Deep Learning for Time Series Forecasting:
A Comparison with Traditional Models

Paul Laliberté
Georgia Institute of Technology
paullaliberte@gatech.edu

Daehyun Kim
Georgia Institute of Technology
daehyun.kim@gatech.edu

Abstract

We compare the traditional time series forecasting models, autoregressive integrated moving average (ARIMA) and generalized autoregressive conditional heteroskedasticity (GARCH), against several common recurrent neural network (RNN) architectures, long short-term memory (LSTM) and gated recurrent units (GRU), and a convolutional neural network (CNN). Along with this comparison, we also create GARCH-RNN hybrid models. We find through empirical results that our RNNs, GARCH-RNNs, and CNN perform better than the traditional methods, and provide a theoretical basis for why they do. We test these various models on historical realized volatility calculated from five S&P 500 firms dating from 2015 through 2019.

1. Introduction

1.1. Volatility in Financial Markets

Volatility is an important concept in financial markets. Generally, it measures the variance of stock prices or returns over a period of time. Along with being useful in trading strategies, volatility is also important in calibrating risk models, portfolio management (including retirement accounts like 401Ks), and options pricing [16]. There is not a formal definition of volatility, but some common definitions include variance of log-returns, standard deviation of log-returns, squared log-returns, and many other more complex definitions like that of drift-independent volatility [15].

The basis of volatility is tied to a concept called *log-returns*. Let $p_{d,t}$ be the price of a stock on day d at time $t \in [9.5, 16)$ down to the millisecond. Then we define a log-return on day d at time t as

$$r_{d,t} = \log \frac{p_{d,t}}{p_{d,t-1}} = \log p_{d,t} - \log p_{d,t-1}. \tag{1}$$

We typically call these types of log-returns, *intraday log-returns*, since they capture the percent increasing or decreasing in a stock while the market is open from 9:30am to 4pm. Figure 1 shows intraday log-returns of Apple Incorporated (AAPL) from 2015 through 2018.

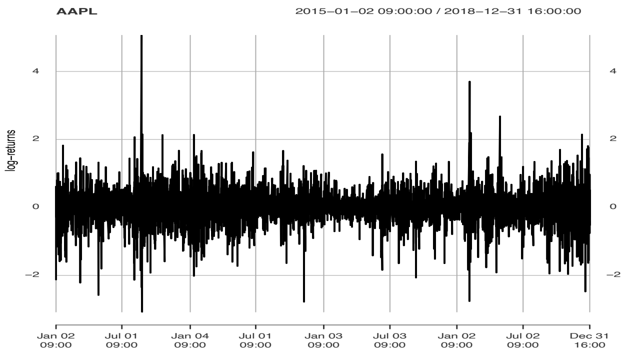

Figure 1. Log-returns of AAPL

To construct historical volatility we constrain our time window to 1-hour volatility measurements, and follow the steps outlined in Dacorogna et al. [5]. That is, let $p_{d,t}$ and $r_{d,t}$ defined as in (1). Then we find that the hourly intraday log-returns r_d^h for $h \in \{9, 10, \ldots, 15\}$ to be

$$\begin{aligned} r_d^h &= \log\left(\frac{p_{d,t}}{p_{d,t-1}}\right) + \cdots + \log\left(\frac{p_{d,t+n}}{p_{d,t+n-1}}\right) \\ &= r_{d,t} + \cdots + r_{d,t+n} \\ &= \sum_{t=1}^{t+n} r_{d,t}, \quad n \in \mathbb{N}. \end{aligned} \tag{2}$$

We calculate a version of volatility called *realized volatility*. This is a slight modification to the above in that we square each $r_{d,t}$ before summing, and then take the square root of the sum. Formally, realized volatility is defined as

$$\sigma_d^h = \sqrt{\sum_{t=1}^{t+n} r_{d,t}^2}. \tag{3}$$

1.2. Stationarity Assumption

1.2.1 Stationarity and Traditional Models

Traditional time series models require an assumption called *stationarity*. A stationary time series is a stochastic process $\{X_t\}_{t\in\mathbb{Z}}$ whose properties do not depend on time. Formally, let the autocovariance function of $\{X_t\}_{t\in\mathbb{Z}}$ be defined as

$$\gamma_X(r,s) = \mathbb{E}\left[(X_r - \mathbb{E}X_r)(X_s - \mathbb{E}X_s)\right]. \tag{4}$$

Then $\{X_t\}$ is said to be stationary if

1. $\mathbb{E}X_t = c$, for all $t \in \mathbb{Z}$, and where c is a constant.
2. $\mathbb{E}X_t^2 < \infty$, for all $t \in \mathbb{Z}$.
3. $\gamma_X(r,s) = \gamma_X(r+t, s+t)$, for all $r, s, t \in \mathbb{Z}$.

