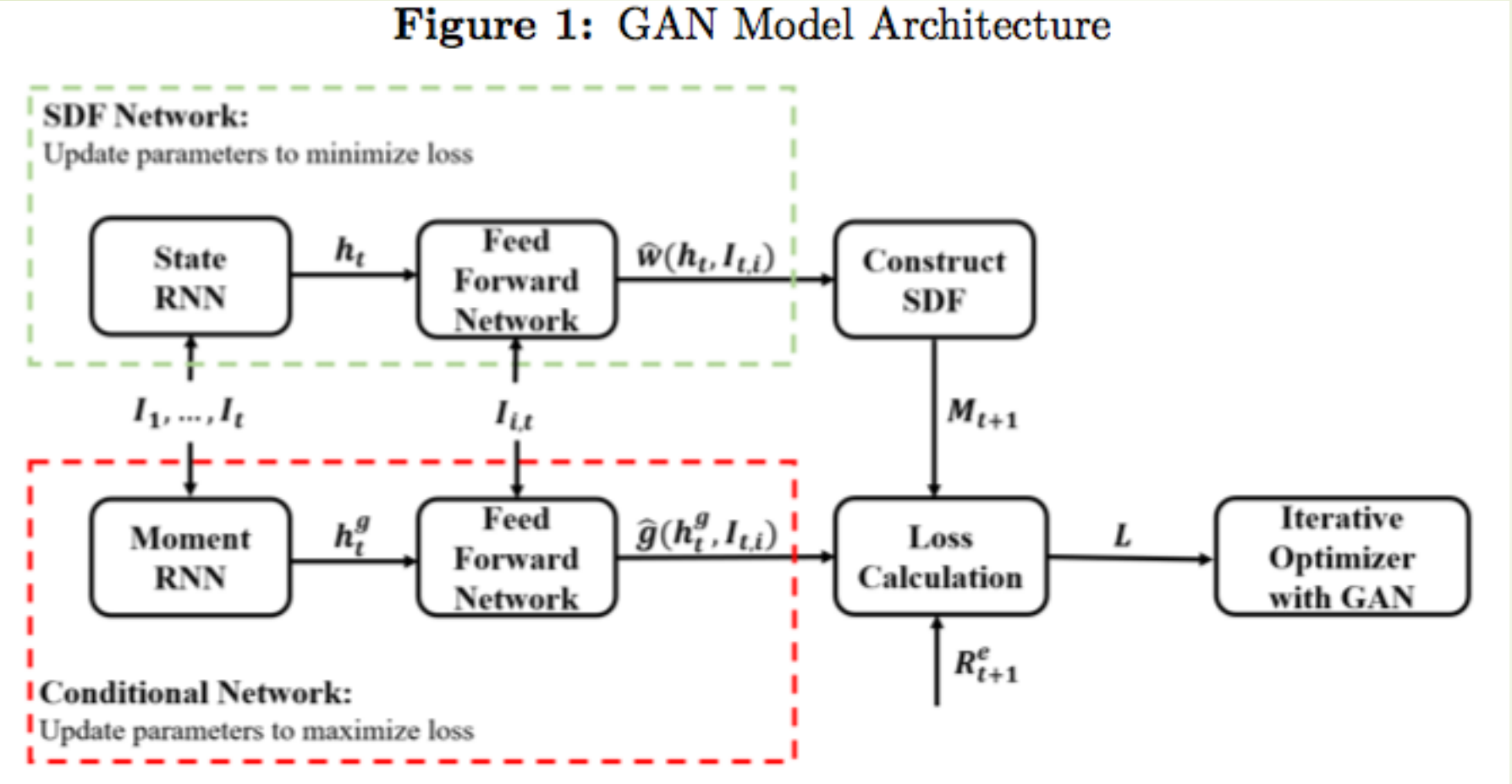
  for a given conditioning function $\hat{g}(.)$ and information set.

- As the convergence rates of the moments under suitable conditions is $1/T_i$, they weight each cross-sectional moment condition by $1/T_i$ which assigns a higher weight to moments that are estimated more precisely and down-weights the moments of assets that are observed only for a short time period.

Adversarial GMM

- For a given conditioning function $\hat{g}(.)$ and choice of information set the SDF portfolio weights are estimated by a feedforward network that minimizes the pricing error loss $\hat{\omega} = \min_{\omega} L(\omega \,|\hat{g}, I_t, I_{t,i})$
- This is the SDF network.
- CPZ then construct the conditioning function $\hat{g}$ via a conditional network with a similar neural network architecture.
- The conditional network serves as an adversary and competes with the SDF network to identify the assets and portfolio strategies that are the hardest to explain.
- The macroeconomic information dynamics are summarized by macroeconomic state variables h_t which are obtained by LSTM.
- The model architecture is summarized on the next page.

Adversarial GMM



Appendix

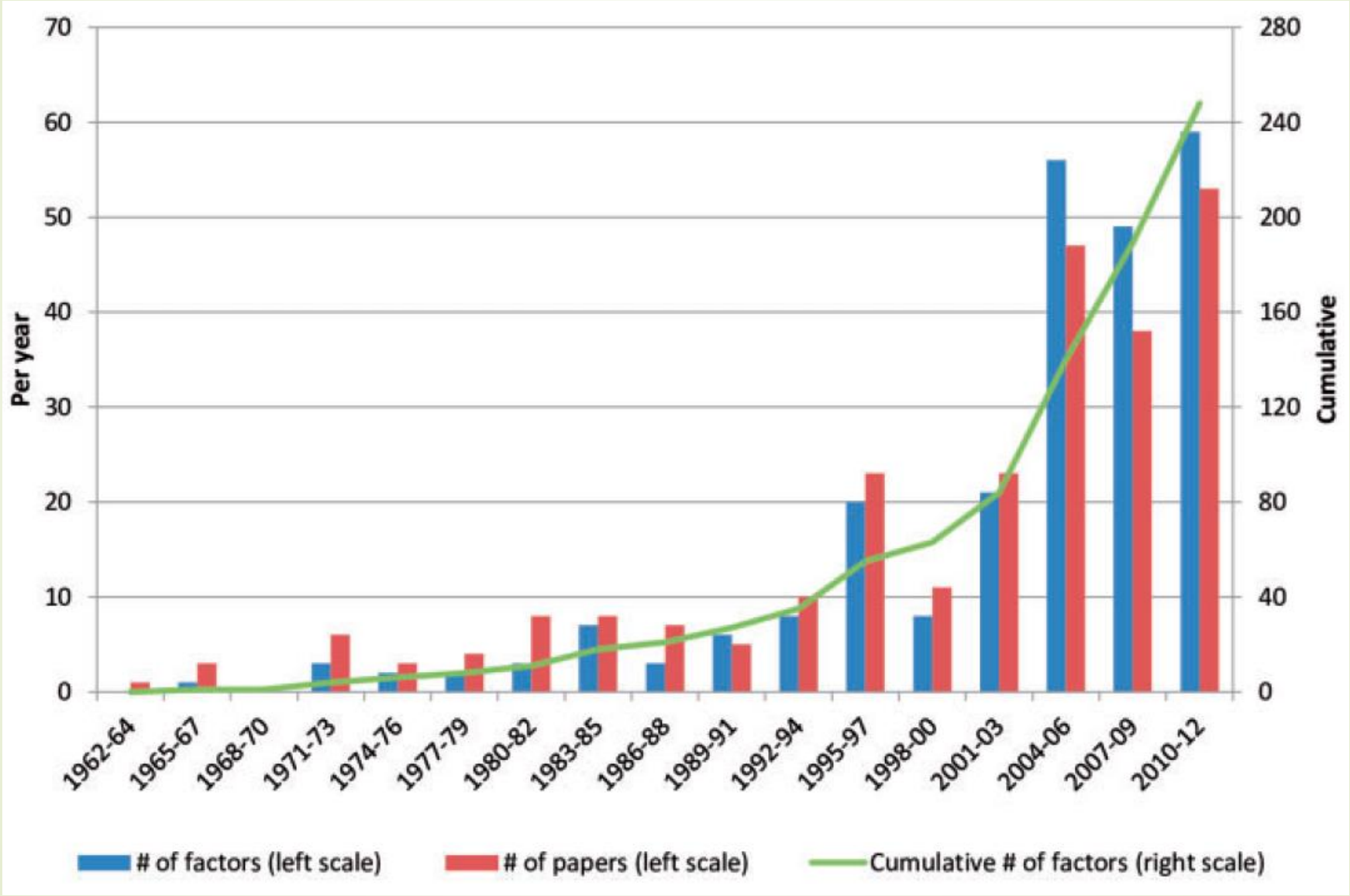
MACHINE LEARNING VERSUS ECONOMIC RESTRICTIONS: EVIDENCE FROM STOCK RETURN PREDICTABILITY

Doron Avramov, IDC Herzliya, Israel

Si Cheng, Chinese University of Hong Kong

Lior Metzker, Hebrew University of Jerusalem

Zoo of Anomalies



Source: Harvey, Liu, and Zhu (2016)

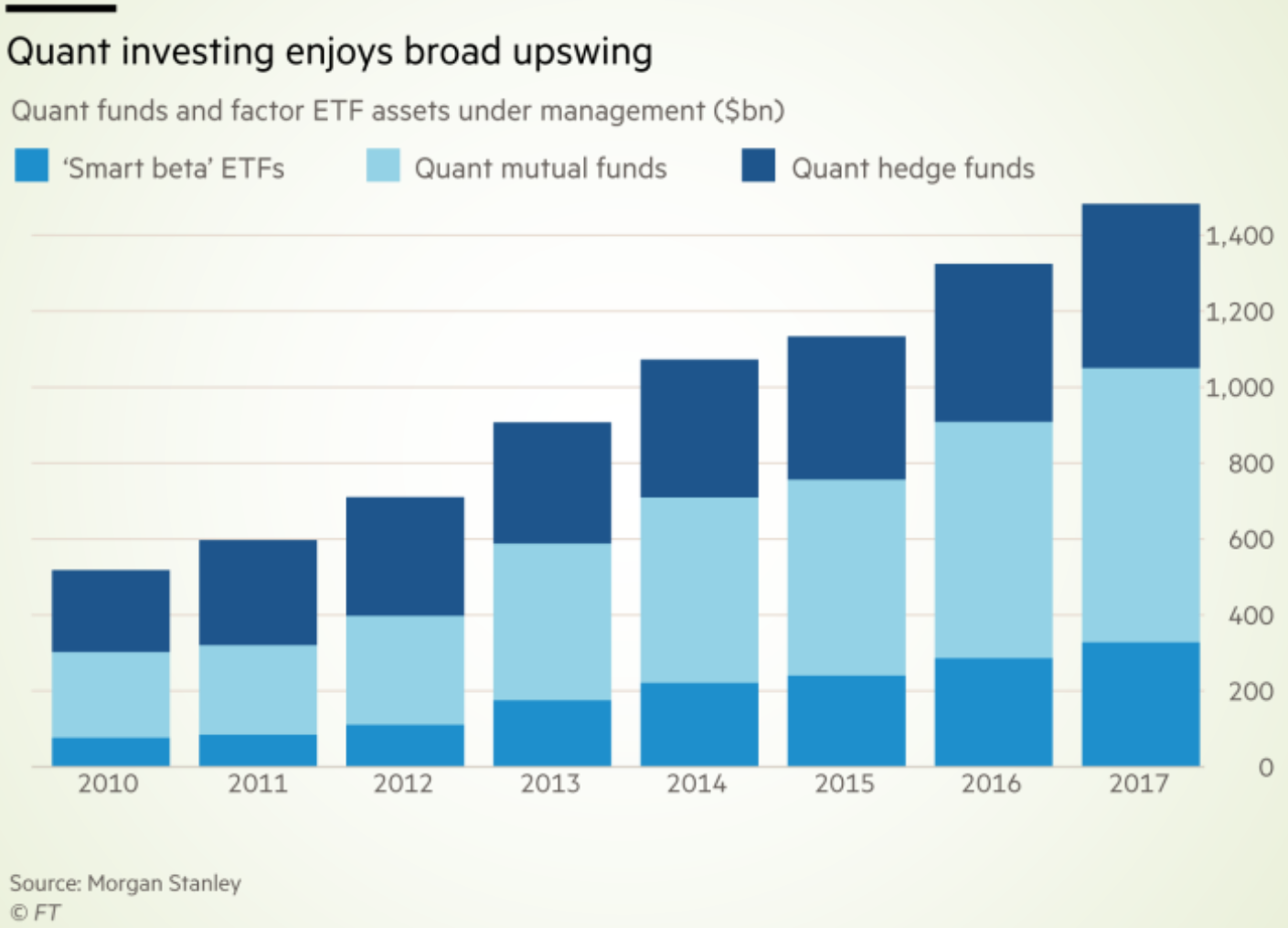
Zoo of Anomalies: Challenges

- Harvey, Liu, and Zhu (2016): 296 anomalies, 27% to 53% are likely to be false discoveries
- Hou, Xue, and Zhang (2020): 452 anomalies, 82% turn insignificant upon excluding microcaps + value-weighting
- Other challenges: anomaly profits mostly originate from short-leg distressed stocks and often disappear in recent years (e.g., Avramov, Chordia, Jostova, and Philipov, 2013)

Zoo of Anomalies: Challenges

- Traditional methods:
  - Portfolio sorts and cross-sectional regressions
  - Low-dimensional
- What needs to be done?
  - High-dimensional, noisy and correlated predictors
  - Flexible functional forms
  - Model selection
  - Mitigate overfitting biases
- Machine Learning: automated detection of complex patterns in data; combine multiple weak sources of information into a meaningful composite signal
- Growing literature on return prediction and asset pricing models

FinTech Adoption in Asset Management



Research Questions

- Two strands of literature: diminishing anomalies vs. increasing prominence of ML methods
- Do ML methods clear the common economic restrictions in asset pricing?
  - Exclude difficult-to-arbitrage stocks
  - Cannot infer from individual anomalies
- Does the return predictability of ML signals vary over time?
  - Exclude market states with high limits to arbitrage
  - Again, cannot infer…
- What are the economic grounds for the seemingly opaque ML methods?

