Comparing Autoencoder and Wavelet Methods for Stock Price Prediction

Andy Liu - Dasion Summary Report

August 2020

1 Introduction

In this report, I summarize the work I completed during my time spent with Dasion over the Summer of 2020. This consisted of studying and implementing predictive models for one-day-ahead financial time series forecasting. I mainly focused on models that implemented one of two preprocessing techniques - autoencoders and wavelet transform methods. However, even within this limited subset of predictive models, I found and implemented a variety of different approaches, which I will explain below. Implementations, as well as more detailed writeups, can be found at the Github repository https://github.com/Dasion-S2020/andy.

1.1 Autoencoders

Autoencoders are a type of unsupervised data representation method, similar to methods such as Principal Component Analysis. It can be used for feature selection and dimensionality reduction; however, it has the advantage of being nonlinear, and therefore more easily flexible when considering different data. Thus, less financial expertise is needed by the creator of the model during the feature selection process.

Mechanically, autoencoders are neural networks with "encoder" and "decoder" neuron components. These neural networks will learn a "reduction" side, where dimensionality is reduced, and a "reconstruction" side, where the original data is recovered from the reduced data. The parameters of the model are constructed by minimizing reconstruction error, or the distance

from the reconstructed data to the original data. This process allows the autoencoder to find the most accurate representation of the original data.

1.2 Wavelet Transforms

Another class of data processing techniques used to train predictive models on financial time series are those that involve **wavelet transform** methods. Wavelet Transform is a process similar to Short Term Fourier Transform, but capable of analyzing different signals at different frequencies. By using a Wavelet Transform, one can then use all time-frequency data, rather than just time data, to inform a predictive model trained on financial time series data.

Of the current literature using wavelet transforms in financial predictive modeling, the overwhelming majority favor Discrete Wavelet Transforms. Signals from the wavelet transform are filtered by strength (with different strengths for each level of the wavelet transform), and the original signal can then be reconstructed from the filtered signals. In this way, DWT is often used to denoise or smooth out time series to strengthen predictive models.

DWT (as well as MODWT, whose difference will be explained later) can also be used for multiresolution analysis, a process by which a signal time series can be split into a smooth (or approximation) series, as well as a number of detail series used to approximate noise and divergences from the smooth series. This decomposition process makes DWT MRA and MODWT MRA very useful for filtering out signals, and can therefore also contribute to strengthening a predictive model when used in preprocessing.

2 Method Descriptions

2.1 Autoencoder + LSTM

The first and simplest of the methods tested, this method combined a one-layer autoencoder with a Long-Short Term Memory network in order to generate one-day-ahead stock price predictions on a test dataset. The LSTM model (whose architecture can be found on this Python notebook) was used to generate one-day-ahead predictions using both the unprocessed stock data as well as the autoencoder output, to determine the specific impact of the autoencoder. Replicating the results of this Github repository confirmed that

using the Autoencoder had a positive impact on reducing test loss.

2.2 Stacked Autoencoder + Wavelet + LSTM

A stacked autoencoder is a type of autoencoder that uses stacked autoencoders to encode and decode data. It consists of multiple single-layer autoencoders in which the output feature of every layer is fed into the input of the next layer, thus "stacking" the autoencoders on top of each other. SAEs are trained by training each individual AE to minimize the error between output and input data. Similarly to autoencoders, they are usually used for unsupervised feature extraction for other types of predictive models, rather than being used on their own.

This method used the wavelet transform simply as a denoising tool before using LSTM to generate predictions with the denoised time series. It used a variety of technical indicators that were generated from daily finance open, close, high, and low price time series. These indicators were fed into a five-layer stacked autoencoder, and the autoencoder-generated representation was then fed into an LSTM to predict one-day-ahead close prices. The specific LSTM architecture used for the LSTM can be found in this Jupyter notebook. A previous paper utilizing this model showed great promise in stock predictions; however, they did have issues with data leakage in the way that they preprocessed their data. To remedy this, the wavelet transform was applied separately to training and test datasets.

2.3 Wavelet + ARIMA

A recent study found that using a Wavelet-Based Multiresolution Analysis in conjunction with an ARIMA model could result in a very strong predictive model. While they specifically applied this to predict the prices of metals, replicating this model on stock price time series also yielded positive results. Multiresolution analysis was performed by using a maximal-overlap discrete wavelet transform (a type of discrete wavelet transform that downsamples to utilize some values removed from the series when performing a normal DWT). This split the time series (which consisted of daily close price data) into a smooth series and a number of detail series (we found that six detail series yielded the optimal results).