Table 1 consists of results from a stationarity test, called a Ljung-Box test. Here, the null hypothesis is that the hourly log-returns are uncorrelated. Depending on the significance level chosen, almost all of our data passes the stationarity assumption.

Box-Ljung Tests					
Company	AAPL	IBM	JNJ	VZ	XOM
P-value	0.015	0.29	0.55	0.001	0.92

Table 1. Box-Ljung Tests for Stationarity

1.2.2 Stationarity and Deep Learning Models

An immediate advantage of deep learning models is that there is no requirement that the time series be stationary. While it may indeed be helpful to transform a non-stationary time series into a stationary one before training and forecasting with deep learning models, there is not a strict requirement for doing so. There is also no guarantee that a non-stationary time series can be transformed to a stationary one. If a traditional model is fit to a non-stationary time series, then inference and forecasting may suffer. In this sense, deep learning models are more flexible on the types of times series data they can be used on.

1.2.3 Other Limitations

There are several other limitations to traditional time series models that deep learning models do not necessarily suffer from. Traditional methods are typically used in short-term forecasting problems. They lack the ability to uncover long-term patterns, and suffer in long-term forecasting problems [10]. RNNs were partially invented with the goal in mind of learning complex long-term patterns [9].

Traditional models also commonly suffer from convergence issues, which in turn increases the amount of time to fit these models, and lowers the forecasting accuracy. These convergence issues are tied to solving a particular optimization problem that we will not discuss in detail here. When using traditional models the common procedure is to predict one time step ahead, and then refit the entire model for the next time step. For example, if one needs to forecast 2000 time steps, we need to refit the model 1999 times. Thus, if convergence issues happen one will need to apply different optimization solvers, or data transformations, to alleviate these issues. For our data set, we experienced convergence issues on 2% of testing observations, and needed to use interpolations after the forecasts to fill these values in. The deep learning models experienced no such issues, and we did not need to perform any post-processing of the forecasts.

A surprising strength of the deep learning models is that, in the case of our problem, they can be quicker and more efficient to run. A CNN with minimal hyperparameter optimization was able to train and forecast twelve-times as fast on a single cpu than on the traditional models using twenty cpus for parallel forecasting. The CNN achieved a better out-of-sample mean-squared error (MSE) and mean-absolute error (MAE) than the traditional models.

1.3. Stock Market Data

Our data set consists of trades from the New York Stock Exchange, and was accessed through Wharton Research Data Services (WRDS) [1]. We chose five stocks, which did not suffer from liquidity issues (no observations for an extended time), and corrected known data issues using the process laid out by Barndorff-Nielsen et al. [2]. These stocks are Apple Inc. (AAPL), International Business Machines Corp. (IBM), Johnson & Johnson (JNJ), Verizon Communications Inc. (VZ), and Exxon Mobil Corp. (XOM). The raw data points are prices of trades down to the millisecond. We then calculated the log-returns and realized volatility on the hourly level described in Section 1.1.

There are several rare events that have occurred over the past years that have effected financial markets in drastic ways. These include the financial crisis of 2008, and the novel coronavirus from the end of 2019 to present. We decided to exclude these events from our data since our goal was to compare traditional time series models with those of deep learning ones in as clear a manner as possible. Our training set consisted of years 2015 through 2018, and we tested on year 2019.

Lastly, and most importantly, we chose this specific data set because the traditional time series models we used were invented to handle underlying issues that are inherent to these types of financial data sets [6, 3].

2. Architectures and Approaches

2.1. ARIMA and GARCH

We briefly introduce the traditional models used. These models are called the autoregressive integrated moving average (ARIMA) model and the generalized autoregressive conditional heteroskedasticity (GARCH) model. These are commonly joined together into an ARIMA+GARCH model. The I in ARIMA stands for integrated, and is the order for which to difference the data. We did not need to difference the data, and set this term to zero. Each model consists of an autoregressive (AR) part with order p and a moving average (MA) part with order q.