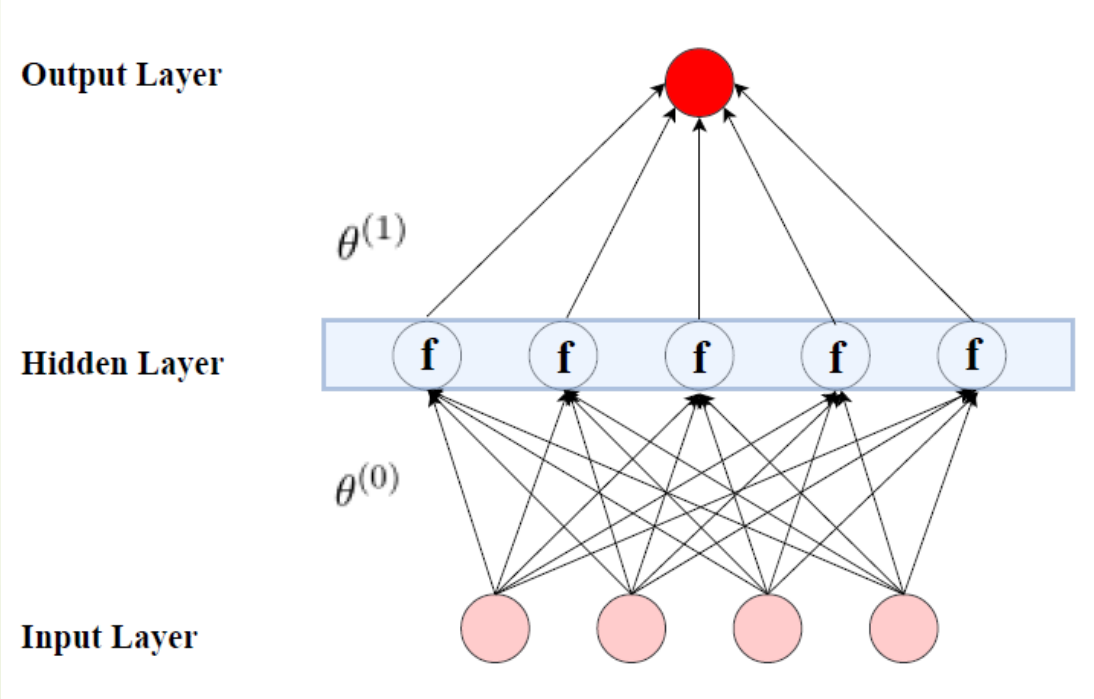
Summary of Machine Learning Methods

	Linearity	Asset Pricing Model	Testing Asset	Predictors
GKX	Nonlinear	Reduced Form	Stock	Firm + Macro
CPZ	Nonlinear	Pricing Kernel	Stock	Firm + Macro
IPCA	Linear	Beta Pricing	Stock	Firm
CA	Nonlinear	Beta Pricing	Stock	Firm
KNS	Linear	Pricing Kernel	Portfolio	Firm

- GKX: Gu, Kelly, and Xiu (2020)
- CPZ: Chen, Pelger, and Zhu (2019)
- IPCA: Kelly, Pruitt, and Su (2019)
- CA: Gu, Kelly, and Xiu (2019)
- KNS: Kozak, Nagel, and Santosh (2020)

Machine Learning Method I: GKX

- Neural network with 3 hidden layers (NN3)
- Example: 1 hidden layer with 5 neurons
  - $(4 + 1) \times 5 + 6 = 31$ parameters

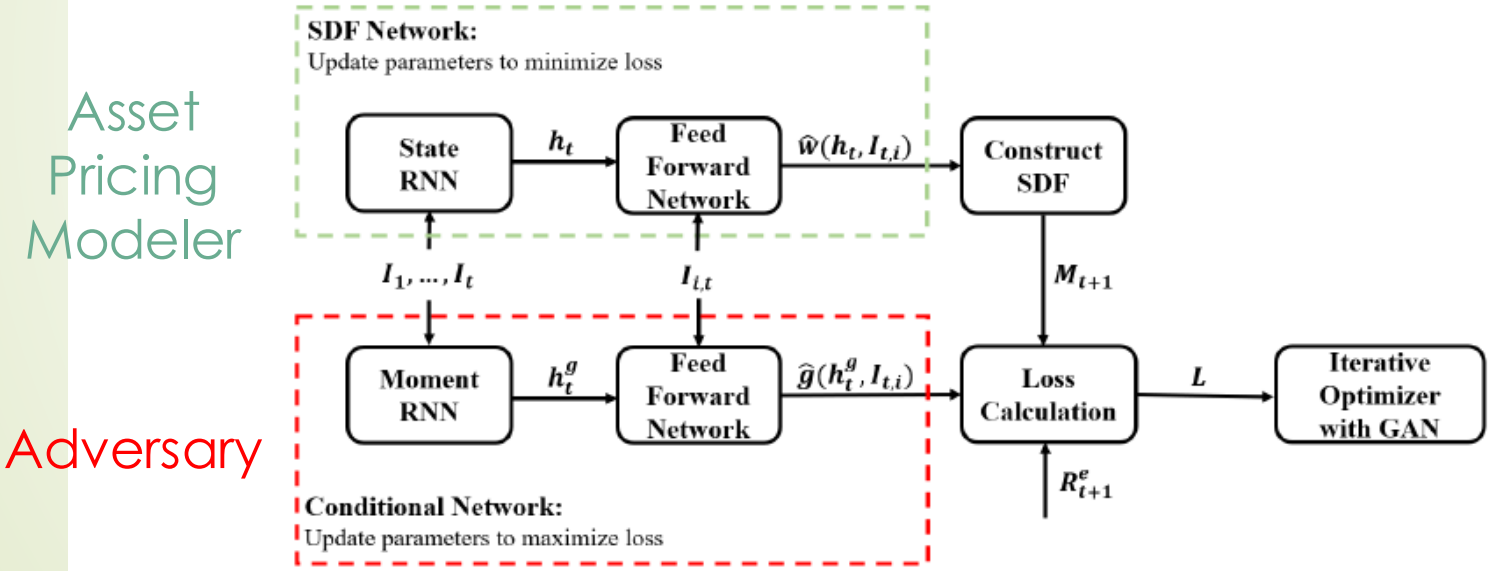


Machine Learning Method I: GKX

- Neural network with 3 hidden layers (NN3)
  - 32, 16, and 8 neurons per layer
  - Reduced form, no economic restriction
- 94 firm characteristics + 8 macroeconomic predictors + 74 industry dummies + interactions
  - (8+1) × 94 + 74 = 920 predictors
- Training sample: 18 years, 1957 to 1974
- Validation sample: 12 years, 1975 to 1986
- Out-of-sample test: 31 years, 1987 to 2017

Machine Learning Method II: CPZ

- Adversarial approach, multiple connected neural networks
  - Incorporate no-arbitrage condition to estimate SDF and stock risk loadings



Model architecture of GAN (Generative Adversarial Network) with RNN (Recurrent Neural Network) with LSTM cells.

Machine Learning Method II: CPZ

- Adversarial approach, multiple connected neural networks
- 46 firm characteristics + 178 macroeconomic predictors + interactions → 10,000+ predictors
- Training sample: 20 years, 1967 to 1986
- Validation sample: 5 years, 1987 to 1991
- Out-of-sample test: 25 years, 1992 to 2016

Machine Learning Method III: IPCA

- Instrumented principal component analysis
  - $r_{i,t+1} = \alpha_{i,t} + \beta'_{i,t} f_{t+1} + \epsilon_{i,t+1}$
  - $\beta'_{i,t} = z'_{i,t} \Gamma_\beta + \upsilon'_{\beta,i,t}$
  - Factor loadings vary with predictive characteristics linearly, 6 latent factors
  - Incorporate no-arbitrage condition
- 94 firm characteristics
- Estimated in each month
- Out-of-sample test: 31 years, 1987 to 2017

Machine Learning Method IV: CA

- Conditional autoencoder with 2 hidden layers (CA2)
  - Factor loadings vary with predictive characteristics nonlinearly through neural networks.
  - 32 and 16 neurons per layer, 5 latent factors
  - Incorporate no-arbitrage condition
- 94 firm characteristics
- Training sample: 18 years, 1957 to 1974
- Validation sample: 12 years, 1975 to 1986
- Out-of-sample test: 30 years, 1987 to 2016

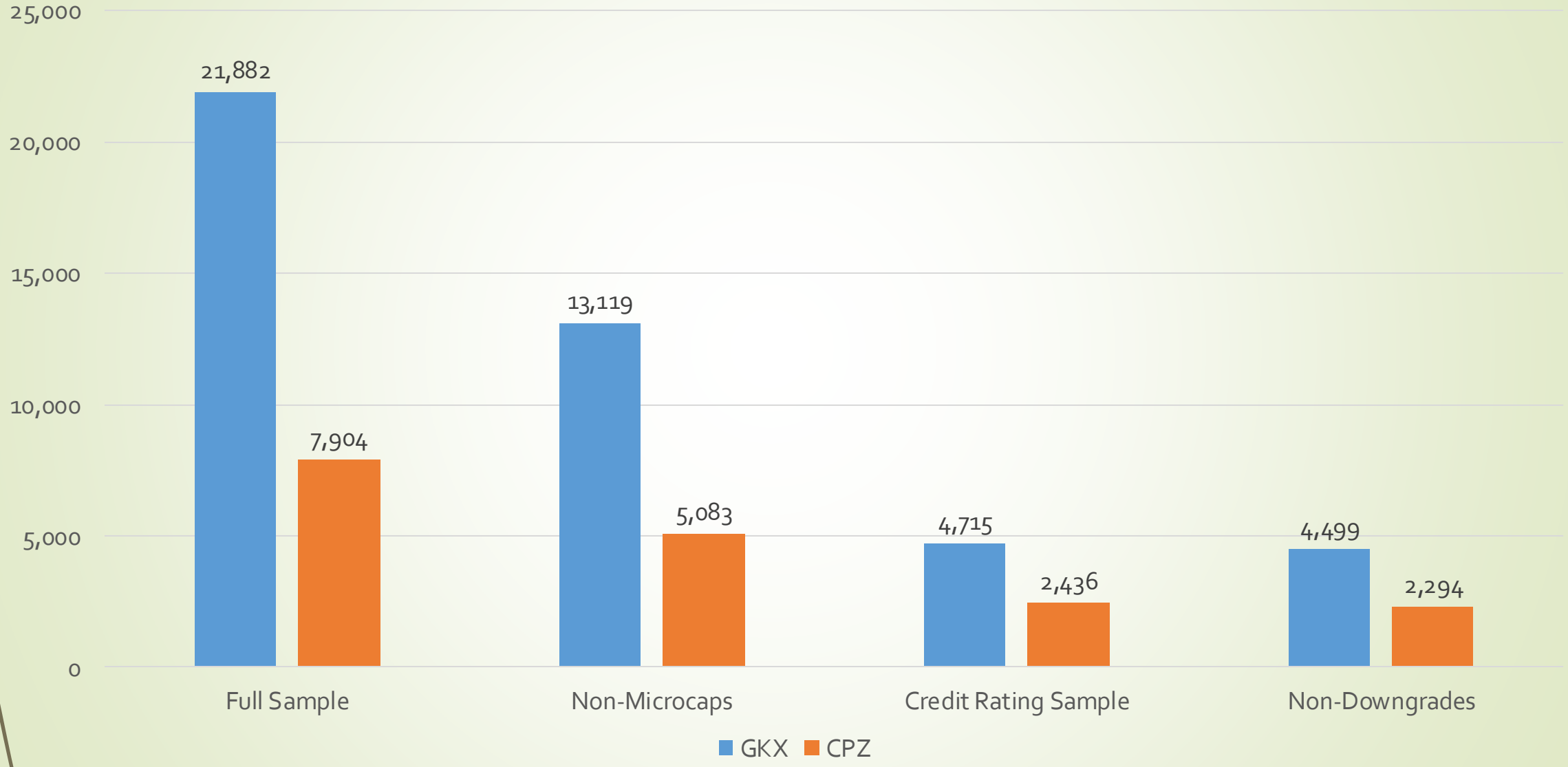
Data

- CRSP: daily and monthly stock data
- COMPUSTAT: quarterly and annual financial statement data
- GKX, IPCA, and CA: all NYSE/AMEX/Nasdaq stocks, set missing characteristics to cross-sectional median
  - 21,882 stocks, between 5,117 and 7,877 per month
- CPZ: all U.S. stocks from CRSP with available data on firm characteristics
  - 7,904 stocks, between 1,933 and 2,755 per month

