# A Neural Network for Stock Return Prediction?

Stefan Pohle, Prosper Kwabena Adjei, Philipp Schulze

Machine Learning with TensorFlow - Final Project

June 30th 2020

## Outline

## Repository

- The cleaned data set, data preparation files, source code and cited papers are stored in the following repository: **https://github.com/bergsteve/stock-return**

## Introduction and goal of the estimation

- Problem/Research question: Can future stock returns be predicted using publicly available information (past returns)?
- Task to solve: We want to test the weak form of market efficiency:
  - Prices reflect information about past prices.
  - Information set = all historical security prices.
- Method: We train a Neural Network with past returns as features and the next periods return as label.

## Characteristics of the dataset

- Gu, S., Kelly, B., Xiu, D., 2020. Empirical Asset Pricing via Machine Learning. The Review of Financial Studies 33, 2223–2273.
- Sample begins in March 1957 (start date of the S&P 500) and ends in December 2016, totaling 60 years.
- The number of stocks in the sample is almost 30000, with the average number of stocks per month exceeding 6200.
- The data set contains:
  - Monthly total individual equity returns from CRSP for all firms listed in the NYSE, AMEX, and NASDAQ.
  - 94 stock-level predictive characteristics based on the cross-section of stock returns literature.
  - 74 industry dummies corresponding to the first two digits of Standard Industrial Classification (SIC) codes.

## Used technical environment

- Data cleaning/preprocessing: **Excel**, **PyCharm**, **R Studio**.
- Model fitting and evaluation: **Google Colaboratory**.

## Data preparation steps (1)

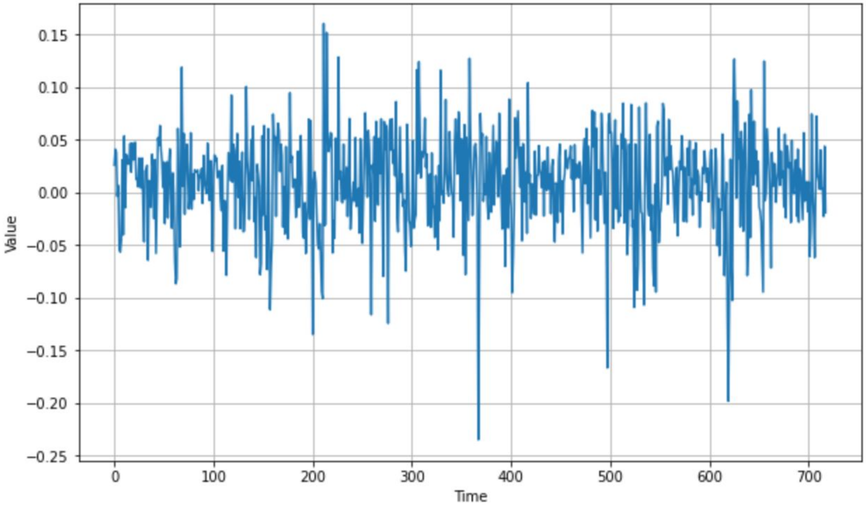
- The full data set was provided in a csv file, which is very large (3.41 GB) and therefore not uploaded.
- The data preparation steps are as follows:

1. Piecewise extraction of rows (individual stocks) from the full data set to handle it's size.
   - This is done using the Python script **get_past_2005_lines.py** with **PyCharm** on a local machine.
   - The script is provided in the repository.
2. The resulting csv files are loaded into **R** and columns not needed are removed (we only need **permno**, **DATE**, **mvel1** and **mom1m**).
   - Descriptions of the variables can also be found in the Online Appendix of Gu et al. (2020).

## Data preparation steps (2)

3. Afterwards, the individual data frames are merged again.
4. The variable **mom1m** is lagged by one month to obtain monthly total individual returns.
   - We call the new variable **return** and remove **mom1m** as well as all observations having **NA** in a column.
5. We build a value weighted index by averaging all stock returns available at a particular date, weighted by their share of total market capitalization at that particular date.
6. We remove all columns but **date** and **return**.
   - The clean data set contains a univariate time series of 718 observations and is stored as **data_complete.csv** in the repository, together with the R script **dataprep.R**.