2.1.1 Formal Definitions

An ARIMA$(p, 0, q)$ (also commonly referred to as, ARMA(p, q)) model is defined as

$$\begin{aligned} X_t - &\phi_1 X_{t-1} - \cdots - \phi_p X_{t-p} \\ &= Z_t + \theta_1 Z_{t-1} + \cdots + \theta_q Z_{t-q} \end{aligned} \tag{5}$$

where $\{X_t\}$ is stationary, and $Z_t \sim \mathcal{N}(0, \sigma^2)$. The left-hand side of (5) is considered the AR part and the right-hand side the MA part. Note that the ARIMA model assumes constant variance (homoskedasticity), which can be an issue. This is where the GARCH model comes in.

We define a GARCH(p, q) model as

$$\begin{aligned} Y_t = &\mu + Z_t, \qquad Z_t | F_{t-1} \sim \mathcal{N}(0, \sigma_t^2) \\ \sigma_t^2 &= \gamma_0 + \gamma_1 Z_{t-1}^2 + \cdots + \gamma_p Z_{t-p}^2 \\ &+ \beta_1 \sigma_{t-1}^2 + \cdots + \beta_q \sigma_{t-q}^2, \end{aligned} \tag{6}$$

where $F_{t-1} = \{Z_{t-1}, Z_{t-2}, \dots\}$. Note that here we capture the conditional variance σ_t^2 dependent on time t (heteroskedasticity). In this sense we use the ARIMA model to fit the mean (log-returns) and GARCH to fit the variance (realized volatility). For example, an ARIMA$(1, 0, 1)$+GARCH$(1, 1)$ may be defined as

$$\begin{aligned} X_t &= \mu + \phi_1 X_{t-1} + Z_t + \theta_1 Z_{t-1} \\ Z_t &| F_{t-1} \sim \mathcal{N}(0, \sigma_t^2) \\ \sigma_t^2 &= \gamma_0 + \gamma_1 Z_{t-1}^2 + \beta_1 \sigma_{t-1}^2. \end{aligned} \tag{7}$$

For a deeper analysis of these models we refer you to [10, 3].

2.1.2 Application

The ARIMA+GARCH models are fit iteratively for each new time step. The optimal orders can be found in Table 3. For more info on finding these orders we refer you to Mills's text on applied forecasting methods [12].

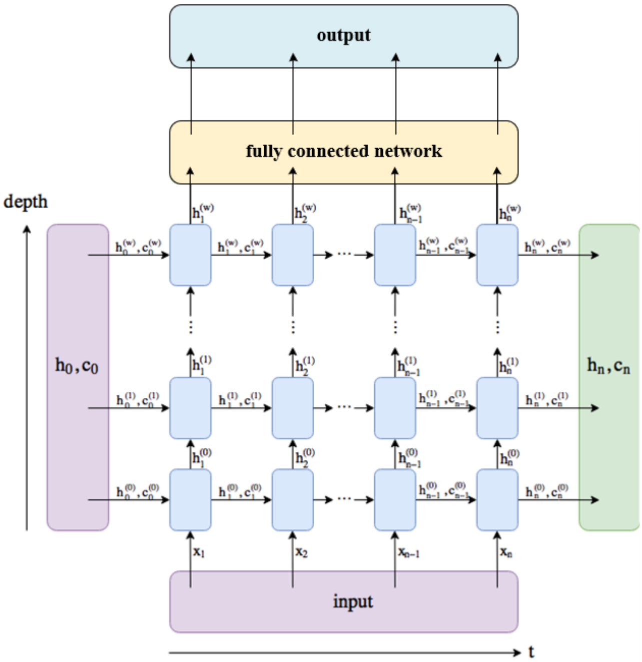

Figure 2. A general RNN architecture.

2.2. RNN Architectures

We implement both LSTM and GRU models. Fig. 2 indicates the general RNN architecture used in our experiments. The RNN architecture includes the RNN specifc cells and a fully-connected layer. The fully-connected layer input dimension is the same as hidden layer dimension, and has an output dimension of 1, which is the hourly volatility forecast. We treat the number of layers and the dimensions of the hidden layers as hyperparameters that we can optimize for. The loss function for all RNNs was a MSE-loss. We discuss our hyperparameter optimization in Section 3.3.