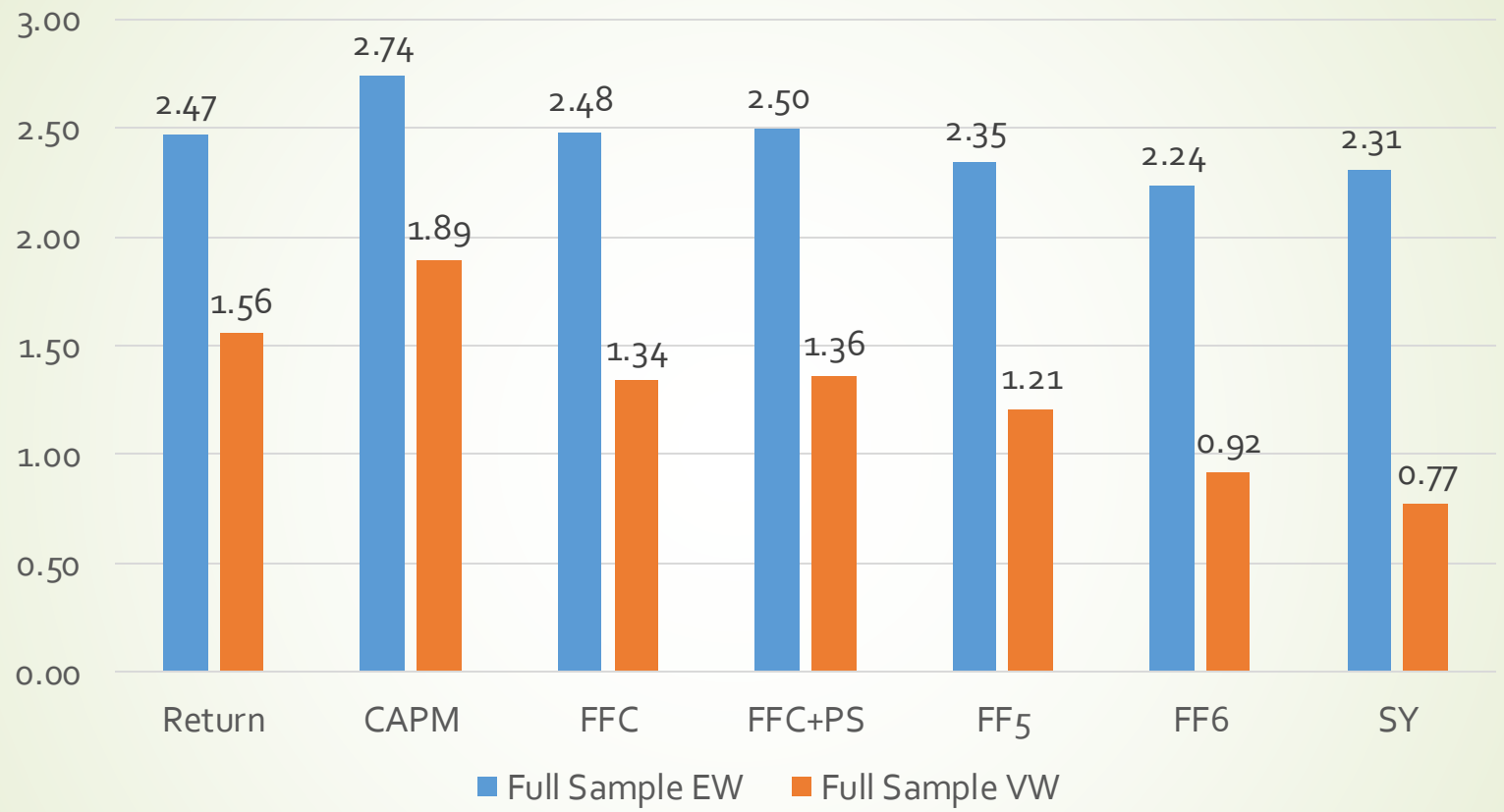
Economic Restrictions

- Cross-sectional return predictability is concentrated in microcaps and distressed firms
- Exclude microcaps: market cap smaller than the 20th NYSE size percentile
- Rated firms: firms with data on S&P long-term issuer credit rating
- Exclude distressed firms: [–12, +12] months around an issuer credit rating downgrade

Subsamples with Economic Restrictions

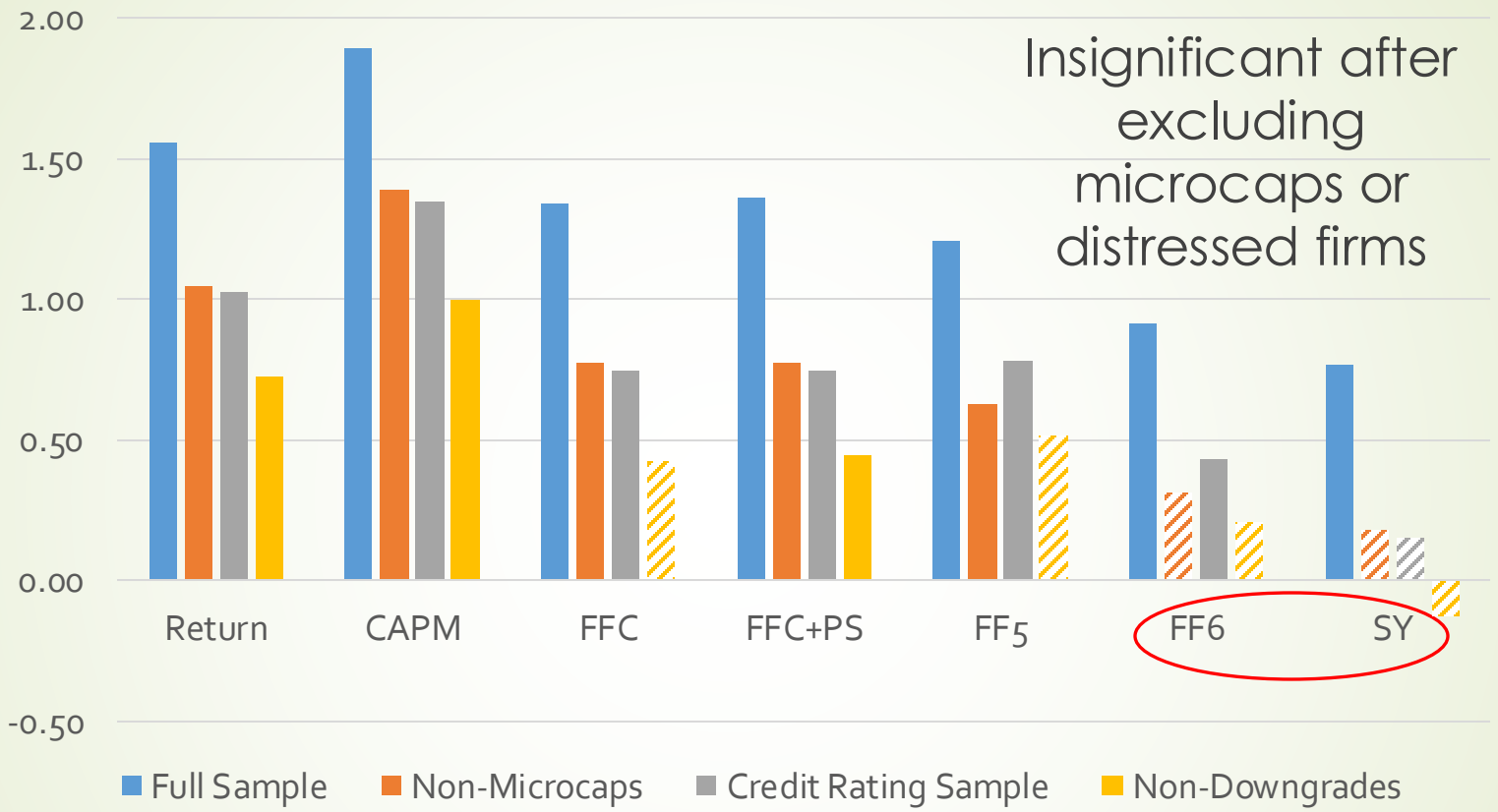


GKX Portfolio Return Spread: EW vs. VW



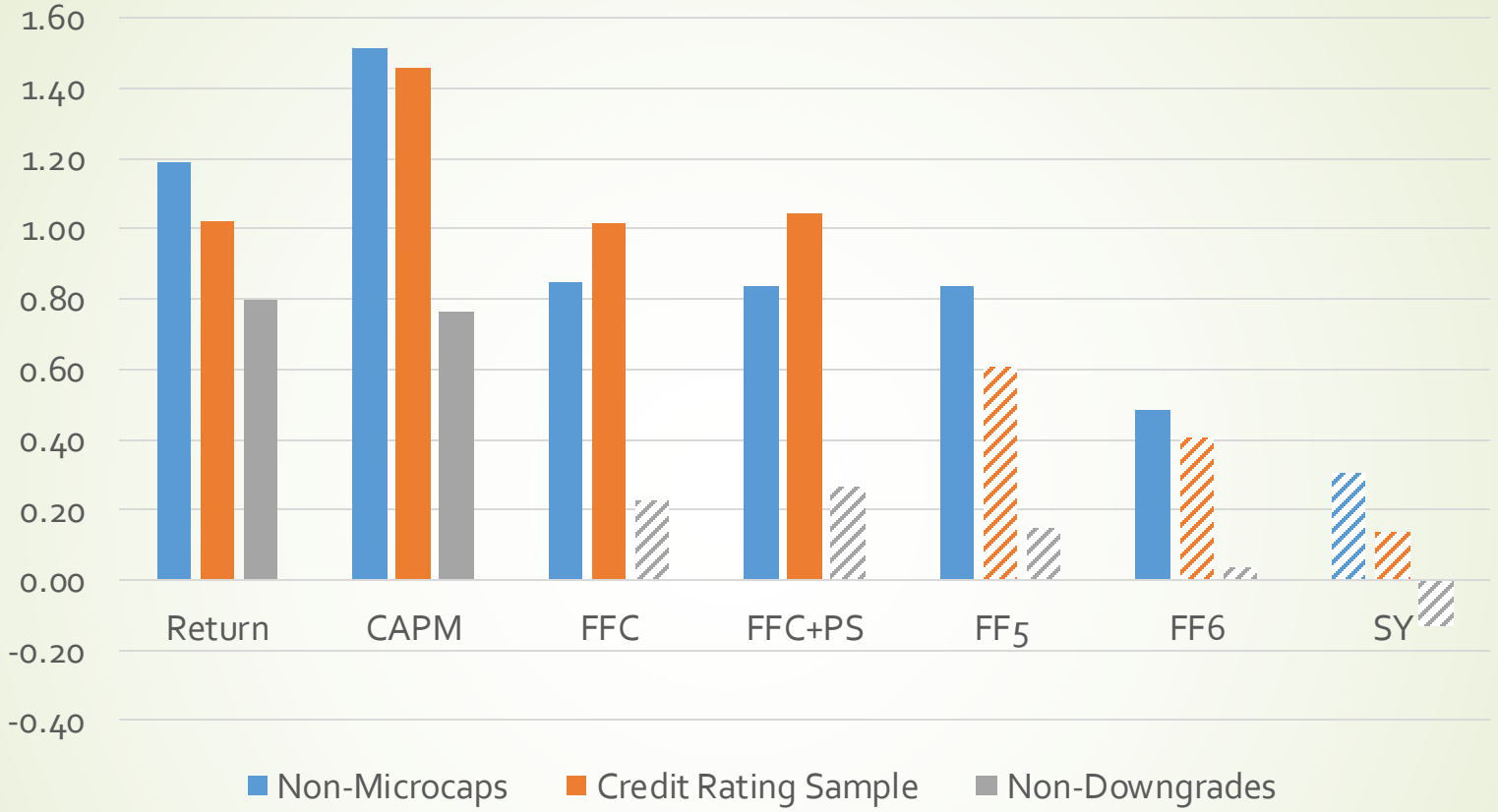
- VW performance is 48% lower than EW.

