CASE STUDY: DEEP NEURAL NETWORK–BASED EQUITY FACTOR MODEL

The following case study was developed and written by Matthew Dixon, PhD, FRM.

An investment manager wants to select stocks based on their predicted performance using a fundamental equity factor model. She seeks to capture superior performance from stocks with the largest excess return using a non-linear factor model and so chooses a deep neural network to predict the stock returns. The goal of this mini-case study is to demonstrate the application of deep neural networks to fundamental equity factor modeling. We shall focus on using feed-forward (i.e., forward propagation) network regression in place of ordinary least squares linear regression. Since neural networks are prone to over-fitting, we shall use LASSO penalization, the same penalty score–based approach used previously with regression, to mitigate this issue.

Introduction

Cross-sectional fundamental factor models are used extensively by investment managers to capture the effects of company-specific factors on individual securities. A fixed universe of *N* assets is first chosen, together with a set of *K* fundamental factors. Each asset's sensitivity (i.e., exposure or loading) to a fundamental factor is represented by beta, *B*, and the factors are represented by factor returns (f_t). There are two standard approaches to estimating a factor model: (i) adopt time-series regression (TSR) to recover loadings if factors are known or (ii) use cross-sectional regression (CSR) to recover factor returns from known loadings. We shall follow the CSR approach; the factor exposures are used to predict a stock's return (r_t) by estimating the factor returns using multivariate linear regression (where ε_t is the model error at time *t*):

$$r_t = Bf_t + \varepsilon_t.$$

However, this CSR model is too simplistic to capture non-linear relationships between stock returns and fundamental factors. So, instead we use a deep neural network to learn the non-linear relationships between the betas (*B*) and asset returns (r_t) at each time *t*. The goal of deep learning is to find the network weights which minimize the out-of-sample mean squared error (MSE) between the predicted stock returns, $\hat{r}$, and the observed stock returns, *r*. We shall see that simply increasing the number of neurons in the network will increase predictive performance using the in-sample data but to the detriment of out-of-sample performance; this phenomenon is the bias–variance trade-off. To mitigate this effect, we add a LASSO penalty term to the loss function to automatically shrink the number of non-zero weights in the network. In doing so, we shall see that this leads to better out-of-sample predictive performance.

Note that each weight corresponds to a link between a node in the previous and current layer. Reducing the number of weights generally means that the number of connections—not the number of nodes—is reduced. The exception is when all weights from the neurons in the previous layer are set to zero—in which case the number of nodes in the current layer would be reduced. In the special case when the previous layer is the input layer, the number of features is also reduced.

We shall illustrate the data preparation and the neural network fitting using six fundamental equity factors. This choice of number and type of fundamental factor is arbitrary, and an investment manager may use many more factors in her or his model, often representing industry sectors and sub-sectors using dummy variables.

Data Description

A description of the stock price and fundamental equity factor data used for training and evaluating the neural network is shown in Exhibit 31.

Exhibit 31 Dataset of S&P 500 Stocks and Fundamental Factors

Description:

A subset of S&P 500 Index stocks, historical monthly adjusted closing prices, and corresponding monthly fundamental factor loadings.

Time period: June 2010 to November 2018

Number of periods: 101

Number of stocks (*N*): 218 stocks

Number of features (*K*): 6

Features: Fundamental equity factors:

Exhibit 31 (Continued)

1. Current enterprise value (i.e., market values of equity + preferred stock + debt − cash − short-term investments)
2. Current enterprise value to trailing 12-month EBITDA
3. Price-to-sales ratio
4. Price-to-earnings ratio
5. Price-to-book ratio
6. Log of stock's market capitalization (i.e., share price × number of shares outstanding)

Output: Monthly return for each stock over the following month.

We define the universe as the top 250 stocks from the S&P 500, ranked by market capitalization as of June 2010. All stock prices and factor loadings are sourced from Bloomberg. An illustrative extract of the data is given in Exhibit 32. Note that after removing stocks with missing factor loadings, we are left with 218 stocks.