2.2.1 The LSTM Architecture

The LSTM architecture that we implemented includes a forget gate [7]. We define a layer as

$$\begin{aligned} i_t &= \sigma\left(W_{ii} x_t + b_{ii} + W_{hi} h_{t-1} + b_{hi}\right) \\ f_t &= \sigma\left(W_{if} x_t + b_{if} + W_{hf} h_{t-1} + b_{hf}\right) \\ g_t &= \tanh\left(W_{ig} x_t + b_{ig} + W_{hg} h_{t-1} + b_{hg}\right) \\ o_t &= \sigma\left(W_{io} x_t + b_{io} + W_{ho} h_{t-1} + b_{ho}\right) \\ c_t &= f_t \odot c_{t-1} + i_t \odot g_t \\ h_t &= o_t \odot \tanh(c_t), \end{aligned}$$

where i_t, f_t, g_t, and o_t are called the input, forget, cell, and output gates. The cell and hidden states are c_t and h_t, and σ is the sigmoid function. Note that we used multiple layers. Meaning, the input of the l-th layer can be represented as x_t^l, and is the output from the hidden state h_t^{l-1}

of the previous layer. A dropout layer is also applied after each hidden state transition.

2.2.2 The GRU Architecture

We also implemented a GRU model due to them exhibiting stronger performance than LSTM models on certain data sets [4, 8]. We define a layer as

$$\begin{aligned} r_t &= \sigma\left(W_{ir}x_t + b_{ir} + W_{hr}h_{t-1} + b_{hr}\right) \\ z_t &= \sigma\left(W_{iz}x_t + b_{iz} + W_{hz}h_{t-1} + b_{hz}\right) \\ n_t &= \tanh\left(W_{in}x_t + b_{in} + r_t * \left(W_{hn}h_{t-1} + b_{hn}\right)\right) \\ h_t &= (1 - z_t) * n_t + z_t * h_{t-1}, \end{aligned}$$

where r_t, z_t, and n_t are called the reset, update, and new gates. The hidden state is h_t, and the same remark about multiple layers and dropout in the LSTM applies here as well.

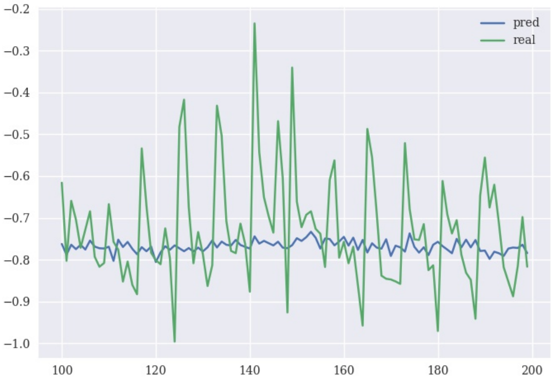

Figure 3. True and LSTM predicted volatility of AAPL. The x-axis is the time and the y-axis is the volatility rescaled as -1 to 1.

2.3. GARCH-RNN Hybrid

For this model we proposed combining the ARIMA+GARCH and RNN variants. We call these the GARCH-RNN models. We first fit several ARIMA+GARCH models for each company, which are similar to what is presented in (7), but with optimal orders p and q. Then we are able to model the realized volatility of the training set given our ARIMA+GARCH models, and pass this as additional input into the RNNs. The motivation behind this is that the ARIMA+GARCH are known for their strong performance on short-term forecasts [10, 14]. In Figure 3, we see that the LSTM model predicted close to the mean, and adding this additional information may help capture the sporadic spikes in volatility we see here. We see a similar pattern with the GRU.

Hyperparameter	Candidates			
Dimension of Hidden Layer	32	64	128	256
Number of Layers	2	3	4	5
Dropout Rate	0.01	0.05	0.1	0.2

Table 2. Hyperparameter search for VZ. Red values are the hyperparameters chosen for the final RNN.

2.4. CNN Architecture

We fit a one-dimensional convolution layer with two fully-connected layers, and a dropout layer in between the fully-connected ones. Since we are dealing with a univariate time-series, the kernel size is one by default. With a batch size of two, where each element in the batch consisted of 3-hour lags of realized volatility, this model achieved very quick training and forecasting in comparison to the RNNs and ARIMA+GARCH models. The loss function was a MSE-loss.