GKX Portfolio Return Spread: Economic Restrictions



- Non-microcaps: 48% lower than the full sample; Rated firms: 46% ↓; Non-downgrades: 70% ↓

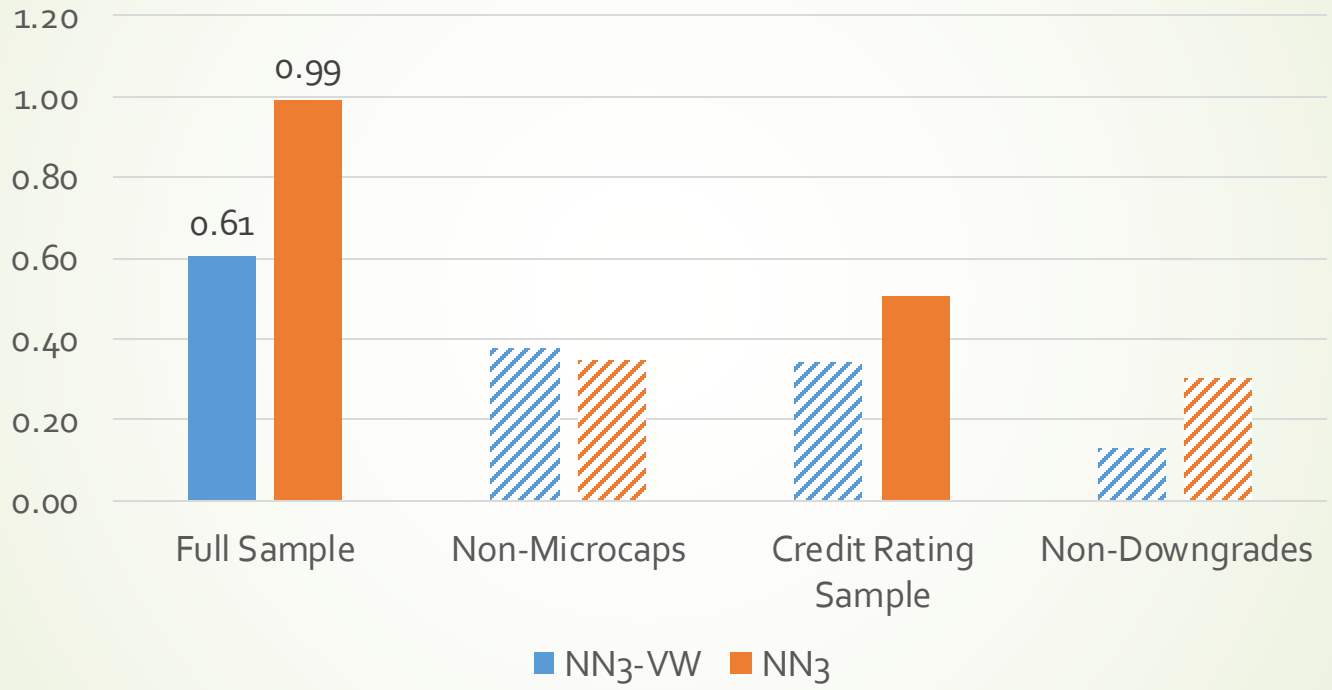
Robustness Test: Train the NN3 Model in Subsamples



- Non-microcaps: 37% lower than the full sample; Rated firms: 50% ↓; Non-downgrades: 84% ↓

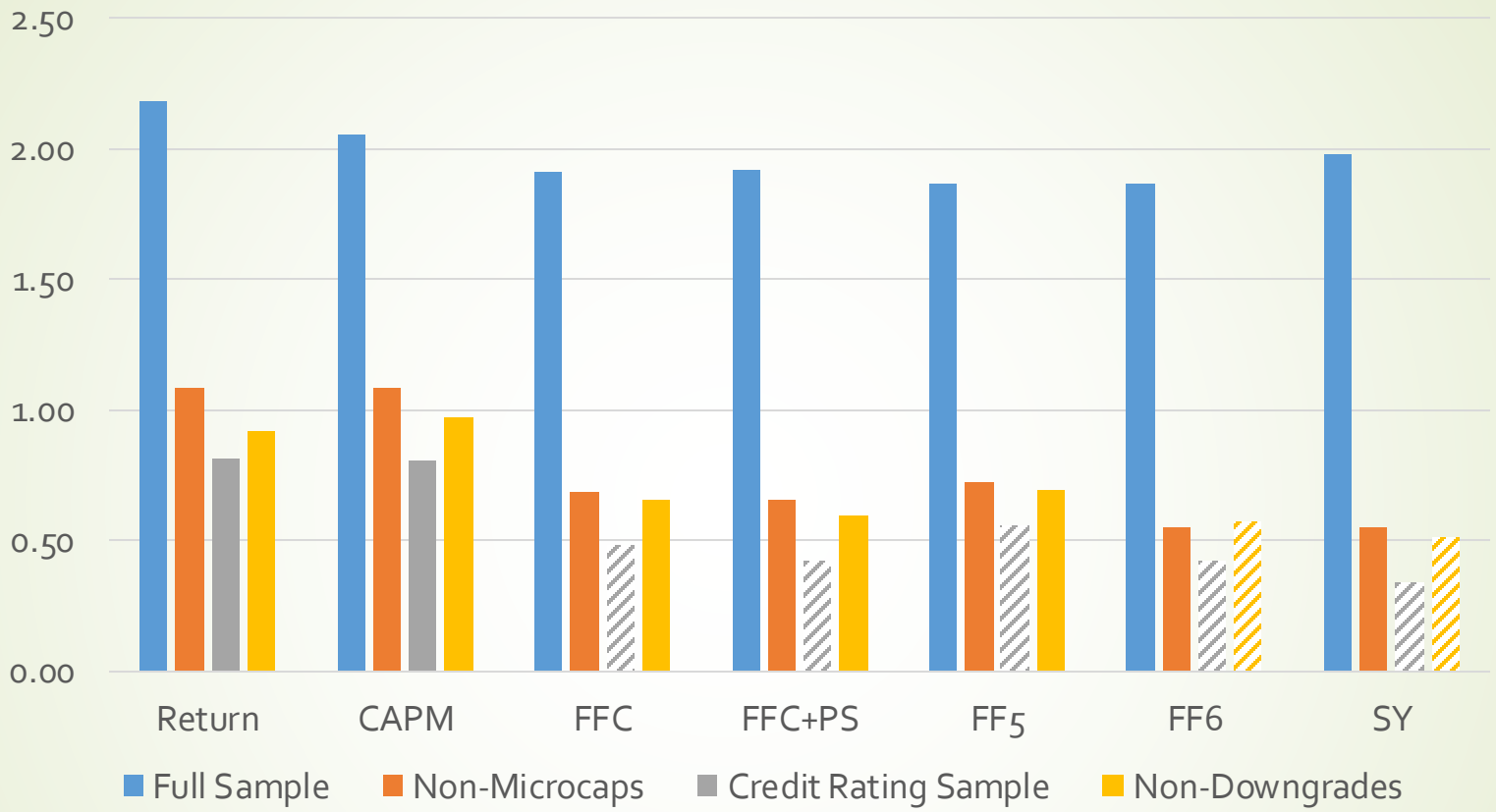
Robustness Test: Alternative Objective Function

- NN3: EW loss function, predict return
- NN3-VW: VW loss function, predict FF6 alpha



- The seemingly more aligned objective function does not necessarily improve the predictive performance.

CPZ Portfolio Return Spread: Economic Restrictions

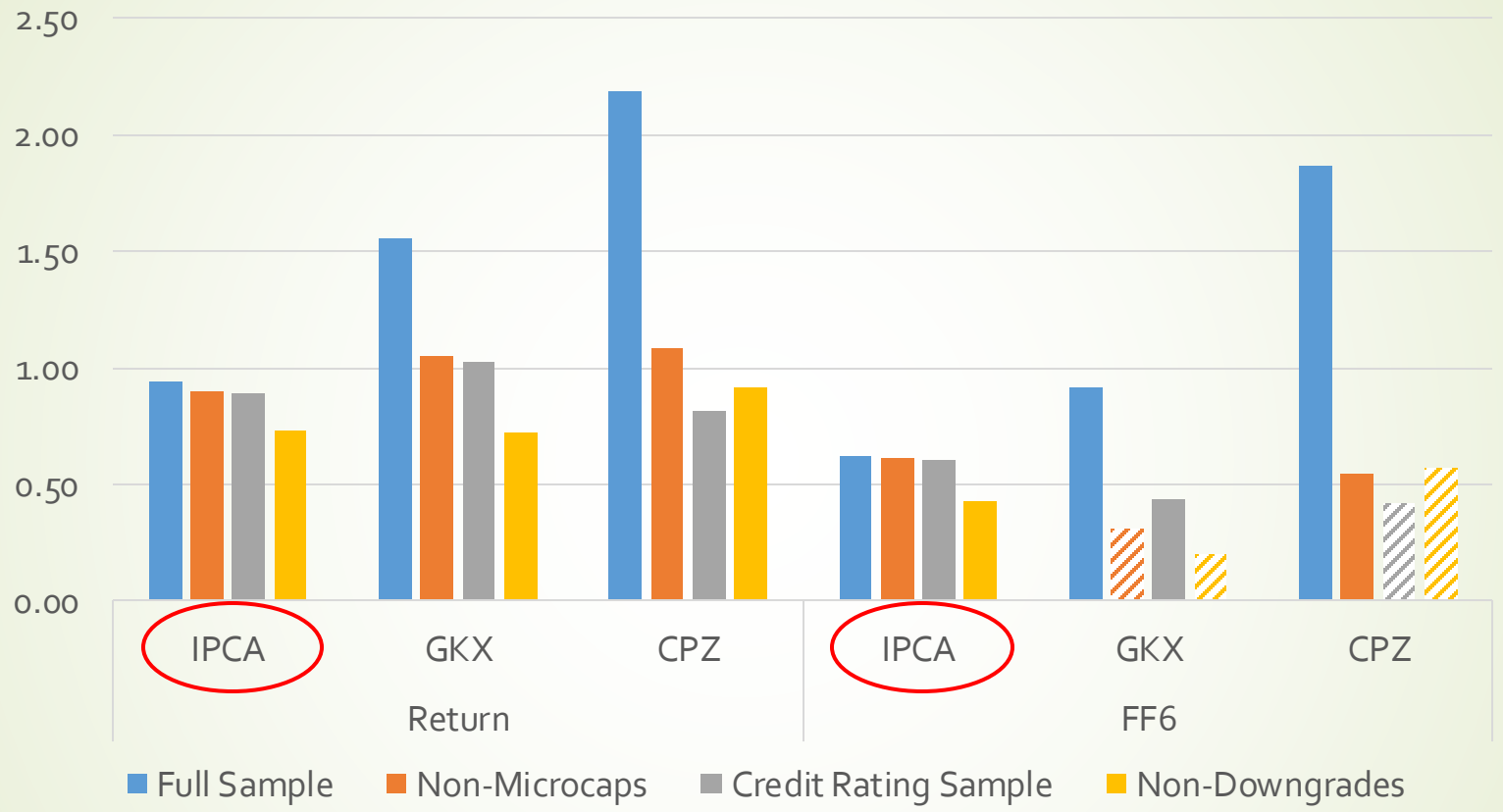


- Non-microcaps: 62% lower than the full sample; Rated firms: 72% ↓; Non-downgrades: 65% ↓