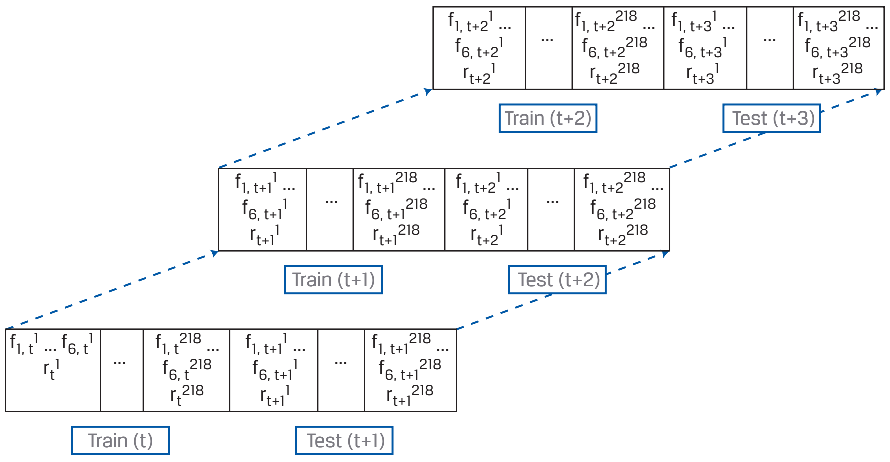
Exhibit 32 Extract of Six Factor Loadings and Return for Three Selected Stocks

TICKER	CURR_EV ($ Mil.)	CURR_EV_ TO_T12M_ EBITDA (X)	PX_ TO_ SALES (X)	PX_ TO_ EARN (X)	PX_ TO_ BOOK (X)	LOG_ CAP ($ Mil.)	RETURN (%)
SWK	10,775.676	30.328	1.138	16.985	1.346	9.082970	−0.132996
STZ	7,433.553	15.653	1.052	10.324	1.480	8.142253	−0.133333
SRE	19,587.124	10.497	1.286	10.597	1.223	9.314892	−0.109589

Experimental Design

The method used to train the deep neural network is time-series cross-validation (i.e., walk-forward optimization), as depicted in Exhibit 33. At each time period, the investment manager fits a new model; each factor (f_1 to f_6) is a feature in the network, and the loadings of the factors for each stock is a feature vector observation (i.e., the set of observations for each stock for each period), leading to N = 218 observations of pairs of feature vectors and output (monthly return, r_t) in the training set per period. The network is initially trained at period t, and then it is tested over the next period, t +1, which also has N = 218 observations of pairs of feature vectors and output. In the next iteration, the t + 1 data become the new training set and the revised model is tested on the t + 2 data. The walk-forward optimization of the neural network continues until the last iteration: model training with t + 99 data (from Period 100) and testing with t + 100 data (from the last period, 101).

Exhibit 33 Time-Series Cross-Validation on Asset Returns (Walk-Forward Optimization)—The First Three Iterations



$f_{1,t}^{1}$ … $f_{6,t}^{1}$ r_{t}^{1}
…
$f_{1,t}^{218}$ … $f_{6,t}^{218}$ r_{t}^{218}
$f_{1,t+1}^{1}$ … $f_{6,t+1}^{1}$ r_{t+1}^{1}
$f_{1,t+1}^{218}$ … $f_{6,t+1}^{218}$ r_{t+1}^{218}
$f_{1,t+2}^{1}$ … $f_{6,t+2}^{1}$ r_{t+2}^{1}
$f_{1,t+2}^{218}$ … $f_{6,t+2}^{218}$ r_{t+2}^{218}
$f_{1,t+3}^{1}$ … $f_{6,t+3}^{1}$ r_{t+3}^{1}
$f_{1,t+3}^{218}$ … $f_{6,t+3}^{218}$ r_{t+3}^{218}
Train (t)
Test (t+1)
Train (t+1)
Test (t+2)
Train (t+2)
Test (t+3)

We use a feed-forward neural network with six input nodes (i.e., neurons), two hidden layers, and one output neuron. There are 50 neurons in each hidden layer to intentionally over-specify the number of parameters needed in the model, meaning bias (variance) is substantially lower (higher) than optimal. LASSO penalization is then used to automatically shrink the parameter set. Additionally, it is important for the number of nodes in each hidden layer not to exceed the number of observations in the training set (50 nodes per layer versus 218 observations). The model training in period *t* involves finding the optimal bias-versus-variance trade-off. Once fitted, we record the in-sample MSE and the out-of-sample MSE in addition to the optimal regularization parameter. This procedure is then repeated sequentially over the horizon of 100 remaining periods, tuning the hyperparameters at each stage using cross-validation. The end result of this procedure is a fitted model, trained monthly on the current cross-sectional data and for which hyperparameters have been tuned at each step.