3. Experiments and Results

3.1. Data Preprocessing and Hardware Specifics

We included some minor preprocessing that is specific to the deep learning models. We find the minimum and maximum volatility value for each time series and rescale the observations to fall within an interval of $[-1, 1]$. We primarily used a Geforce RTX 3070 to train the RNNs. A single cpu was sufficient for the CNN.

3.2. Results from RNNs and GARCH-RNNs

Along with the historical volatility, we also included the volume of a stock within an hour. By volume, we mean the number of total shares traded within an hour. The total number of training epochs was set to 1200. From Table 3, we see that the RNNs or the GARCH-RNNs performed the best in terms of mean-squared error (MSE), and the best in terms of mean-absolute error (MAE) except in the case of XOM, where the CNN performed the best. It is worth mentioning that the the results from the RNNs and GARCH-RNNs were all very similar. Overall, the LSTM and GRU were the best performing models. We anticipated that this would be a likely scenario due to their ability to model long-term relationships in the historical volatility.

A bidirectional LSTM, and Transformer architecture were also considered as additional options. In the case of the bidirectional LSTM, we cannot use the backward path during inference, as we cannot use future observations to forecast present ones. In the case of a Transformer, one needs to make several adjustments to insure that only prior data is used. See our concluding remarks in Section 4 for areas of further research involving Transformers.

Table 3. Forecasting Results

Apple Inc. (AAPL)		
Model	MSE	MAE
ARIMA$(9,8)$ + GARCH$(17,17)$	0.047	0.149
LSTM	0.0108	0.0800
GRU	0.0106	0.0792
GARCH-LSTM	**0.0094**	**0.0729**
GARCH-GRU	0.0102	0.0793
CNN	0.023	0.087

Johnson & Johnson (JNJ)		
Model	MSE	MAE
ARIMA$(8,11)$ + GARCH$(16,16)$	0.035	0.125
LSTM	**0.0014**	**0.0273**
GRU	0.0017	0.0320
GARCH-LSTM	0.0019	0.0348
GARCH-GRU	0.0018	0.0338
CNN	0.017	0.076

International Business Machines Corp. (IBM)		
Model	MSE	MAE
ARIMA$(3,2)$ + GARCH$(11,16)$	0.034	0.120
LSTM	0.0149	0.0955
GRU	**0.0144**	**0.0919**
GARCH-LSTM	0.0185	0.1043
GARCH-GRU	0.0189	0.1059
CNN	0.019	0.085

Verizon Communications Inc. (VZ)		
Model	MSE	MAE
ARIMA$(9,11)$ + GARCH$(8,16)$	0.033	0.117
LSTM	**0.0056**	**0.0553**
GRU	0.0060	0.0588
GARCH-LSTM	0.0057	0.055
GARCH-GRU	0.0058	0.0568
CNN	0.027	0.086

Exxon Mobil Corp. (XOM)		
Model	MSE	MAE
ARIMA$(10,7)$ + GARCH$(8,8)$	0.028	0.114
LSTM	**0.0120**	0.0862
GRU	0.0122	0.0869
GARCH-LSTM	0.0141	0.0914
GARCH-GRU	0.0143	0.0900
CNN	0.015	**0.075**

3.3. RNNs and GARCH-RNNs Hyperparameters

As mentioned in Section 2.3, and as can be see in Figure 3, the volatility forecasts hover around -0.8 to -0.7. To help alleviate this problem, we reduced the dropout rate, and the number of layers to induce some small overfitting. Usually, we are trying to avoid overfitting, but we find that the volatility forecasts seemed to need some level of induced overfitting to avoid the mean forecast issue.

We chose to highlight the hyperparameter optimization results for the Verizon Communication Inc. (VZ) time series, which can be found in Table 2. The hyperparameter results of the other companies can be found in the supplementary material, and it is apparent that each company has a different set of optimal hyperparameters. For choosing the optimal hyperparameters we did an iterative grid-like search, using the MSE as a guide.

