Empirical Asset Pricing via Machine Learning

Huei-Wen Teng[1] Ming-Hsiu Hu[2]

Dept. of Information Management and Finance
National Chiao Tung University

June 2020

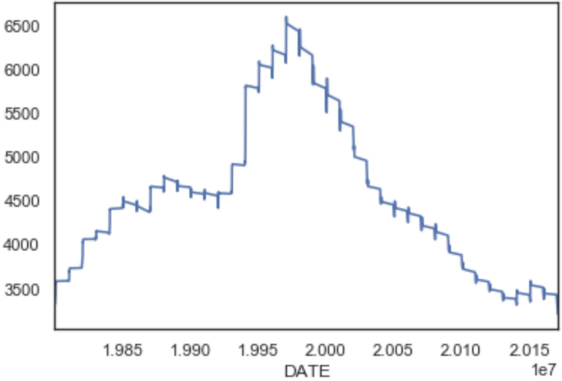
Outline

- Data Pre-processing
- Exploratory Data Analysis
- High Dimensional Regression
  1. Linear Regression
  2. Ridge Regression
  3. Lasso Regression
- Neural Network
  1. Deep Feed-Forward Neural Network
  2. 1D Convolution Neural Network
  3. LSTM

Exploratory Data Analysis

Company Number by time

- We plot the number of companies bar chart by time



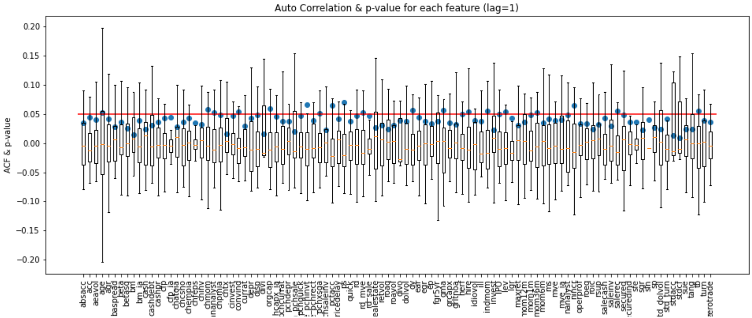
- Lag-l Sample Auto Correlation of r_t is defined as:

$$\hat{\rho}_\ell = \frac{\sum_{t=\ell+1}^{T}(r_t - \bar{r})(r_{t-\ell} - \bar{r})}{\sum_{t=1}^{T}(r_t - \bar{r})^2}, \text{ where } 0 \leq \ell < \mathrm{T}$$

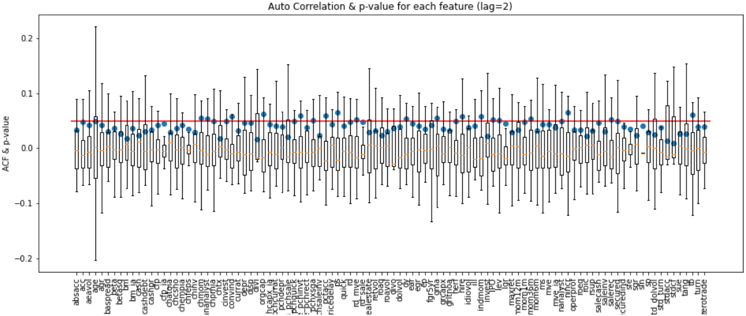
- $H_o : \rho_\ell = 0$ v.s. $H_a : \rho_\ell \neq 0$
- We use t-ratio defined as below to test each feature's p-value:

$$\text{t-ratio} = \frac{\hat{\rho}_\ell}{\sqrt{(1+2\sum_{i=1}^{\ell-1} \hat{\rho}_i{}^2)/T}}$$

- We plot the boxplot of ACF by each company's 102 features (lag=1) and ACF's p-value of each feature(blue dots).
- There are around 84 percent of features's ACF p-value < 0.05.



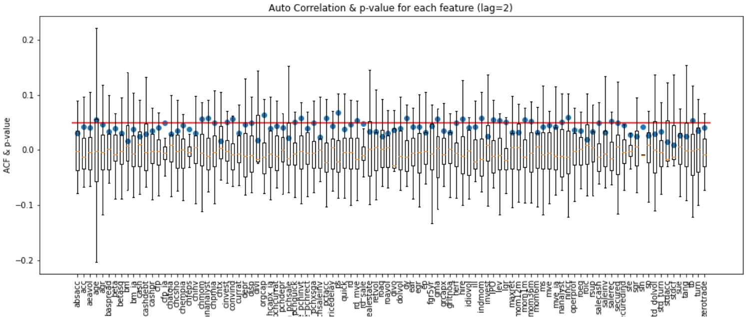
- We plot the boxplot of ACF by each company's 102 features (lag=2) and ACF's p-value of each feature(blue dots).
- There are around 80 percent of features's ACF p-value < 0.05.