Future data was predicted by training separate ARIMA models on the smooth series and each of the detail series, then adding up the predicted

values for each of the series, thus obtaining a forecast of the original time series. I am currently working on implementing a similar algorithm for stock market prices. The paper found that, generally, MODWT is superior for this application as compared to DWT, and was able to obtain high accuracies with sufficient layers and a MODWT-Arima model.

2.4 Wavelet + ANFIS

An ANFIS (Adaptive Neuro-Fuzzy Inference System) is a type of artificial neural network that integrates ANN and fuzzy logic principles, and has been viewed as a "best-of-both-worlds" approach that combines the advantages of both types of models. ANFIS models have been claimed to run more quickly, better adapt to changes in model distributions, and capture more nonlinear structures of a time series relative to neural networks.

Fuzzy Inference Systems generate inferences through a series of fuzzy if-then statements. ANFIS systems use backpropagation to tune the parameters of a FIS using an architecture consists of five layers - a fuzzification layer that converts input values into fuzzy truth values for a given membership function, a rule layer that determines the firing strength of individual rules, a normalization layer to normalize firing strengths into the range [0, 1], a defuzzifier layer that takes normalized values as input as returns defuzzified "consequences" of each rule, and an output layer that combines the consequences to get the predicted output.

The authors of the first paper (Kumar Chandar 2019) to apply Wavelet+ANFIS to financial time series forecasts used wavelet decomposition to decompose time series corresponding to daily volume, open price, high price, and low price into one smooth and two detail series each. These twelve time series were then used as input variables to predict closing prices for the Wavelet+ANFIS model. Using this "WANFIS" model, they were able to outperform ANN and Wavelet+ANN models, as measured by RMSE, on a variety of stock tickers. We were able to implement this method in R, while using Python for data processing purposes. Replicating these methods in MATLAB were no longer feasible due to the depreciation of the Genfis1 method, as well as memory constraints; however, we were able to replicate Kumar Chandar's model and results.

2.5 Wavelet + HFCM

An HFCM is a type of fuzzy-modeling approach to predict time series. A FCM (Fuzzy Cognitive Map) can be represented as a directed graph where nodes represent concepts relevant to a given domain and edges represent causal relations between nodes; each edge has a weight value between -1 to 1 that reflects the nature of this causal relationship. A tanh of a weighted sum of the connections leading into a node is used to update the nodes at each step. However, in FCMs, the state value of each node at iteration T+1 only depends on all connected nodes at iteration T; as a result, they cannot effectively model long-term dependencies. To deal with this, HFCMs (higher-order FCMs) introduce a "memory parameter" that allows preceding values to have a greater impact on node values. The node networks can be created manually before being updated with ridge regression.

The approach used for a Wavelet+HFCM model has two steps. First, the time series is normalized into the range [-1, 1]. The time series is then converted into a multivariate time series using a Haar wavelet transform (the level of the transform is determined experimentally by choosing the one that yields the lowest RMSE). Ridge regression is used to optimize the weight matrix for the HFCM by decomposing the HFCM into connections between individual nodes, which can be optimized separately and therefore more quickly. The optimized HFCM then predicts future values by summing up the values of all nodes in the HFCM while still in the "wavelet space" before reconstructing the original time series by reversing the wavelet transform and normalization.

3 Implementation

3.1 Data Sourcing

Three datasets were used to evaluate models. Both were sourced from Yahoo Finance daily resolution stock data. The first dataset consisted of approximately twenty years' (1-1-2000 to 7-1-2020) worth of Apple daily volume and open, close, high, and low price data. From this, the Python technical analysis library was used to generate a second dataset, consisting of daily resolution data for a number of technical indicators, including BBM, BBH, BBL, ATR, MACD, CCI, EMA, ROC, SMA12, SMA5, MTM6, and MTM12. This yielded 5030 total observations, each consisting of seventeen features,

and were used for the autoencoder and stacked autoencoder + wavelet LSTM models.

The third dataset consisted of open, close, high, low, and volume values for the S&P 500 index over the past five years. This was used to complement the AAPL stock data, as it was both shorter-term and represented a more stable stock range compared to Apple. All three datasets can be located in the data-processing folder of the attached Github repository.