3.3.1 Target hyperparameter: Number of layers

Figure 4 shows the MSE results based on varying the number of RNN layers. The optimal number of layers is 3, with a MSE of approximately 0.0062. We see that a 2 layer network is not complex enough, and the 4 and 5 layers networks are too complex, and tend to overfit.

3.3.2 Target hyperparameter: Dimension of hidden layer

In Figure 5, the RNNs perform the best with the hidden layer dimensions set to 64. For the GARCH-RNNs, they perform the best with hidden layer dimensions set to 32.

3.3.3 Target hyperparameter: Dropout rate

Figure 6 indicates that the LSTM model and the GARCH-GRU model perform the best when we have a dropout rate of exactly 0. This is an interesting result, since typically some level of overfitting occurs without any dropout, which ends up leading to a lower accuracy. In the case of the GRU model, it performs the best when we set the dropout rate to 0.05. For the GARCH-LSTM model, it performs the best when we set the dropout rate to 0.01. Overall, there seems to be some level of correlation between the model architectures and the amount of dropout needed to achieve the best performance.

3.4. CNN Results and Hyperparameters

The CNN required only a small amount of hyperparameter optimization to achieve results better than the ARIMA+GARCH, but did not outperform the RNNs. The final input and output dimensions of the convolution layer were 3 and 64, with the fully-connected layers having di-

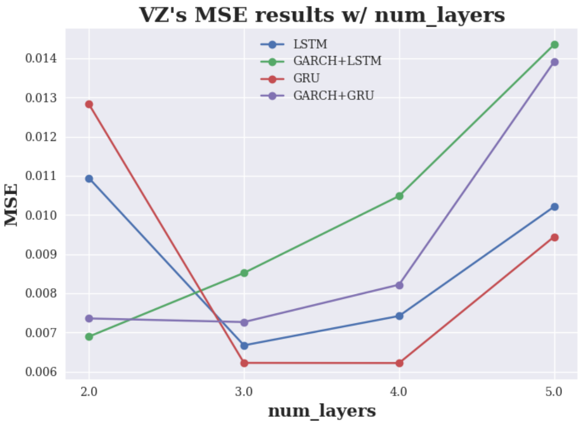

Figure 4. MSE results of VZ based on the number of layers.

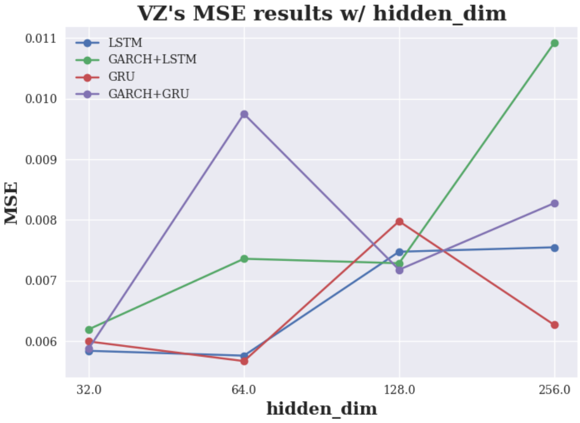

Figure 5. MSE results of VZ based on the dimension of the hidden layer.

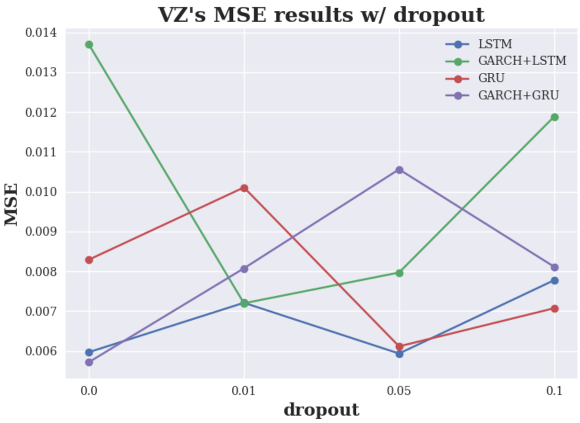

Figure 6. MSE results of VZ based on the number of layers.

mensions 128 and 50 for the first, and 50 and 2 for the second. The dropout rate was set to 0.2

A further benefit of the CNN is that it does not suffer from the exploding (vanishing) gradient issues that a traditional LSTM model may have (we did not experience this issue in the LSTM however) [13]. Overall, the CNN provided evidence that a less complex model that requires minimal hyperparameter tuning, and can be trained relatively fast, can outperform the ARIMA+GARCH models.