Exploratory Data Analysis

Auto Correlation Factor (lag=3)

- We plot the boxplot of ACF by each company's 102 features (lag=3) and ACF's p-value of each feature(blue dots).
- There are around 76 percent of features's ACF p-value < 0.05.



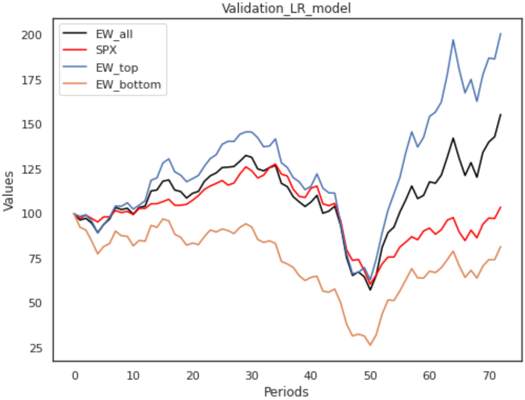
- Linear Regression: $\min_{\beta} \sum_{i=1}^{n}(y_i - (X_i)^T\beta)^2$
- Linear Regression's estimate: $\hat{\beta} = (x^T x)^{-1} x^T y$

- Linear Regression: $\min_{\beta} \sum_{i=1}^{n}(y_i - (X_i)^T\beta)^2$
- Linear Regression's estimate: $\hat{\beta} = (x^T x)^{-1} x^T y$

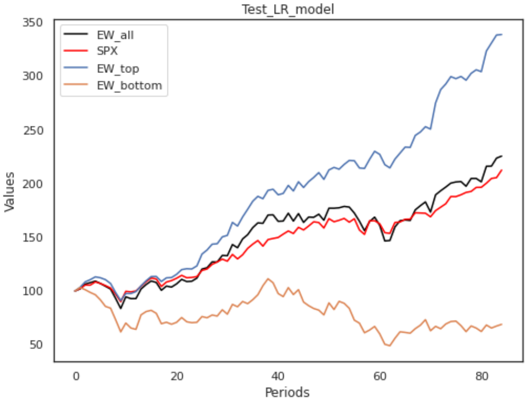
High Dimensional Regression

Linear Regression

- Validation MSE = 0.0357
- Validation R^2_{OOS} = -0.0836



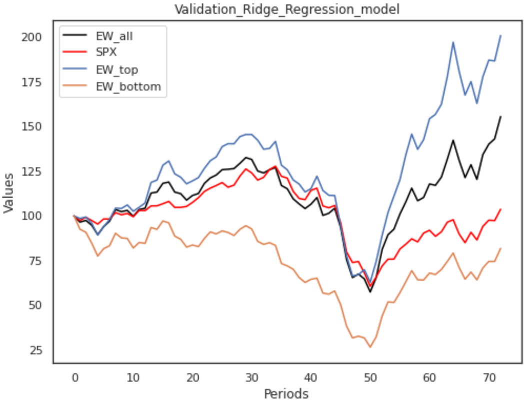
- Test MSE $= 0.0274$
- Test $R^2_{OOS} =$ -0.1728



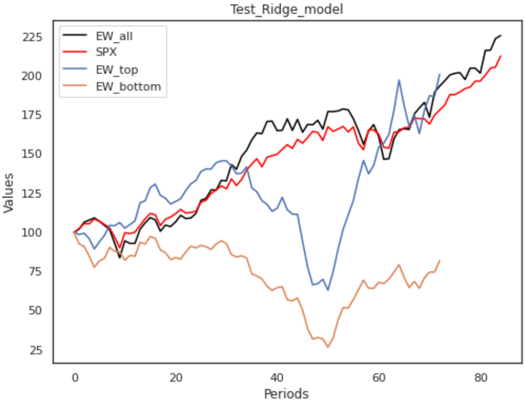
- Ridge Regression: $\min_{\beta} \sum_{i=1}^{n} (y_i - (X_i)^T \beta)^2 + \lambda \sum_{j=1}^{p} \beta_j^2$
- Here $\lambda \geq 0$ is the tuning parameter
  1. When $\lambda = 0$: we get the linear regression
  2. When $\lambda = \infty$: we get $\hat{\beta_{bridge}} = 0$
  3. For λ in between, we are balancing the two ideas: fitting a linear model of y on x, and shrinking the coefficients.

