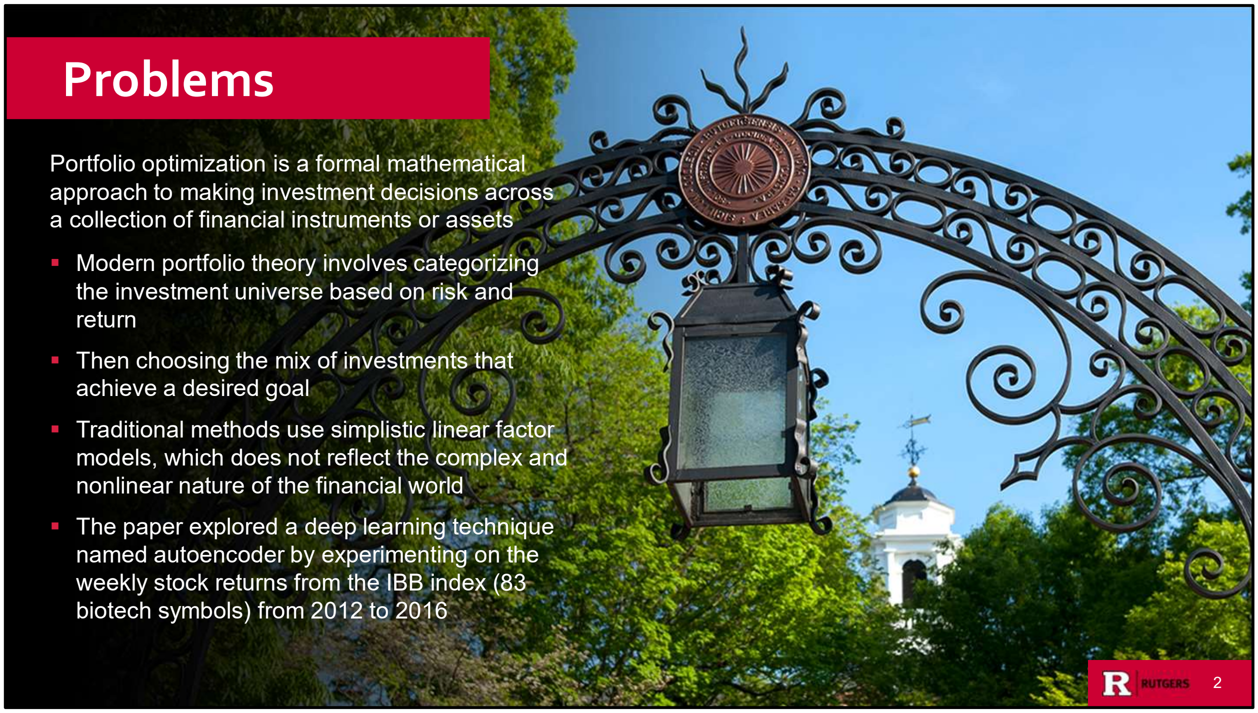
Portfolio with Deep Learning

Rick Shen
10/8/2024



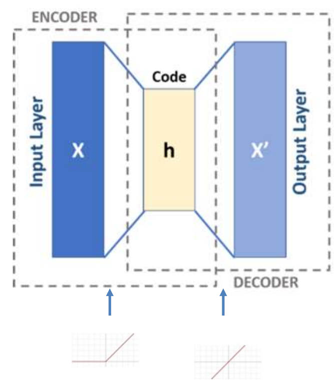
https://www.mathworks.com/discovery/portfolio-optimization.html

Heaton et.al., *Appl. Stochastic Models Bus. Ind.* ***2017***, 33 3–12

Actual code and data from Derek Snow

https://drive.google.com/drive/folders/1-hOEAiJqaNTUYIyamj26ZvHJNZq9XV09

Model



I. Autoencoding (unsupervised neural network)
   - 83 inputs $\boldsymbol{X}$ → 1 hidden layer (*latent space*) with 5 neurons → 83 outputs $\boldsymbol{X}'$
   - Regularized by 2-norm $\|\boldsymbol{X} - \boldsymbol{X}'\|_2$
   - Aim to encode $\boldsymbol{X}$ and create a more information-efficient representation of itself
   - Rank stocks by 2-norm difference then select subsets of them as our portfolio

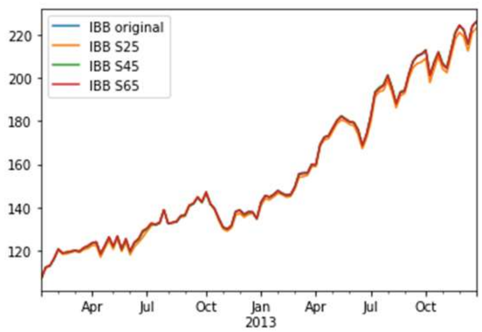
II. Calibrating (supervised artificial neural network)
   - 15/ 45/ 65 selected subsets of $\boldsymbol{X}$ → 1 hidden layer with 5 neurons → 1 output $\boldsymbol{y}'$
   - Aim to find weights to minimize 2-norm $\|\boldsymbol{y} - \boldsymbol{y}'\|_2$, with $\boldsymbol{y}$ being the IBB index