Results

Exhibit 34 presents the results from model evaluation; it compares the in-sample and out-of-sample MSEs of the deep neural network over all 101 months. Note that the out-of-sample error (dotted line) is typically significantly larger than the in-sample error (solid line). However, as the time periods pass and the model is repeatedly trained and tested, the difference between the out-of-sample and in-sample MSEs narrows dramatically.

Exhibit 34 In-Sample and Out-of-Sample MSE for Each Training and Testing Period

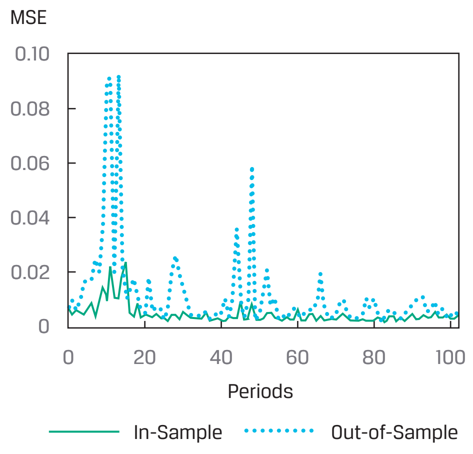
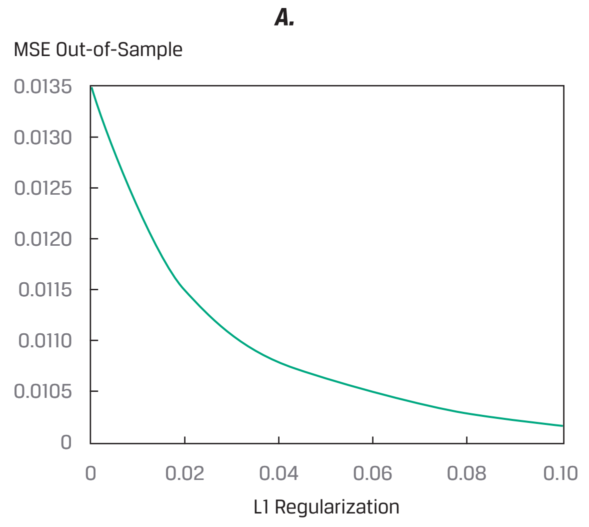
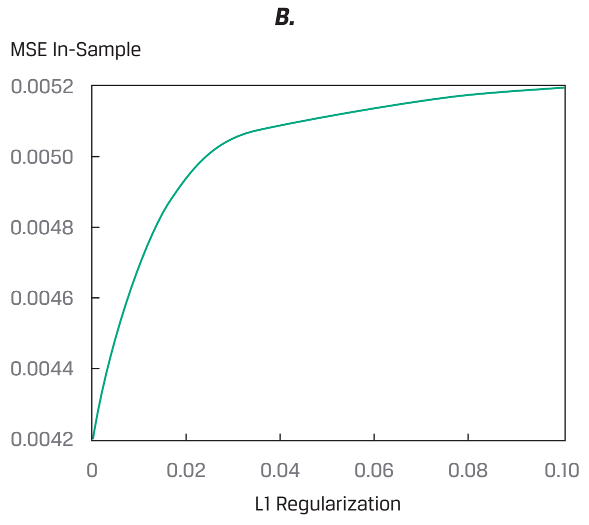


Exhibit 35 shows the effect of LASSO regularization on the in-sample MSE (lower panel, B) and the out-of-sample MSE (upper panel, A) for the first iteration of the time-series cross-validation (training with data from period t and testing with data from period t +1). The degree of LASSO regularization needed is found by cross-validation using 50 neurons in each hidden layer. Increasing the LASSO regularization, which reduces the number of non-zero weights in the model, introduces more bias and hence increases the in-sample error. Conversely, increasing the LASSO regularization reduces the model's variance and thereby reduces the out-of-sample error. Overall, the amount of LASSO regularization needed is significant, at 0.10; typically the regularization hyperparameter is between 0.001 and 1.0. Also, the out-of-sample and in-sample MSEs have not yet converged. There is still a substantial gap, of roughly 0.0051 (= 0.01025 – 0.0052), and the slope of the curves in each plot suggests the optimal value of the regularization hyperparameter is significantly more than 0.10. Note that the value of the regularization hyperparameter is not interpretable and does not correspond to the number of weights eliminated. Suffice it to say, the larger the value of the regularization hyperparameter, the more the loss is being penalized.