- We use GridSearchCV for hyperparameter tuning.
- Choose $\lambda = 50$ out of [0.0001, 0.001, 0.01, 0.1, 1, 5, 10, 20, 25, 30, 35, 40, 45, 50]

- Validation MSE = 0.0342
- Validation R^2_{OOS} = -0.0856



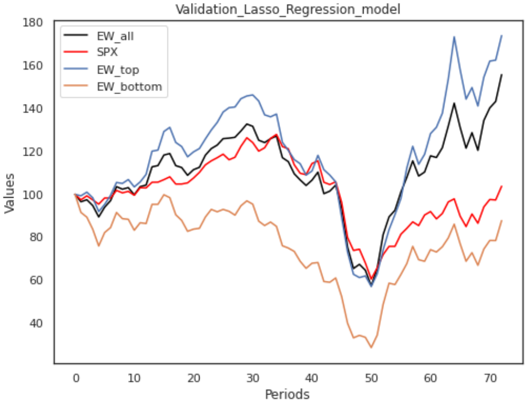
- Test MSE = 0.0253
- Test R^2_{OOS} = -0.0404



- Lasso Regression: $\min_{\beta} \sum_{i=1}^{n}(y_i - (X_i)^T\beta) + \lambda \sum_{j=1}^{p} \mid \beta_j \mid$
- Replace the 2-norm in Ridge Regression with 1-norm
- Main differences between Ridge Regression and Lasso Regression: Lasso Regression is able to perform variable selection in linear model.

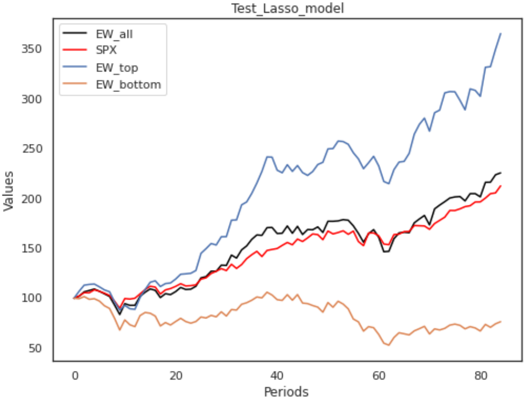
- We use GridSearchCV for hyperparameter tuning.
- Choose $\lambda = 0.001$ out of [0.00001, 0.0001, 0.001, 0.01, 0.1, 1]

- Validation MSE = 0.0329
- Validation R^2_{OOS} = -0.0013



High Dimensional Regression
Lasso Regression

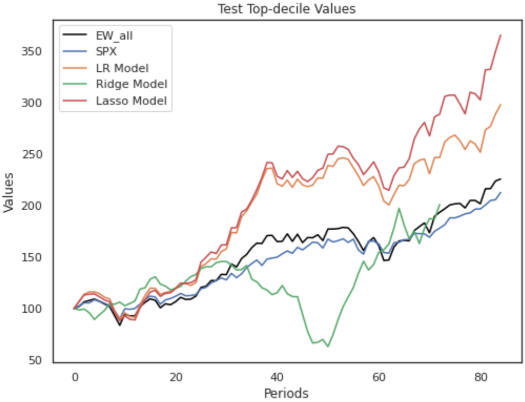
- Test MSE $= 0.0232$
- Test $R^2_{OOS} = 0.0044$



High Dimensional Regression

Regression Long Portfolio Comparison

- Compare Linear Regression / Ridge Regression / Lasso Regression with Long Top-decile portfolio



- The results align with Gu's paper
- Vast predictor sets are viable for linear prediction when either penalization or dimension reduction is used.
- Allowing for nonlinearities substantially improves predictions

- Below is the shape of a single data (company) for 1D-CNN input
- $x_{p,t}$ means the value of p factor for a single company at time t
- $$X_{p,T} = \begin{pmatrix} x_{1,1} & x_{2,1} & \cdots & x_{p,1} \\ x_{1,2} & x_{2,2} & \cdots & x_{p,2} \\ \vdots & \vdots & \ddots & \vdots \\ x_{1,T} & x_{2,T} & \cdots & x_{p,T} \end{pmatrix}$$

- Here we illustrate 1D-CNN architecture using one single data (company X).