ML Portfolio Return Spread: IPCA vs. GKX vs. CPZ

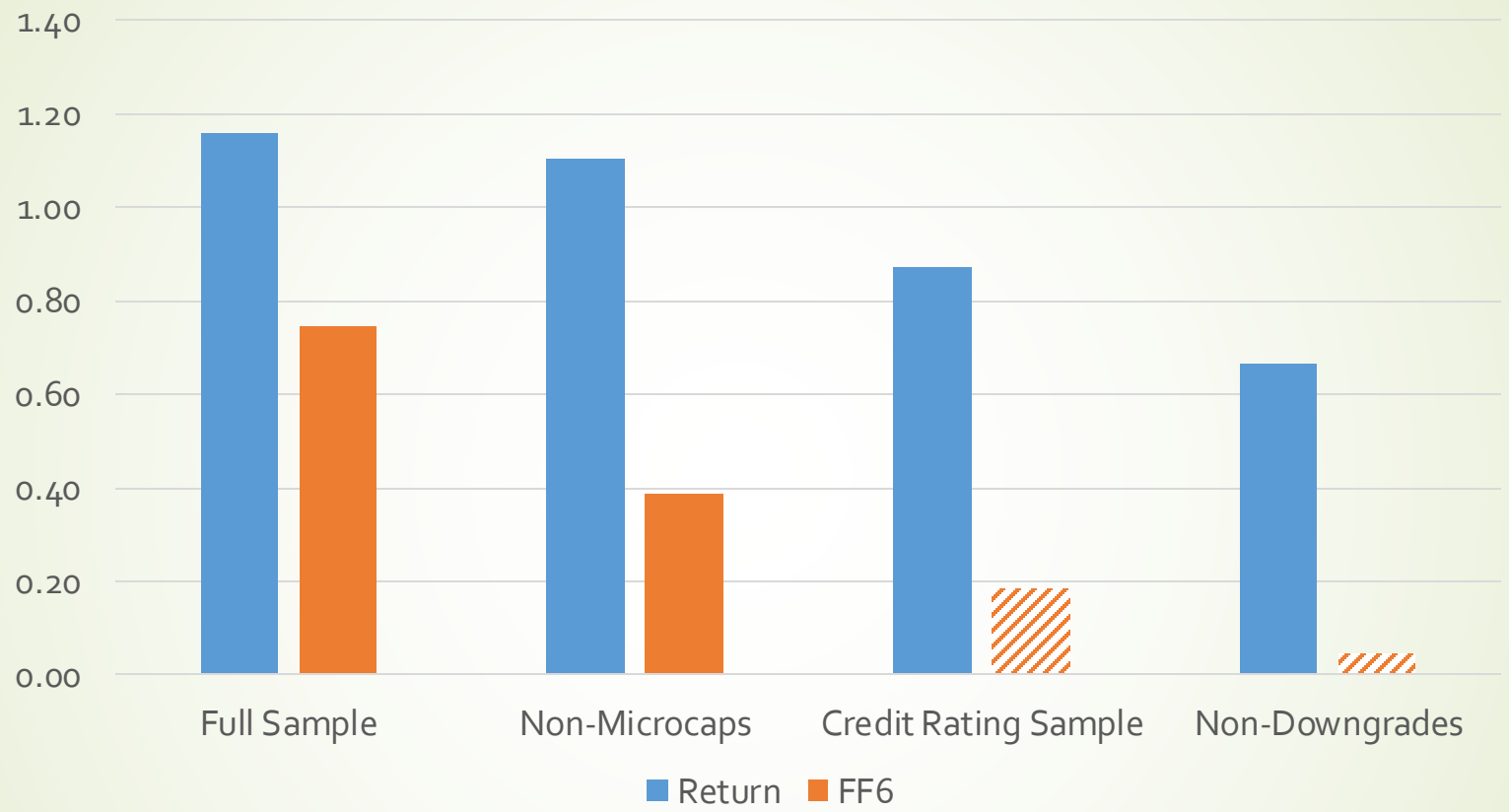
- IPCA underperforms deep learning models in the full sample.

ML Portfolio Return Spread: IPCA vs. GKX vs. CPZ



- IPCA: no material deterioration of performance among the cheap-to-trade stocks

CA Portfolio Return Spread: Economic Restrictions



- Non-microcaps: 48% lower than the full sample; Rated firms: 75% ↓; Non-downgrades: 94% ↓

Characteristics of ML Portfolios

- ML methods: positive/less negative skewness; smaller maximum drawdown than the market; higher return during the crisis period

	Sharpe Ratio	Skewness	Excess Kurtosis	Maximum Drawdown	Return in Crisis	Turnover
Panel A: Sorted by NN3-Predicted Return						
Full Sample	0.944	0.631	5.222	0.350	4.100	0.976
Non-Microcaps	0.644	0.361	7.062	0.349	3.563	0.869
Panel B: Sorted by Risk Loading						
Full Sample	1.225	1.063	5.932	0.209	0.472	1.664
Non-Microcaps	0.839	0.326	1.582	0.246	0.677	1.625
Panel C: Sorted by IPCA-Predicted Return						
Full Sample	0.967	-0.449	4.805	0.203	0.574	1.186
Non-Microcaps	0.978	-0.267	5.369	0.234	1.493	1.130
Panel D: Sorted by CA2-Predicted Return						
Full Sample	0.784	-0.077	2.418	0.202	-0.047	1.565
Non-Microcaps	0.748	0.291	4.684	0.207	-0.529	1.478
Panel E: Market Portfolio						
Full Sample	0.527	-0.978	3.323	0.486	-6.954	0.089
Non-Microcaps	0.530	-0.959	3.222	0.485	-6.907	0.086

Characteristics of ML Portfolios

- ML methods: require high turnover in portfolio rebalancing
- One-side turnover: < 10% (size, value); 14% to 35% (failure probability, IVOL); > 90% (short-term reversals, seasonality)

	Sharpe Ratio	Skewness	Excess Kurtosis	Maximum Drawdown	Return in Crisis	Turnover
Panel A: Sorted by NN3-Predicted Return						
Full Sample	0.944	0.631	5.222	0.350	4.100	0.976
Non-Microcaps	0.644	0.361	7.062	0.349	3.563	0.869
Panel B: Sorted by Risk Loading						
Full Sample	1.225	1.063	5.932	0.209	0.472	1.664
Non-Microcaps	0.839	0.326	1.582	0.246	0.677	1.625
Panel C: Sorted by IPCA-Predicted Return						
Full Sample	0.967	-0.449	4.805	0.203	0.574	1.186
Non-Microcaps	0.978	-0.267	5.369	0.234	1.493	1.130
Panel D: Sorted by CA2-Predicted Return						
Full Sample	0.784	-0.077	2.418	0.202	-0.047	1.565
Non-Microcaps	0.748	0.291	4.684	0.207	-0.529	1.478
Panel E: Market Portfolio						
Full Sample	0.527	-0.978	3.323	0.486	-6.954	0.089
Non-Microcaps	0.530	-0.959	3.222	0.485	-6.907	0.086

Break-Even Transaction Cost

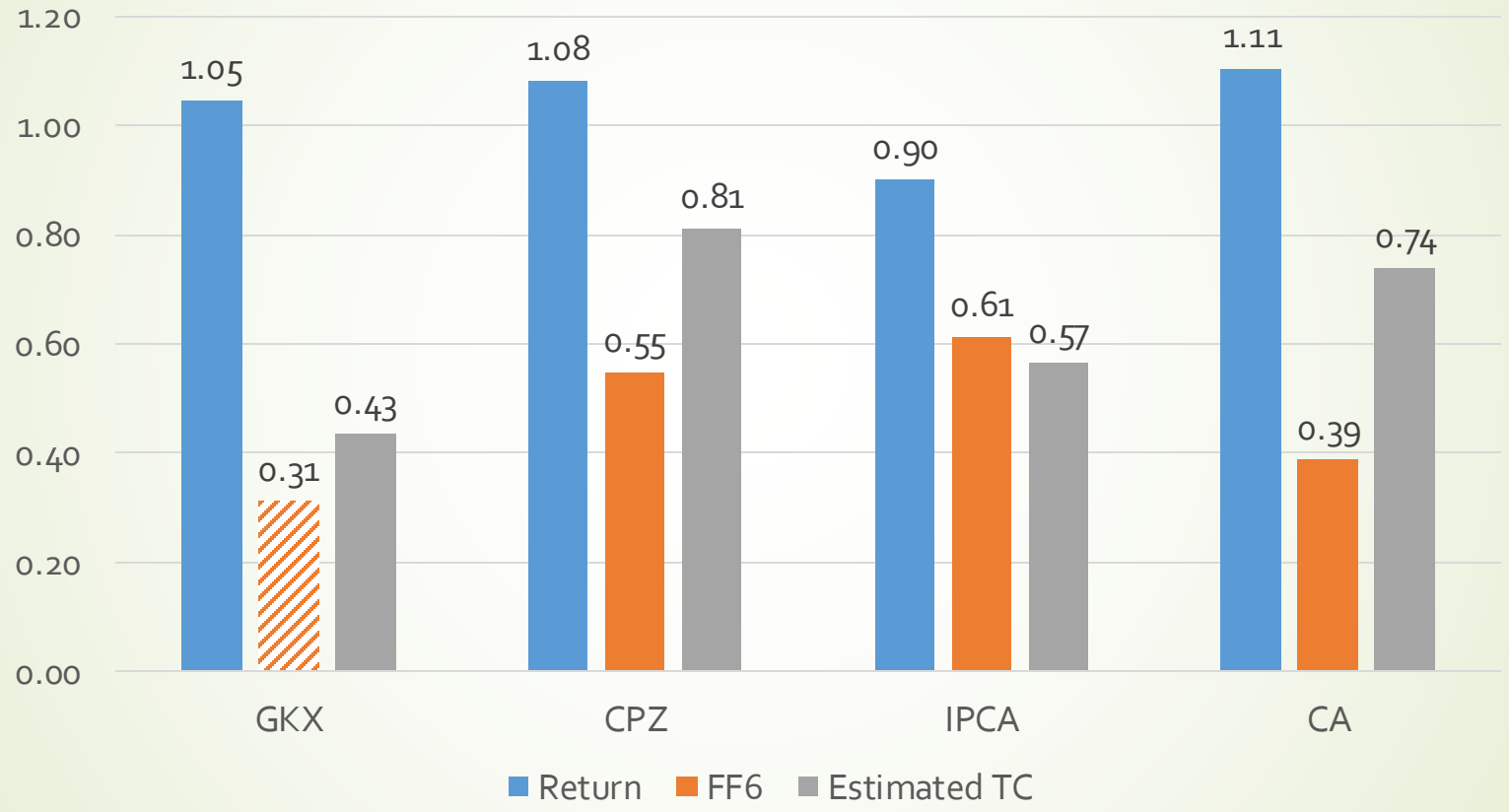
- Novy-Marx and Velikov (2016): > 0.5%
- Brandt, Santa-Clara, and Valkanov (2009): $c_{i,t} = z_{i,t} \times T_t$, where $z_{i,t} = 0.006 - 0.0025 \times ME_{i,t}$

→ 0.67% for the full sample and 0.64% for non-microcaps

	FF6	Turnover	Break-Even Cost
Panel A: Sorted by NN3-Predicted Return			
Full Sample	0.916	0.976	0.94
Non-Microcaps	0.312	0.869	0.36
Panel B: Sorted by Risk Loading			
Full Sample	1.867	1.664	1.12
Non-Microcaps	0.548	1.625	0.34
Panel C: Sorted by IPCA-Predicted Return			
Full Sample	0.624	1.186	0.53
Non-Microcaps	0.613	1.130	0.54
Panel D: Sorted by CA2-Predicted Return			
Full Sample	0.746	1.565	0.48
Non-Microcaps	0.387	1.478	0.26

ML Portfolio Return Spreads: Non-Microcaps + VW

- Assume transaction cost = 0.5% of the long-short portfolio turnover



An Alternative ML Method

- CPZ: estimate SDF for individual stocks
- Kozak, Nagel, and Santosh (2020): estimate SDF for equity portfolios, i.e., long-short portfolio return based on predictive characteristics
- Minimize the Hansen-Jagannathan (1991) distance
- Ridge regression with three-fold cross-validation
- Apply the 94 characteristics in GKX
- In-sample estimation: 1964 to 2004
- Out-of-sample test: 2005 to 2017

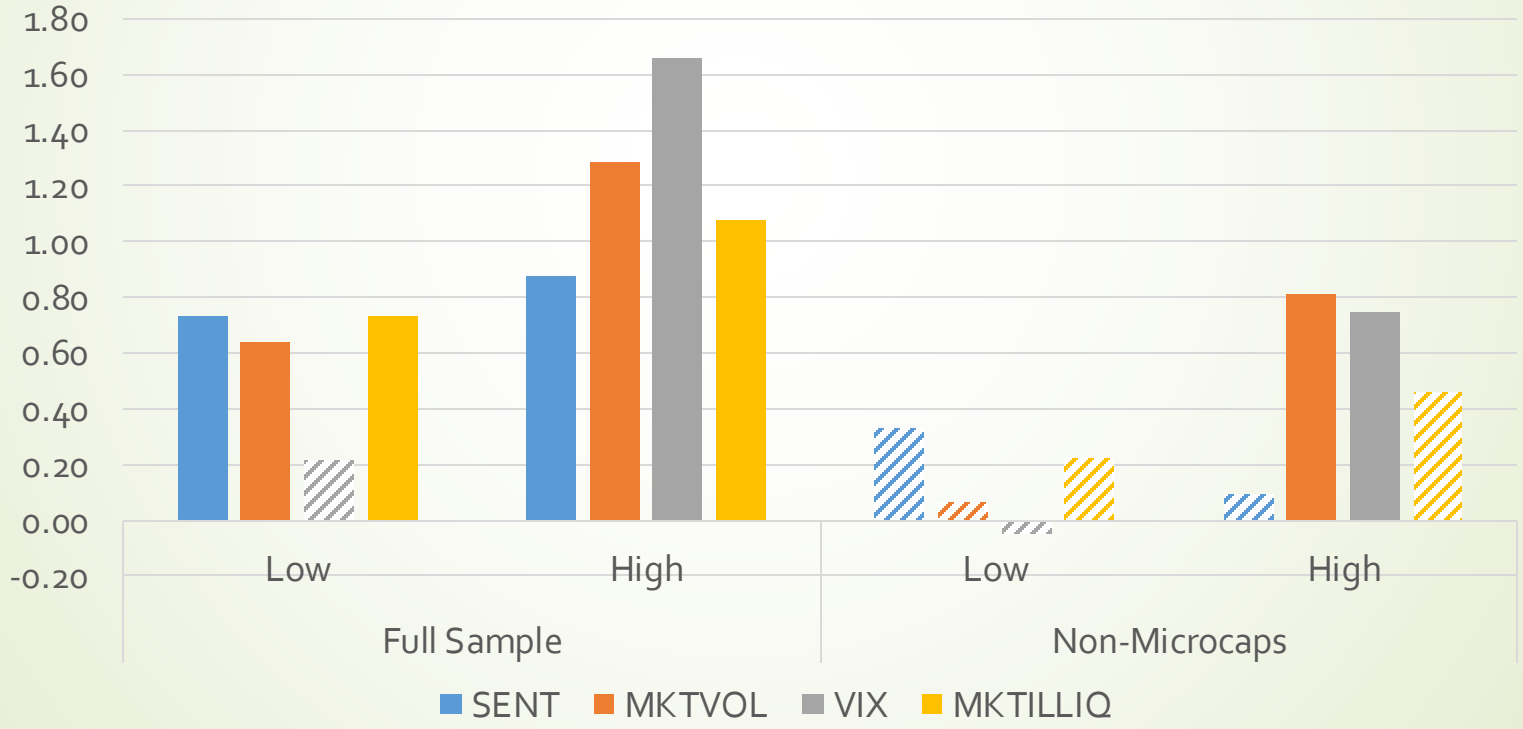
Characteristics of SDF-Implied MVE Portfolios