Exhibit 35 LASSO Regularization for Optimizing Bias–Variance Trade-Off (First Iteration)

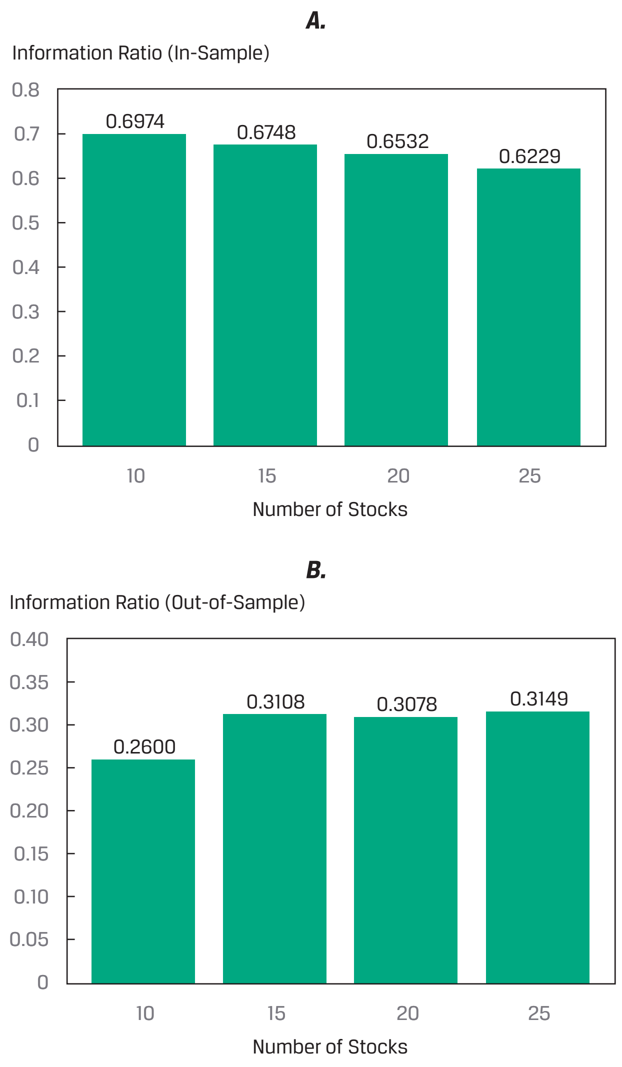




It is important to recognize that although the out-of-sample MSE of this deep learning neural network is key to characterizing its predictive performance, it does not necessarily follow that a stock selection strategy based on the neural network will be successful. This is because the neural network predicts the next month's expected (i.e., mean) asset returns and not the full distribution of returns. Hence a simple stock selection strategy—measured by information ratios (recall the information ratio, or IR, is alpha divided by nonsystematic risk, so it measures the abnormal return per unit of risk for a well-diversified portfolio) of the portfolio returns—that selects stocks ranked by predicted returns will not necessarily lead to positive information ratios.

Exhibit 36 presents the information ratios found by back-testing a simple stock selection strategy that picks the top performing stocks determined by the neural network's forecasted returns realized in month t +1 using features observed in month t. Note these IRs do not account for transaction costs, interest rates, or any other fees. The upper panel (A) shows the best-case scenario; the neural network in-sample prediction is used to select the n (where n is 10, 15, 20, or 25) top performing stocks. The IRs are shown for each of the different-sized portfolios; they range from 0.697 to 0.623. Note that as a rule of thumb, IRs in the range of 0.40–0.60 are considered quite good. The lower panel (B) shows the IRs from back-test results for the same strategy applied to the out-of-sample data. The out-of-sample IRs range from 0.260 to 0.315 and so are substantially smaller than in-sample IRs.

Exhibit 36 Information Ratios from Back-Testing a Stock Selection Strategy Using Top Performers from the Neural Network



Importantly, the out-of-sample performance provides the most realistic assessment of the likely future investment performance from applying this deep learning neural network to stock selection. It is a baseline for further model refinements, including adding more fundamental and macroeconomic factors. With such refinements, it can be expected that the out-of-sample IRs should improve substantially.