3.2 Autoencoder+LSTM

The autoencoder consisted of six l2-regularized dense layers using ReLU activation, three encoding and three decoding. This differed from a normal stacked autoencoder in the sense that the autoencoding layers formed one large encoding and one large decoding layer, rather than alternating encoding and decoding layers. The seventeen features were compressed into a representation that contained eight features.

The LSTM had three layers containing 60, 120, and 240 neurons, respectively, interspersed with Dropout layers of value 0.2, all with ReLU activation. A 70-15-15 training-validation-testing split was used to split the data, which was normalized into the range (0,1) with a sklearn MinMaxScaler. The LSTM was then tested for a maximum of 1000 layers, with an early stopping mechanism based on validation loss, before being tested on the test data.

One potential area for improvement would be smoothing the predicted time series, as this model often seemed to overreact to changes in the actual closing price. The validation loss stopper was also deemed necessary, as the model had a tendency to drastically overfit the training data, resulting in a loss of test data accuracy (especially when the training and test data occupied different ranges, which they often did for Apple due to the large time period used).

3.3 Stacked Autoencoder + Wavelet + LSTM

First, a discrete wavelet transform with the Haar wavelet (implemented using the PyWt python library) was used to smooth the open, close, high, and low price data. Then, a stacked autoencoder consisting of five encoding and five decoding layers using ReLU activation was used for feature selection. The seventeen features were compressed into a representation that contained eight features.

The LSTM had three layers containing 60, 120, and 240 neurons, respectively, interspersed with Dropout layers of value 0.2, all with ReLU activation. A 70-15-15 training-validation-testing split was used to split the data, which was normalized into the range (0,1) with a sklearn MinMaxScaler. The LSTM was then tested for a maximum of 5000 layers, with an early stopping mechanism based on validation loss, before being tested on the test data.

The validation loss stopper was also deemed necessary, as the model had a tendency to drastically overfit the training data, resulting in a loss of test data accuracy (especially when the training and test data occupied different ranges, which they often did for Apple due to the large time period used).

3.4 Wavelet + ARIMA

In order to implement the maximal overlap DWT-based multiresolution analysis described in the Wavelet + ARIMA paper, custom Python code was implemented to perform MODWTMRA given a number of layers and any wavelet type in the PyWt library. This was used to decompose a daily-resolution time series of close prices into an approximation and six detail series.

The PMDArima Python library's AutoARIMA function was then used to implement the ARIMA models used to predict each of the decomposed time series. The AutoARIMA library selects p and q parameters for an ARIMA model by calculating the AIC (Akaike information criterion) for each set of parameters and selecting the combination that yields the lowest AIC. AIC is calculated to be -2(log-likelihood) + 2K, meaning that better-fitting, less complex models are favored by AutoARIMA. The d parameter must be selected manually; this was done by running the KPSS and ADF stationarity tests to figure out how many times the data needed to be differenced before becoming stationary and selecting the larger of the two outputs from the tests. ARIMA parameters were periodically recalculated with AutoARIMA in order to maintain an updated model, before carrying on as before.

One flaw of these models is how computationally expensive AutoARIMA can be, even when using an AWS server. Combined with having to update the model daily with new data, this model may not be able to deal with minute resolution data or efficiently generate longer-term predictions.

3.5 Wavelet + ANFIS

Due to memory constraints with MATLAB (which the original paper worked in) and a lack of existing Python libraries for fuzzy logic-based methods such as FIS, the FRBS.learn R package was used to develop the WANFIS method. PyWt was used to perform a Discrete Wavelet Transform on the original data, breaking it into its component time series, and the resulting dataframe consisting of the time series values each day for each level and each original feature was saved as a csv. R was then used to normalize the values, perform an 80-20 train-test split, and build the ANFIS predictive model. 3 ANFIS labels, a learning rate of 0.01, and a product membership function were used to replicate the paper's methods.

3.6 Wavelet + HFCM

As we were able to source the original authors' code used for Wavelet + HFCM, we made very few implementation changes other than changing the dataset and experimenting with different validation techniques.

4 Results

Accuracy and time metrics for each of the methods on the Apple 20-year data, as well as accuracy metrics on the S&P 500, are summarized below:

METHOD	AAPL MSE	AAPL RMSE	AAPL time (min)	SPY RMSE
AE+LSTM	0.00014	0.038	120	0.014
SAE+Wavelet+LSTM	1.4e-5	0.004	39	0.017
Wavelet+ARIMA	0.0004	0.019	244	0.058
Wavelet+ANFIS	0.0007	0.0805	111	0.12
Wavelet+HFCM	0.0009	0.097	61	0.118