	CAPM	FF6	Sharpe Ratio	SDF-Implied MVE Portfolio Weights					
				Mean	10%	25%	Median	75%	90%
Full Sample	3.662***	3.338***	2.318	0.083	-1.994	-0.912	0.341	0.964	1.687
	(6.01)	(5.90)							
Non-Microcaps	1.543***	0.895***	0.977	0.084	-0.592	-0.238	0.072	0.407	0.647
	(3.88)	(2.87)							
Credit Rating Sample	1.418***	0.717*	0.898	-0.006	-0.382	-0.137	-0.003	0.187	0.326
	(2.97)	(1.93)							
Non-Downgrades	1.308***	0.545	0.828	-0.022	-0.370	-0.217	0.004	0.135	0.293
	(2.92)	(1.59)							

- Imposing economic restrictions reduces performance, and the odds of extreme positions
- Deep learning techniques face the usual challenge of cross-sectional return predictability: concentrated in difficult-to-arbitrage stocks + sizable trading costs (high turnover and extreme positions)

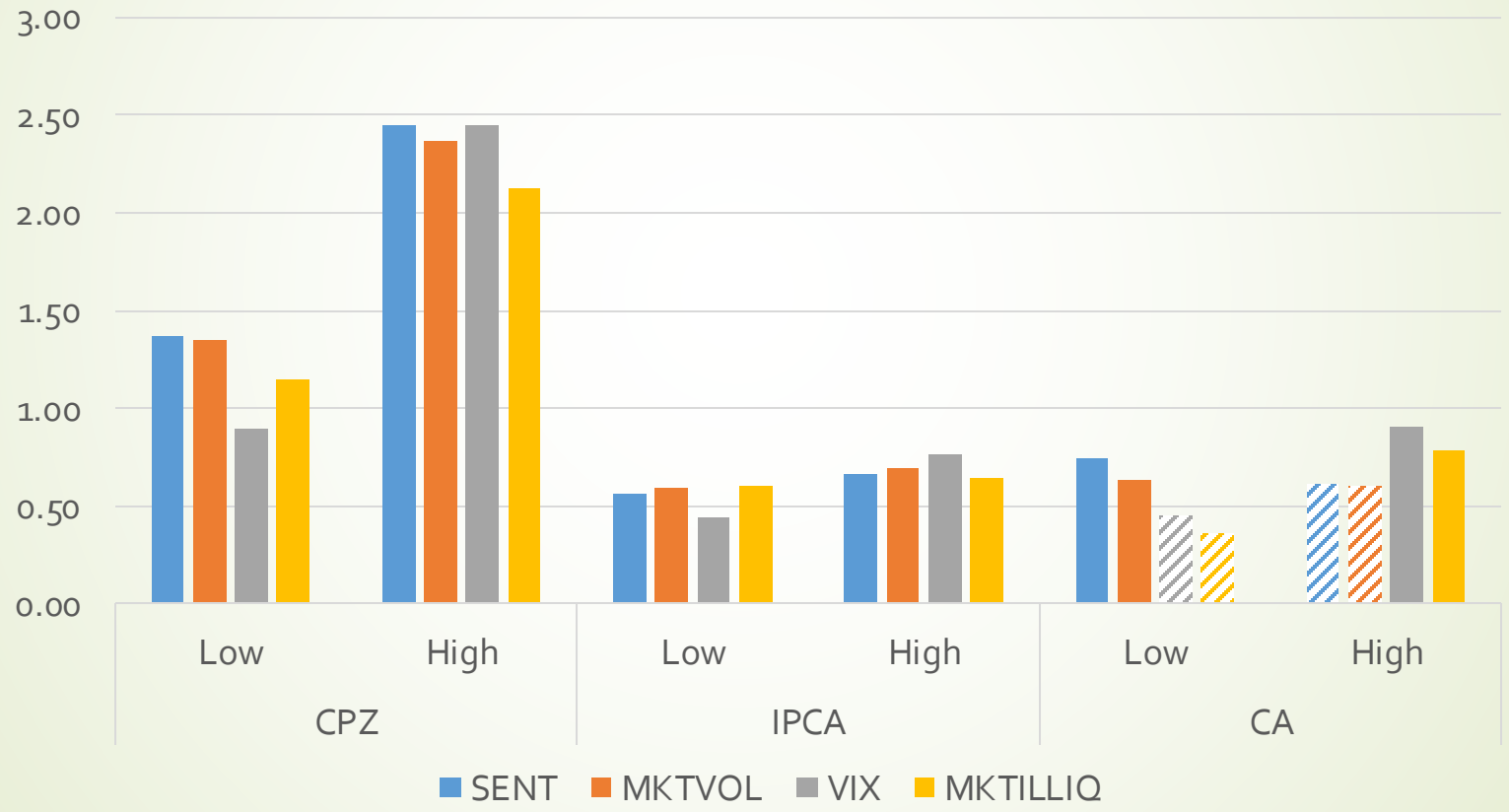
Time-Varying Return Predictability: GKX-FF6

- Binding limits to arbitrage → more profitable anomaly-based trading strategies
  - High sentiment, high volatility, and low liquidity



Time-Varying Return Predictability: Full Sample-FF6

- CPZ: outperforms in high limits-to-arbitrage periods
- IPCA and CA: low time series variation, mixed evidence



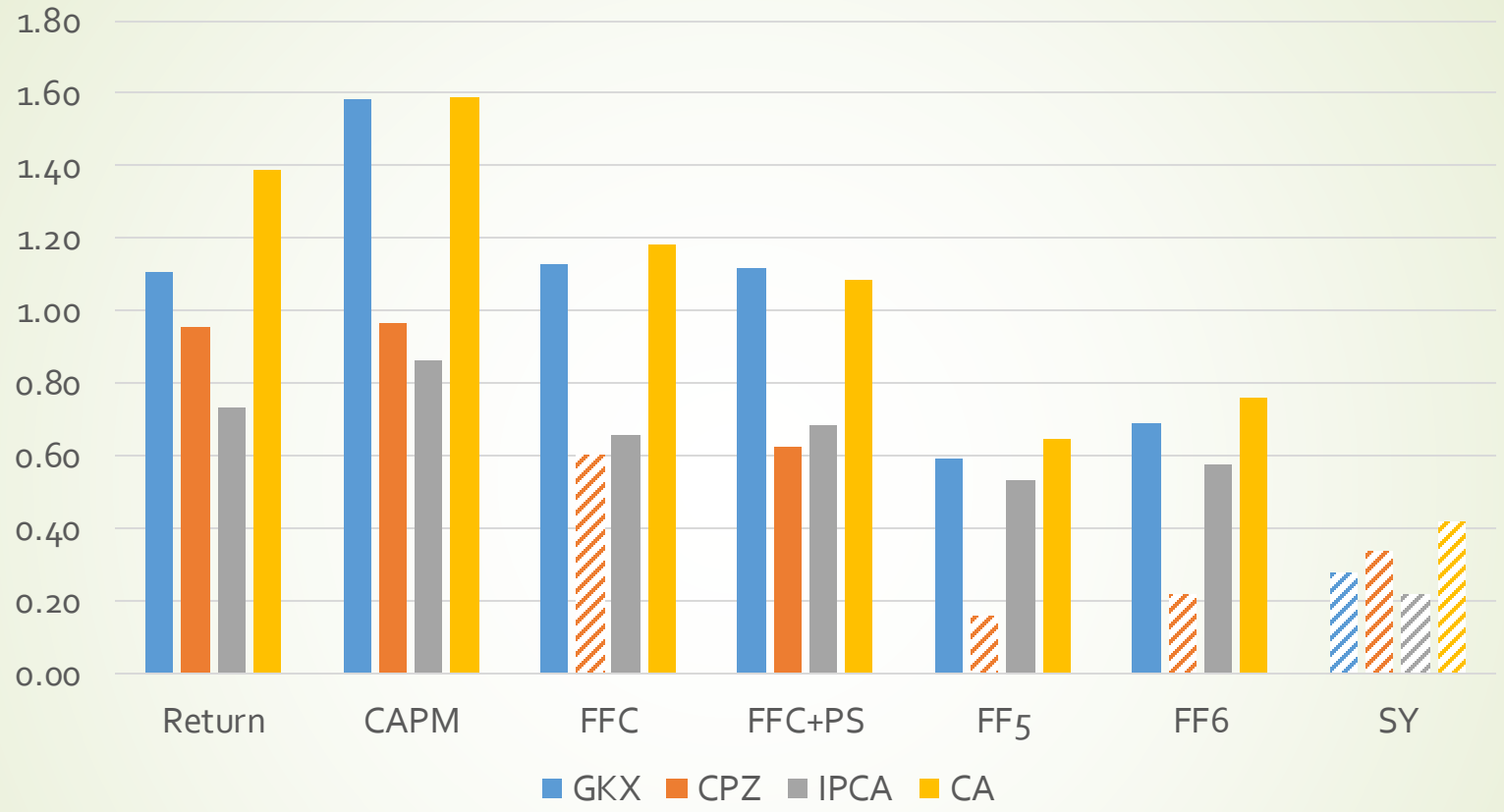
Time-Varying Return Predictability: GKX and CPZ

- $HML_t = \alpha_0 + \beta_1 High\ SENT_{t-1} + \beta_2 High\ MKTVOL_{t-1} + \beta_3 High\ MKTILLIQ_{t-1} + \beta_4 M_{t-1} + c'F_t + e_t$

	Sorted by NN3-Predicted Return				Sorted by Risk Loading			
	Model 1	Model 2	Model 3	Model 4	Model 5	Model 6	Model 7	Model 8
Constant	0.016	-0.453	0.865	1.252	0.103	-0.081	0.432	0.528
	(0.03)	(-0.92)	(1.14)	(1.42)	(0.16)	(-0.13)	(0.32)	(0.36)
High SENT	1.534**	1.710**	0.228	-0.005	1.412**	1.395**	1.161*	1.124*
	(2.43)	(2.51)	(0.58)	(-0.01)	(2.32)	(2.23)	(1.91)	(1.75)
High MKTVOL	0.791		0.959*		0.787		1.065	
	(1.24)		(1.93)		(1.29)		(1.56)	
High VIX		1.851***		1.647***		1.255**		1.563**
		(2.85)		(3.28)		(2.09)		(2.28)
High MKTILLIQ	0.754	0.529	0.592	0.695	1.828***	1.918***	1.609**	1.851**
	(1.24)	(0.78)	(1.37)	(1.41)	(2.86)	(2.89)	(2.20)	(2.31)
Controls	N	N	Y	Y	N	N	Y	Y