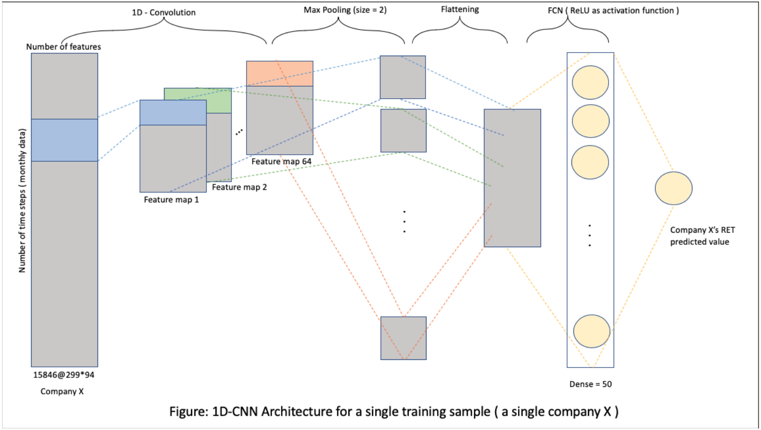
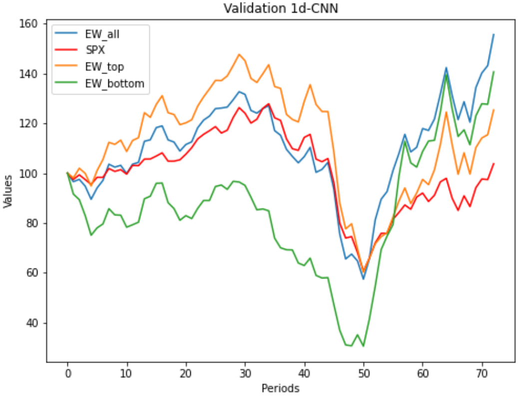
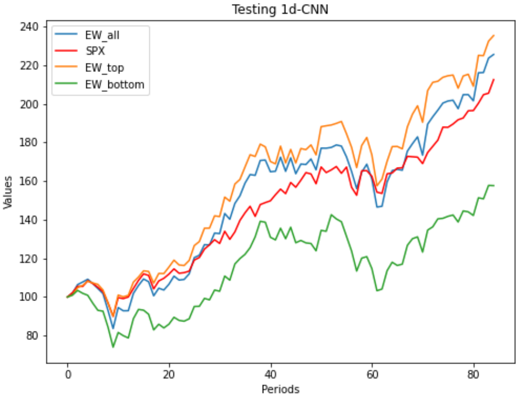


Figure: 1D-CNN Architecture for a single training sample

- Validation MSE $= 0.000234$
- Validation $R^2_{OOS} = 0.33$



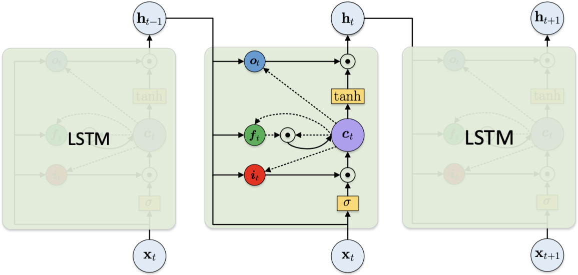
- Testing MSE $= 0.000332$
- Testing $R^2_{OOS} = 0.29$



- LSTM networks are belong to the class of recurrent neural networks (RNNs).
- It has been introduced by Hochreiter and Schmidhuber (1997) and were further refined in the following years until now.
- LSTM networks are specifically designed to learn long term dependencies and are capabale of overcoming the previously inhernet problems of RNNs, such as vanishing and exploding gradients for large time step(Sak, Senior, Beaufays, 2014).

- LSTM networks are composed of an input layer, one or more hidden layers, and an output layer.
- The number of neurons in the input layer is equal to the number of explanatory variables (which we often called features).
- The number of neurons in the output layer reflects the output space.
- In our question, we have one neuron since we would like to predict the Holding Period Return at time t for each given company's features at time t-1.

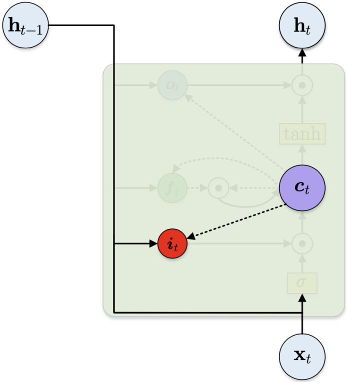
- LSTM can "preserve" the earlier hidden node activations for prediction at current time t (Hochreiter and Schmidhuber, 1997).
  - remove and add information to the cell.
  - gated mechanism.