## Full Time Series

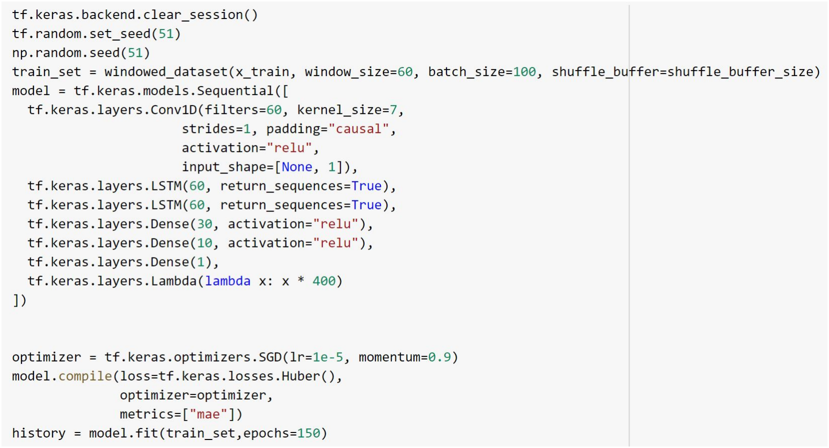


## Description of the model

- We split the data set in a training set (70%, 503 obs), a validation set (20%, 143 obs) and test set (10%, 72 obs).
- We first use the Learning Rate Scheduler and determine the optimal learning rate as $10^{-5}$ (not shown in the slides).

## Model selection (1)

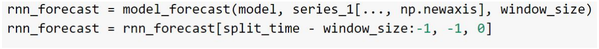
- We first train the model on the (standardized) training data set and assess its performance on the validation data set. We adjust parameters and the number of layers to improve the performance.



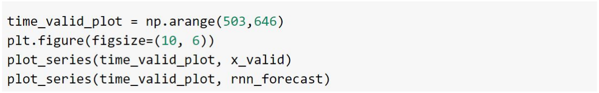
```
tf.keras.backend.clear_session()
tf.random.set_seed(51)
np.random.seed(51)
train_set = windowed_dataset(x_train, window_size=60, batch_size=100, shuffle_buffer=shuffle_buffer_size)
model = tf.keras.models.Sequential([
  tf.keras.layers.Conv1D(filters=60, kernel_size=7,
                      strides=1, padding="causal",
                      activation="relu",
                      input_shape=[None, 1]),
  tf.keras.layers.LSTM(60, return_sequences=True),
  tf.keras.layers.LSTM(60, return_sequences=True),
  tf.keras.layers.Dense(30, activation="relu"),
  tf.keras.layers.Dense(10, activation="relu"),
  tf.keras.layers.Dense(1),
  tf.keras.layers.Lambda(lambda x: x * 400)
])


optimizer = tf.keras.optimizers.SGD(lr=1e-5, momentum=0.9)
model.compile(loss=tf.keras.losses.Huber(),
              optimizer=optimizer,
              metrics=["mae"])
history = model.fit(train_set,epochs=150)
```

## Model selection (2)

- We compute the forecasts, plot them against the validation data set and compute the MAE.

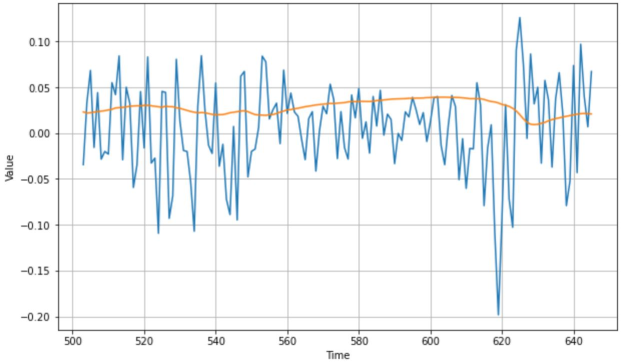


```
rnn_forecast = model_forecast(model, series_1[..., np.newaxis], window_size)
rnn_forecast = rnn_forecast[split_time - window_size:-1, -1, 0]
```



```
time_valid_plot = np.arange(503,646)
plt.figure(figsize=(10, 6))
plot_series(time_valid_plot, x_valid)
plot_series(time_valid_plot, rnn_forecast)
```