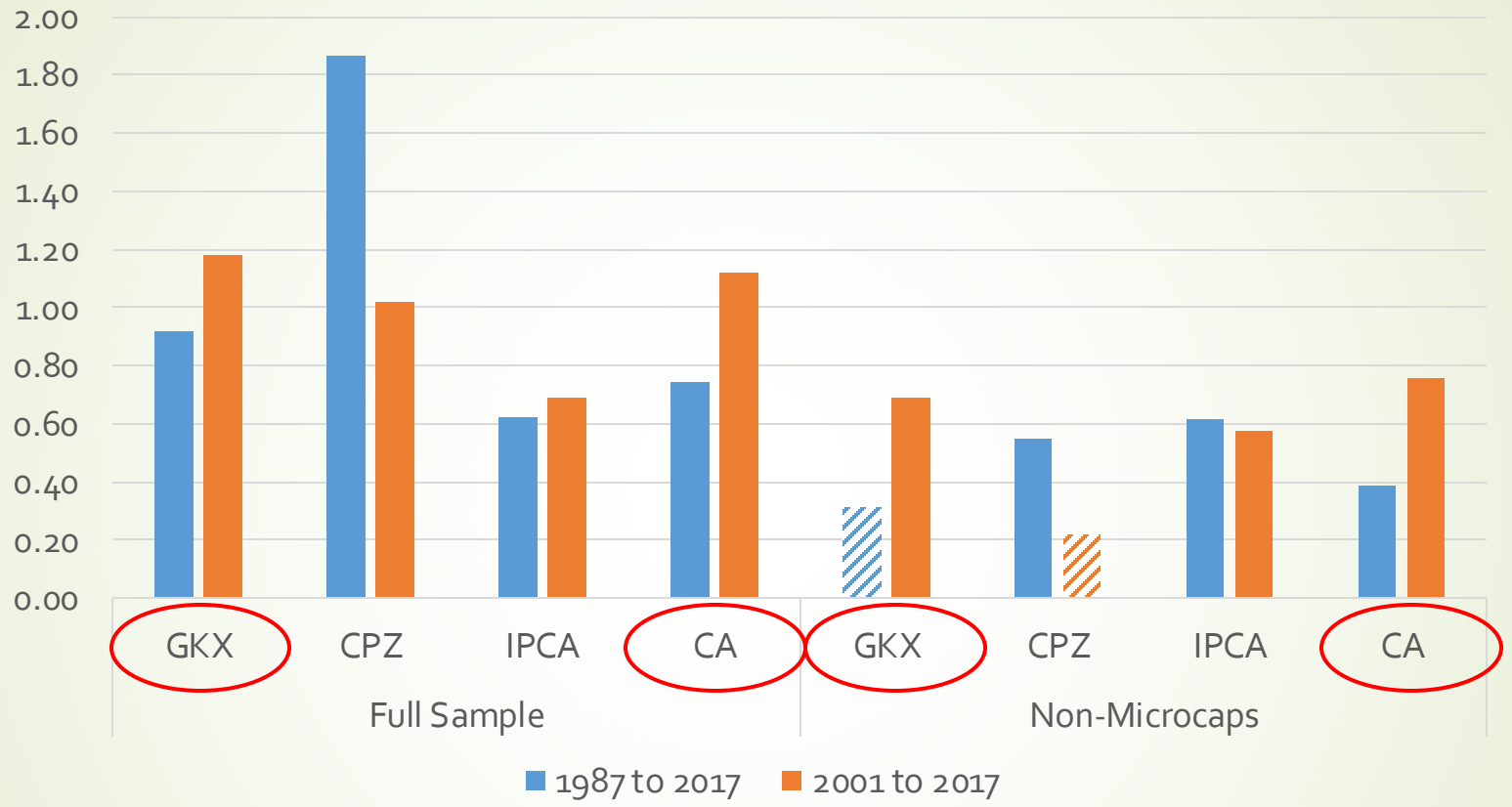
- Controls: down market state, term spread, default spread, Fama-French six factors

Return Predictability in Recent Years: Non-Microcaps



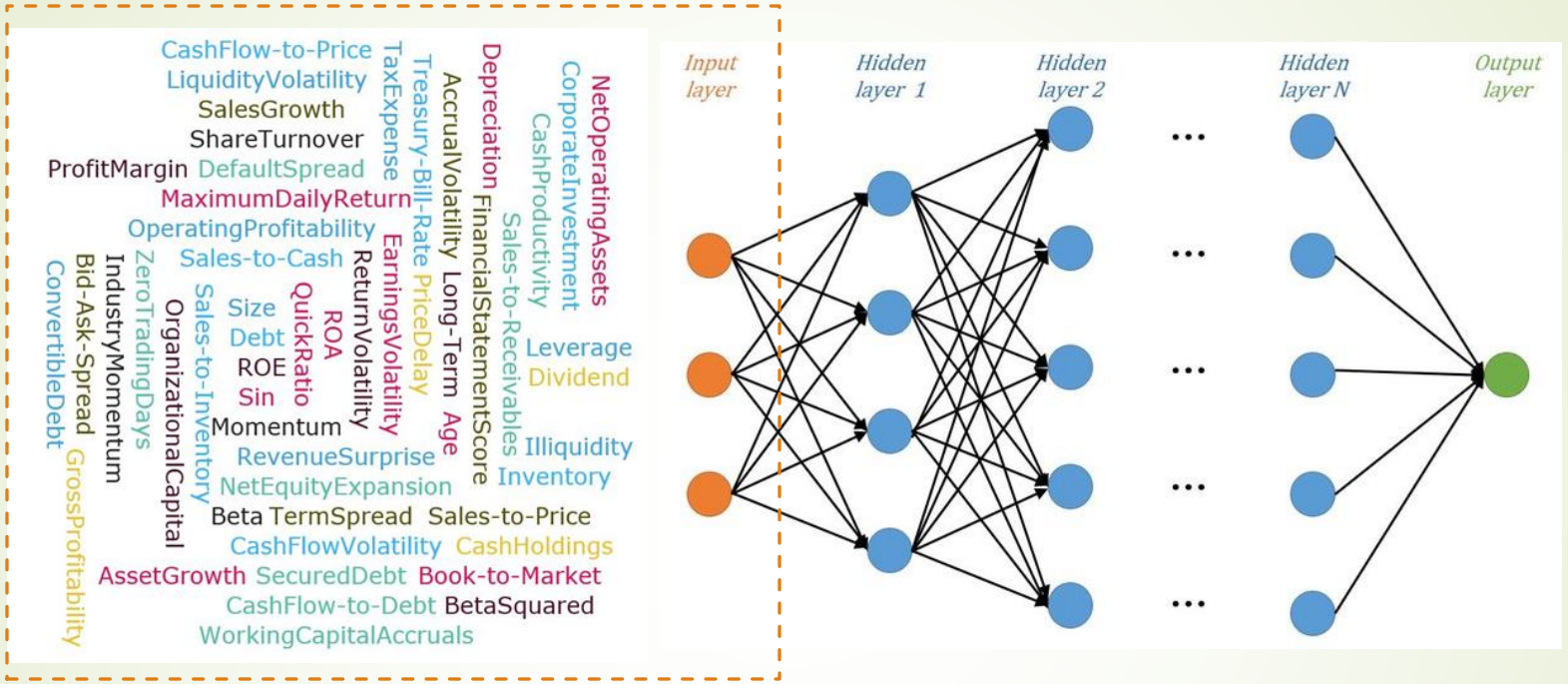
- Unlike individual anomalies, most ML signals continue to predict the stock returns after 2001.

Return Predictability in Recent Years: FF6



- Unlike individual anomalies, there is no vast drop in trading profits of ML signals.

Deep Learning Models as 'Black Boxes'



- Common features of stocks selected by ML methods
- Decompose to intra-industry vs. inter-industry strategy

Stock Characteristics of ML Portfolios

- All ML methods identify stocks in line with most anomaly-based trading strategies.
- Long positions: small, value, illiquid and old stocks with low price, low beta, high 11-month return, low asset growth, low equity issuance, low credit rating coverage, and low analyst coverage.
- Exceptions: long stocks with high corporate investment and high idiosyncratic volatility
  - Nonlinear relation, interact with other firm characteristics or macro conditions
- Advantage: does not require prior knowledge on truly useful characteristics and models, avoid the data snooping problem

Intra-Industry vs. Inter-Industry Return Predictability

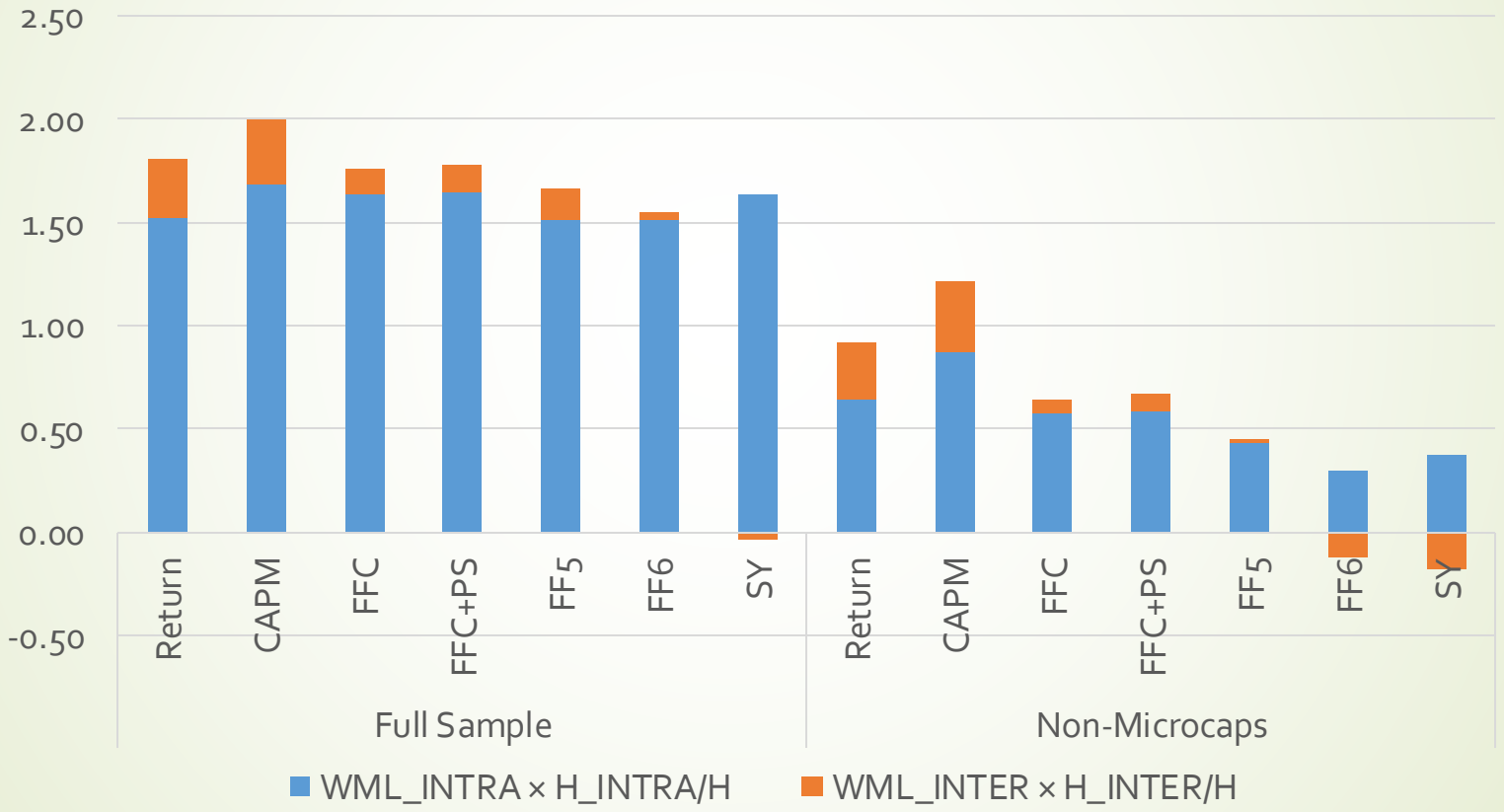
- ML signals identify mispricing in difficult-to-arbitrage stocks.
- Industry-adjustment controls for firm fundamentals and might better predict the subsequent corrections.
- Unconditional strategy: buy market winners and sell market losers
- $WML_{t+1} = \frac{1}{H_t} \sum_{i=1}^{N_t} \left(\hat{R}_{i,t} - \hat{R}_{m,t}\right) R_{i,t+1}$
- $H_t = \frac{1}{2} \sum_{i=1}^{N_t} |\hat{R}_{i,t} - \hat{R}_{m,t}|$
- $\hat{R}_{i,t} - \hat{R}_{m,t} = \hat{R}_{i,t} - \hat{R}_{j,t} + \hat{R}_{j,t} - \hat{R}_{m,t}$