- Memory Cell consists of input gate i_t, forget gate f_t, cell c_t, output gate o_t and hidden state h_t which are operated as:

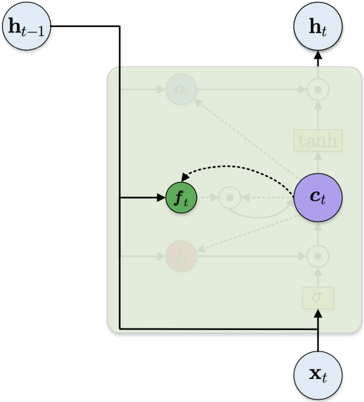
$$
\begin{aligned}
i_t &= \sigma(W_{xi}x_t + W_{hi}h_{t-1} + W_{ci}c_{t-1} + b_i) \\
f_t &= \sigma(W_{xf}x_t + W_{hf}h_{t-1} + W_{cf}c_{t-1} + b_f) \\
c_t &= f_t c_{t-1} + i_t \tanh(W_{xc}x_t + W_{hc}h_{t-1} + b_c) \\
o_t &= \sigma(W_{xo}x_t + W_{ho}h_{t-1} + W_{co}c_t + b_o) \\
h_t &= o_t \tanh(c_t)
\end{aligned}
$$

- Input gate: Decide what to store

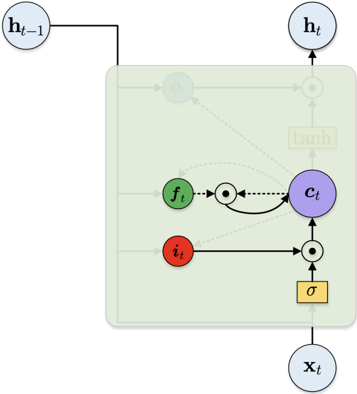


Neural Network
LSTM

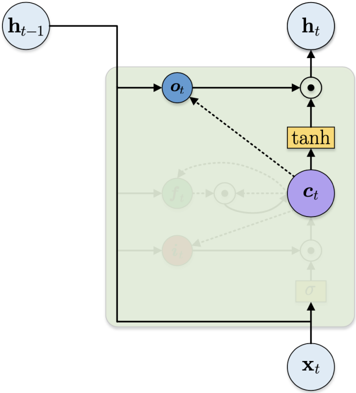
- Forget gate: Decide what to throw away



- Update cell: Update the cell state



- Output gate: Decide what the output is



- We perform LSTM because some of the features are time-correlated shown by the ACF box-plot.
- Below is the matrix of a single input(company) we put in LSTM.
- $k_{p,t}$ means the value of p factor for a single company at time t

$$K_{p,T} = \begin{bmatrix} k_{1,1} & k_{2,1} & \cdots & k_{p,1} \\ k_{1,2} & k_{2,2} & \cdots & k_{p,2} \\ \vdots & \vdots & \ddots & \vdots \\ k_{1,T} & k_{2,T} & \cdots & k_{p,T} \end{bmatrix}$$

- We perform different time-step (rolling window) in the LSTM network
- Below is an input of company 1 of choosing features from t=0 to t=3
- The target y_1 is the return of company 1 at t=4

$$x_{company1} = \begin{bmatrix} k_{1,1} \\ k_{1,2} \\ \vdots \\ k_{p,3} \end{bmatrix} = K_{p,3}^T$$

- The number of total companies cross time:
  1. training: 15846
  2. validation: 5889
  3. testing: 4914
- Furthermore, the length of each single data, whether in training, validation or testing, is different due to the period of existing for each company is different. This is the reason why we prefer to construct our data company by company cross time, not month by month cross company. Apparently, it would be easier for us to construct our model later.

Neural Network
LSTM

- training dataset shape:
  1. torch.Size([1426233, 110, 1]) torch.Size([1426233, 1])
- validation dataset shape:
  1. torch.Size([291493, 110, 1]) torch.Size([291493, 1])
- testing dataset shape:
  1. torch.Size([283855, 110, 1]) torch.Size([283855, 1])

- We choose time-step $= 1$ month for each input company
- Optimizer: adaptive moment estimation algorithm (Adam), an efficient version of the SGD introduced by Kingma and Ba (2014).
- Criterion: Mean Square Error (reduction = 'mean')
- Furthermore, we random shuffle the data for each batch input.

- LSTM Model Setting:
  1. input dimension = 110
  2. hidden dimension = 128
  3. number of hidden layers = 1
  4. output dimension = 1
- Number of parameters for the model: 67201
- Hyperparameters:
  1. batch size = 512
  2. epoch = 500
  3. learning rate = 0.0001