## Model selection (3)

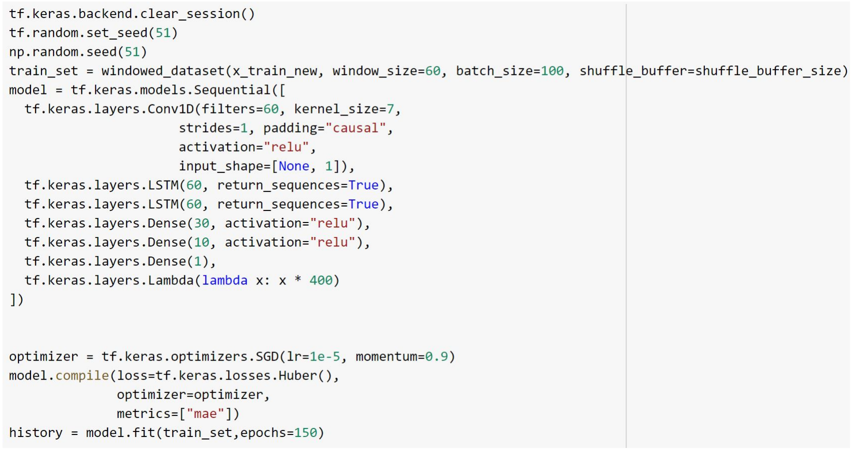


```
tf.keras.metrics.mean_absolute_error(x_valid, rnn_forecast).numpy()
```

```
0.044196643
```

## Application to test data set (1)

- Thereafter, we train the model again using the (standardized) training **and** validation data sets. The optimal learning rate is $10^{-5}$ again.



```
tf.keras.backend.clear_session()
tf.random.set_seed(51)
np.random.seed(51)
train_set = windowed_dataset(x_train_new, window_size=60, batch_size=100, shuffle_buffer=shuffle_buffer_size)
model = tf.keras.models.Sequential([
  tf.keras.layers.Conv1D(filters=60, kernel_size=7,
                      strides=1, padding="causal",
                      activation="relu",
                      input_shape=[None, 1]),
  tf.keras.layers.LSTM(60, return_sequences=True),
  tf.keras.layers.LSTM(60, return_sequences=True),
  tf.keras.layers.Dense(30, activation="relu"),
  tf.keras.layers.Dense(10, activation="relu"),
  tf.keras.layers.Dense(1),
  tf.keras.layers.Lambda(lambda x: x * 400)
])


optimizer = tf.keras.optimizers.SGD(lr=1e-5, momentum=0.9)
model.compile(loss=tf.keras.losses.Huber(),
             optimizer=optimizer,
             metrics=["mae"])
history = model.fit(train_set,epochs=150)
```

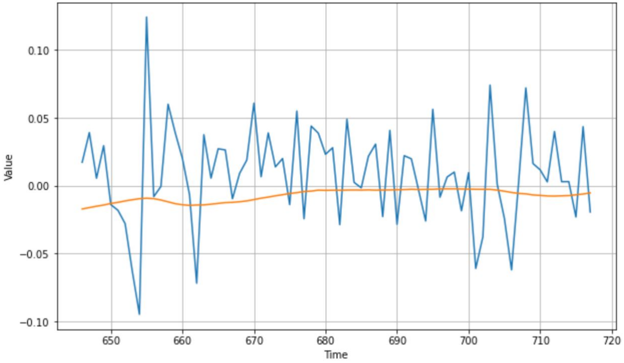
## Application to test data set (2)

- We then assess the performance of the model on the test data set (compute the forecasts, plot them against the test data set and compute the MAE).

```
rnn_forecast = model_forecast(model, series[..., np.newaxis], window_size)
rnn_forecast = rnn_forecast[split_train - window_size:-1, -1, 0]
```

```
plt.figure(figsize=(10, 6))
plot_series(time_test, x_test)
plot_series(time_test, rnn_forecast)
```

## Application to test data set (3)



```
tf.keras.metrics.mean_absolute_error(x_test, rnn_forecast).numpy()
```

```
0.031442568
```

## Results

- The MAE is around 3% while most of the observations are within the range from -5% to 5%.
- The Neural Network does not seem to provide reliable predictions.
- This is as expected, since we only used past prices as predictors.
- The result is also in line with the results of the rolling ridge regression emoloyed in Martin, I., Nagel, S., 2020. Market Efficiency in the Age of Big Data. NBER Working Paper No. 26586, who do not find OOS predictability either.
- Our results do not provide evidence against the Efficient Market Hypothesis in its weak form.

## Difficulties encountered

- The full dataset was too large to open/use it directly to preprocess the data.
- In our setup we end up training the Neural Network on 646 observations, which is a very small number compared to other applications of Neural Networks.
- Another approach would be not to aggregate the stocks to an index, but this would require to extract returns of the 6200 individual stocks (which change over time) and a multivariate model, which was not feasible for us.
- Further improvements/possible regularization techniques would be:
  - Early stopping
  - Batch normalization
  - Ensembles