Intra-Industry vs. Inter-Industry Return Predictability

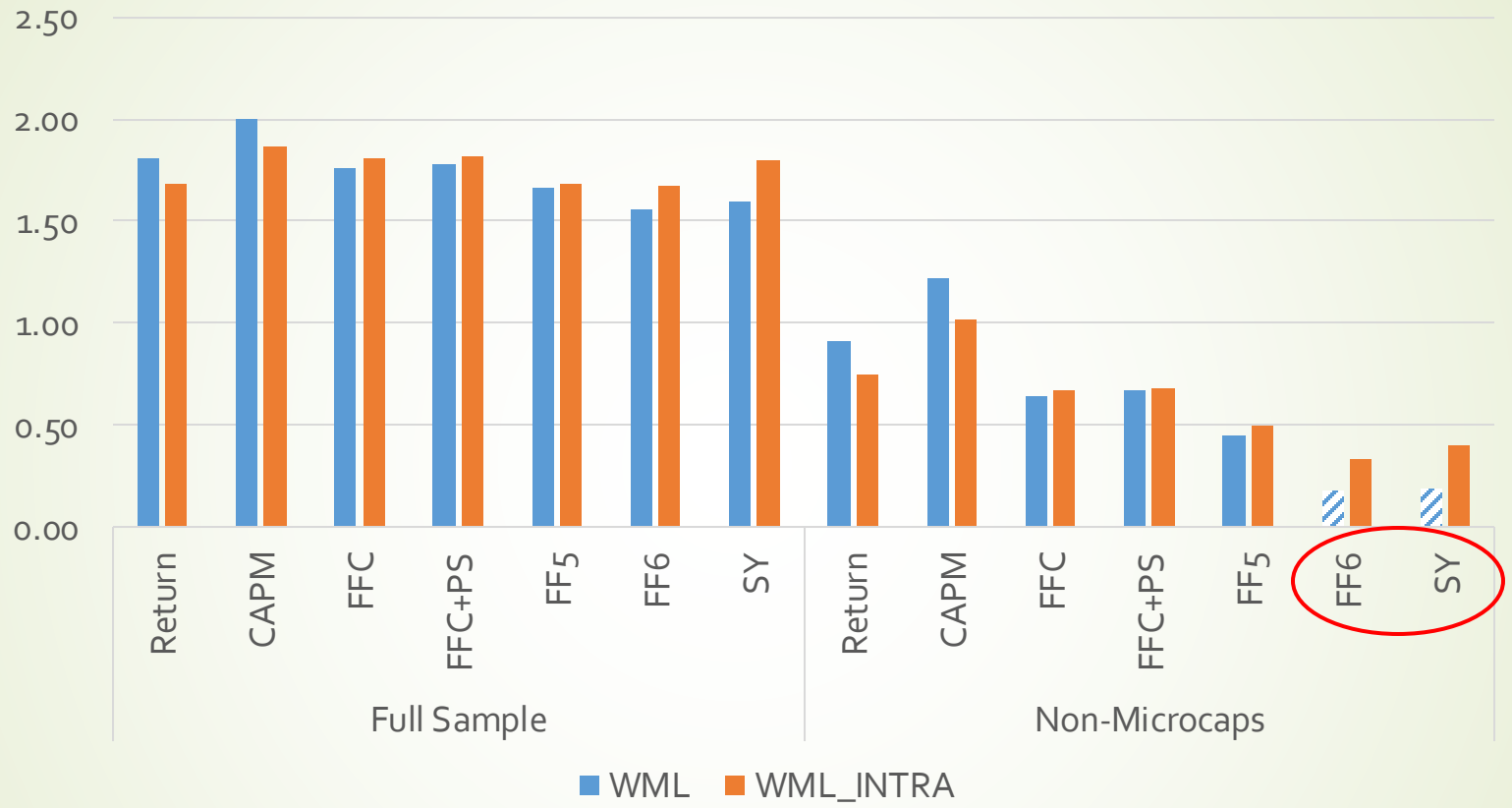
- $WML_{t+1} = \frac{1}{H_t} \sum_{i=1}^{N_t} (\hat{R}_{i,t} - \hat{R}_{j,t}) R_{i,t+1} + \frac{1}{H_t} \sum_{j=1}^{L_t} (\hat{R}_{j,t} - \hat{R}_{m,t}) N_{j,t} R_{j,t+1}$
- Unconditional = Intra-Industry + Inter-Industry
- Intra-Industry strategy: buy industry winners and sell industry losers
- Inter-Industry strategy: buy winner industries and sell loser industries

Intra-Industry vs. Inter-Industry Strategy: GKX

- Intra-industry strategy accounts for 84% (93%) of the unconditional payoff in raw (risk-adjusted) return across all stocks → stock selection



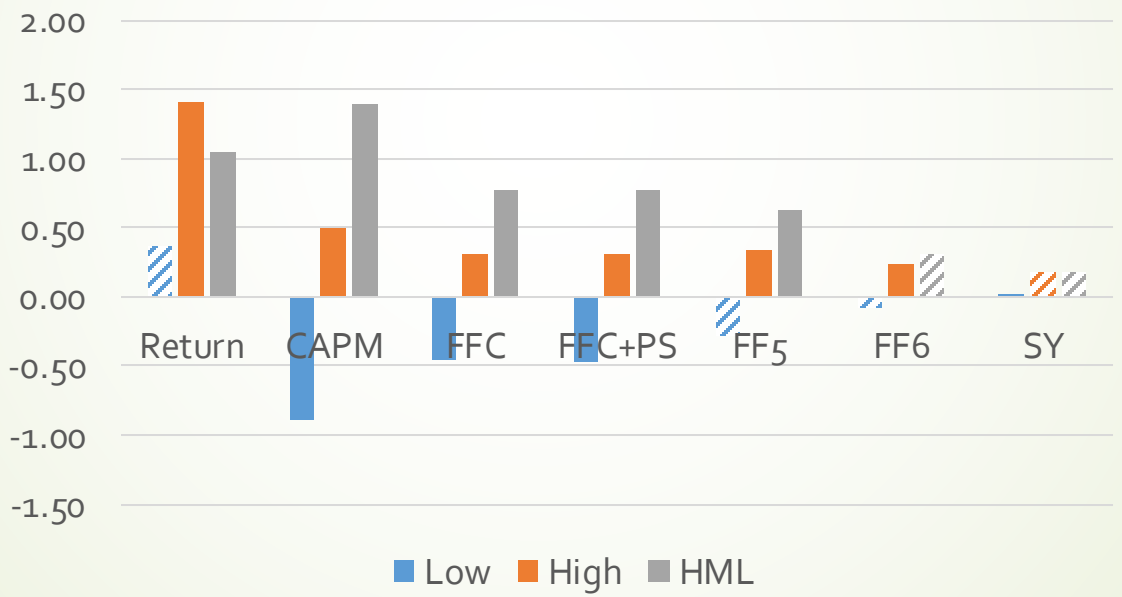
Intra-Industry vs. Unconditional Strategy: GKX



- Intra-industry strategy improves performance, especially on non-microcaps.

ML in Asset Management

- Mitigate the downside risk and hedge against crisis
- Remain profitable in recent years
- Profitable in long positions: e.g., GKX signal, non-microcaps + VW



Conclusion

- Investments based on deep learning signals extract profitability primarily from difficult-to-arbitrage stocks and during high limits-to-arbitrage market states.
- Performance further deteriorates due to sizable trading costs.
- Despite their opaque nature, ML methods generate economically interpretable trading strategies and are mostly informative for stock selection.
- Beyond economic restrictions, ML signals are profitable in long positions, remain viable in recent years, and command low downside risk.
- In "Post-Fundamentals Drift in Stock Price: A Regression Regularization Perspective", my coauthors and I use easy-going methods to "acceleration" in accounting items.
- Investment payoffs do cross all restrictions studied here.

References

- Avramov, D., T. Chordia, G. Jostova, and A. Philipov, 2013, Anomalies and financial distress. *Journal of Financial Economics* 108:139–159.
- Avramov, D., S. Cheng, and L. Metzker, 2021, Machine Learning versus Economics Restrictions: Evidence from Stock Return Predictability, *Management Science*.
- Avramov, D., S. Cheng, L. Metzker, and S. Voigt, 2021, Integrating Factor Models, Working Paper.
- Avramov, D., G. Kaplanski, and A. Subrahmanyam, 2021, Post Fundamentals Price Drift in Capital Markets: A Regression Regularization Perspective. *Management Science.*
- Belloni, Alexandre, Victor Chernozhukov, and Christian Hansen, 2014, Inference on treatment effects after selection among high-dimensional controls, *The Review of Economic Studies* 81, 608-650.
- Borisenko, Dmitry., 2019, Dissecting Momentum: We Need to Go Deeper. Working paper.
- Chaudhari, Sneha, Gungor Polatkan, Rohan Ramanath, and Varun Mithal. 2019. "An Attentive Survey of Attention Models." ArXiv Preprint ArXiv:1904.02874.
- Charpentier, A., Elie R. and Remlinger C., 2020 Reinforcement Learning in Economics and Finance, arXiv 2003.10014v1
- Chen, L., M. Pelger, and J. Zhu , 2019 , Deep learning in asset pricing. Working Paper.
- Chordia, T., A. Subrahmanyam, and Q. Tong , 2014 , Have capital market anomalies attenuated in the recent era of high liquidity and trading activity? *Journal of Accounting and Economics* 58:51–58.
- Cochrane, John H., 2011, Presidential Address: Discount Rates, *The Journal of Finance* 66, 1047-1108.
- Cong L. W., Tang K., Wang J., and Zhang Y, 2020, Deep Sequence Modeling – Development and applications in asset pricing.
- Cong L. W., Tang K., Wang J., and Zhang Y, 2021, AlphaPortfolio, Direct construction through deep reinforcement learning and interpretable AI, working paper
- Feng, Guanhao, Stefano Giglio, and Dacheng Xiu, 2017, Taming the factor zoo, Working Paper.
- Feng, G., S. Giglio, and D. Xiu , 2019 , Taming the factor zoo. Forthcoming in *Journal of Finance*.

- Frank, I.E. and Friedman, J.H. (1993) An Statistical View of Some Chemometrics Regression Tools, *Technometrics* 35, 109-135.
- Freyberger, J., A. Neuhierl, and M. Weber , 2018 , Dissecting characteristics nonparametrically. Working Paper.
- Fu, Wenjiang J., 1998, Penalized regressions: the bridge versus the lasso, *Journal of computational and graphical statistics* 7, 397-416.
- Gu, S., Kelly, B., Xiu, D., 2019, Autoencoder Asset Pricing Models, Forthcoming in Journal of Econometrics.
- Gu, S., B. Kelly, and D. Xiu , 2019 , Empirical asset pricing via machine learning. Working Paper.
- Harvey, C. R., Y. Liu, and H. Zhu , 2016, ...and the cross-section of expected returns. *Review of Financial Studies* 29:5–68.
- Hoerl, Arthur E., and Robert W. Kennard, 1970, Ridge regression: Biased estimation for nonorthogonal problems, *Technometrics* 12, 55-67.
- Hoerl, Arthur E., and Robert W. Kennard, 1970, Ridge regression: applications to nonorthogonal problems, *Technometrics* 12, 69-82.
- Hou, K., C. Xue, and L. Zhang , 2018, Replicating anomalies. *Forthcoming in Review of Financial Studies*.

- Huang, J., J. L. Horowitz, and F. Wei, 2010, Variable Selection in Nonparametric Additive Models, *Annals of statistics* 38, 2282-2313.
- Kelly, B., Pruitt S., and Su Y., 2019, Characteristics are covariances: a unified model of risk and return, Forthcoming in Journal of Financial Economics.
- Kelly, B., Pruitt S., and Su Y., 2017, Instrumental principal component analysis, working papers, Chicago Booth and ASU WP Carey.
- Kozak, S., S. Nagel, and S. Santosh , 2019 , Shrinking the cross-section. Forthcoming in *Journal of Financial Economics*.
- Lettau, M., and M. Pelger. 2018a. Estimating latent asset-pricing factors. Forthcoming in *Journal of Econometrics*.
- Lettau, M., and M. Pelger. 2018b. Factors that fit the time series and cross-section of stock returns. Working Paper
- Luyang, Chen., Markus, Pelger†., and Jason Zhu, 2019, Deep Learning in Asset Pricing. Working paper.

- McLean, R., and J. Pontiff 2016, Does academic research destroy stock return predictability? *Journal of Finance* 71:5–31.
- Nair, Vinod, and Geoffrey E Hinton, 2010, Rectified linear units improve restricted boltzmann machines, *in Proceedings of the 27th international conference on machine learning* (ICML-10), 807–814
- Pástor, Ľuboš, and Robert F. Stambaugh, 2000, Comparing asset pricing models: An investment perspective. *Journal of Financial Economics* 56 (3): 335-81.
- Pástor, Ľuboš. 2000. Portfolio selection and asset pricing models. *The Journal of Finance* 55 (1): 179-223
- Stambaugh, R. F., J. Yu, and Y. Yuan , 2012 , The short of it: Investor sentiment and anomalies. *Journal of Financial Economics* 104:288–302.
- Wang, Jingyuan, Yang Zhang, Ke Tang, Junjie Wu, and Zhang Xiong, 2019, Alphastock: A buying-winners-and-selling-losers investment strategy using interpretable deep reinforcement attention networks, in Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining pp. 1900-1908.
- Zou, Hui, and Trevor Hastie, 2005, Regularization and variable selection via the elastic net, *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 67, 301-320.