4. Conclusion

We did an expansive analysis of traditional time series models (ARIMA+GARCH) against deep learning models (LSTM and GRU). Overall, the RNN models outperform the ARIMA+GARCH approach, and also contain several additional benefits that the ARIMA+GARCH model lacks. These include relaxing the stationarity assumption, avoidance of convergence issues, and faster training times in the case of the CNN. We also implemented a hybird model, which we called the GARCH-RNNs. In theory, we hoped that these variants would help capture the short-term spikes in realized volatility. While this model did outperform the ARIMA+GARCH, it did not always outperform the regular LSTM and GRU. A closer inspection of why it did outperform them in the case of AAPL is a possible area of further research. We also showed that relatively simple deep learning models, like that of a one-dimensional CNN, can also outperform the ARIMA+GARCH models with very little hyperparameter optimization and faster training times.

We have several proposals for areas of further research. Online learning is a possible candidate to be applied to these models, and improve the forecasting results. In this project, we train the architecture with 4 years of historical data and test on 1 year of data. However, incorporating test data into the training data as we make forecasts could be beneficial. This is typically what the ARIMA+GARCH does with the iterative refitting that we mentioned in Section 2.1.2. Therefore, the forecasts could be improved by shifting the training window as we forecast. For Transformers, one could implement a time series based model where we have masked self-attention. Specifically, before a softmax is applied to the output, one could mask the observations that look into the future, and prevent the model from peeking ahead. There is also recent research in time representations and vector embeddings. Kazemi et al. have proposed a vector representation that captures time, called *Time2Vec* [11].

References

[1] Wharton research data services. Accessed: 2021-04-05. 2

[2] O.E Barndorff-Nielson, P. Reinhard, A. Lunde, and N.Shepard. Realized kernels in practice: trades and quotes. *The Econometrics Journal*, 12(3):C1–C32, 2009. 2

[3] Tim Bollerslev. Generalized autoregressive conditional heteroskedasticity. *Journal of Econometrics*, 31(3):307–327, 1986. 2, 3

[4] Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder–decoder for statistical machine translation. *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734, 2014. 4

[5] Michel Dacorogna, Ramazan Gençay, Ulrich Müller, Richard Olsen, and Olivier Pictet. *An Introduction to High-Frequency Finance*. Academic Press, 2001. 1

[6] Robert F. Engle. Autoregressive conditional heteroscedasticity with estimates of the variance of united kingdom inflation. *Econometrica*, 50(4):987–1007, 1982. 2

[7] Felix A. Gers, Jürgen Schmidhuber, and Fred Cummins. Learning to forget: Continual prediction with lstm. *Neural Computation*, 12(10):2451–2471, 2000. 3

[8] Nicole Gruber and Alfred Jockisch. Are gru cells more specific and lstm cells more sensitive in motive classification of text? *Frontiers in Artificial Intelligence*, 3:40, 2020. 4

[9] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997. 2

[10] Rob J. Hyndman and George Athanasopoulos. *Forecasting: principles and practice*. OTexts: Melbourne, Australia, 2 edition, 2018. 2, 3, 4

[11] Seyed Mehran Kazemi, Rishab Goel, Sepehr Eghbali, Janahan Ramanan, Jaspreet Sahota, Sanjay Thakur, Stella Wu, Cathal Smyth, Pascal Poupart, and Marcus Brubaker. Time2vec: Learning a vector representation of time. *CoRR*, abs/1907.05321, 2019. 6

[12] Terence C. Mills. *Applied Time Series Analysis: A Practical Guide to Modeling and Forecasting*. Academic Press, 2019. 3

[13] Roni Mittelman. Time-series modeling with undecimated fully convolutional neural networks, 2015. 6

[14] Ruey S. Tsay. *Analysis of Financial Time Series*. John Wiley & Sons, Inc., 2010. 4

[15] Dennis Yang and Qiang Zhang. Drift-independent volatility estimation based on high, low, open, and close prices. *The Journal of Business*, 73(3):477–91, 2000. 1

[16] Gilles Zumbach. Chapter 26 - relevance of volatility forecasting in financial risk management. *Risk Management*, pages 583–603, 2006. 1