III. Validating (out-of-sample testing)

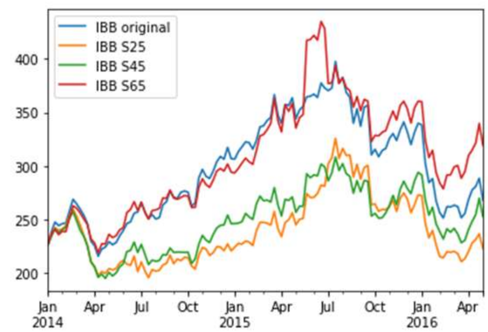
https://en.wikipedia.org/wiki/Autoencoder
Calibrate – weekly returns of all 83 from 01/2012 – 12/2013
Validate –subsets from 01/2012 – 12/2013

Results

- In-sample testing



- Out-of-sample testing



Portfolio was chosen by top 10 stocks whose 2-norm was the smallest, plus (s-10) of the bottom ones, s = 25, 45, 65
Out-of-sample testing includes the “2015-2016 market selloff” period.
https://en.wikipedia.org/wiki/2015%E2%80%932016_stock_market_selloff

Assessments

- The ReLU activation function introduces nonlinearity to the Autoencoder model
  - It can be interpreted as compositions of put and call options
  - The abstract features in the latent space can then be thought of as "deep portfolios"
- Autoencoder vs. Principal Component Analysis (PCA) –dimension reduction techniques
  - Nonlinear
  - Less interpretable
  - Better flexibility (number of layers/ neurons, supervised capable)
  - Can be used as target for other supervised learning techniques
- Comparison of the same experiment done with PCA can be revealing
- Author used top 10 stocks combined with the bottom (s-10), with s = 25, 45, 65, in terms of 2-norm differences from Autoencoder, to construct the portfolios
  - What if only the top ones are used?

https://medium.com/@etorezone/differences-between-autoencoders-and-principal-component-analysis-pca-in-dimensionality-reduction-ca5f24364054

- An Autoencoder model was trained to compress weekly returns of 83 stocks between 2012 to 2013 to reconstruct the data itself
- Based on the model output, subsets of the stock index were selected
- With the selected subsets, a separate 2-layer ANN model was constructed to match the stock index

RUTGERS

Backup Slide – Autoencoder Code

```
encoding_dim = 5 # 5 neurons
num_stock = len(stock_lp.columns) # Use 83 stocks as features

# connect all layers
input_img = Input(shape=(num_stock, ))
encoded = Dense(encoding_dim, activation='relu', kernel_regularizer=regularizers.l2(0.01))(input_img)
decoded = Dense(num_stock, activation= 'linear', kernel_regularizer=regularizers.l2(0.01))(encoded) # see 'Stacked Auto-Encoders' in paper

# construct and compile AE model
autoencoder = Model(input_img, decoded)
autoencoder.compile(optimizer='sgd', loss='mean_squared_error')

# train autoencoder on weekly stock price changes
data = stock['calibrate']['net']
autoencoder.fit(data, data, shuffle=False, epochs=500, batch_size = 10)
autoencoder.save('output/retrack_autoencoder.h5')

# test/reconstruct market information matrix
reconstruct = autoencoder.predict(data)
```

Backup Slide – ANN Calibration Code

```
ibb_predict = defaultdict(defaultdict)
total_2_norm_diff = defaultdict(defaultdict)
dl_scaler = defaultdict(StandardScaler)

for non_communal in [15, 35, 55]:
    # some numerical values
    encoding_dim = 5
    s = 10 + non_communal
    stock_index = np.concatenate((ranking[0:10], ranking[-non_communal:])) # portfolio index


    # connect all layers
    input_img = Input(shape=(s,))
    encoded = Dense(encoding_dim, activation='relu', kernel_regularizer=regularizers.l2(0.01))(input_img)
    decoded = Dense(1, activation= 'linear', kernel_regularizer=regularizers.l2(0.01))(encoded)


    # construct and compile deep learning routine
    deep_learner = Model(input_img, decoded)
    deep_learner.compile(optimizer='sgd', loss='mean_squared_error')

    x = stock['calibrate']['percentage'].iloc[:, stock_index]
    y = ibb['calibrate']['percentage']

    dl_scaler[s] = StandardScaler()       # Multi-layer Perceptron is sensitive to feature scaling, so it is highly recommended to scale your data
    dl_scaler[s].fit(x)
    x = dl_scaler[s].transform(x)

    deep_learner.fit(x, y, shuffle=False, epochs=500, batch_size = 10)    # fit the model
    deep_learner.save('output/retrack_s' + str(s) + '.h5') # for validation phase use


    # is it good?
    relative_percentage = copy.deepcopy(deep_learner.predict(x))
    relative_percentage[0] = 0
    relative_percentage = (relative_percentage /100) + 1

    ibb_predict['calibrate'][s] = ibb['calibrate']['lp'][0] * (relative_percentage.cumprod())
    total_2_norm_diff['calibrate'][s] = np.linalg.norm((ibb_predict['calibrate'][s] - ibb['calibrate']['lp']))
```