Volatility Forecasting Using Deep Learning

Li Yi (Mentor: Dhruv Kedia)

CS221 Final Project, Computer Science Department, Stanford University

CS221

Background

Forecasting volatility of financial assets is of interest to both researchers and practitioners. An accurate prediction of future volatilities has important implications in many areas such as derivative pricing, portfolio optimization and risk management.

Furthermore, although past studies have shown that **short-term financial asset returns are difficult to forecast**, due to their low signal-to-noise ratio, non-stationarity and strong regime dependency, a large body of finance and econometrics literature suggested that **volatility is much "easier" for forecast**, given well-known features such as volatility clustering, leverage effect, and co-movements with other assets' volatilities.

Hence, with cautious optimism and humble expectation, the goal of this project is to experiment and apply deep learning methods developed to tackle time-series problems to forecast volatility of financial assets.

Data and Task Definition

I've downloaded daily levels of both S&P 500 and VIX from Global Financial Data (GFD) via Stanford Library, covering the time period Jan-1988 to Oct-2019 (N=8012). Using the close-to-close log returns, I then calculated the daily volatility as the sample standard deviation of the index and annualized it using $\sqrt{260}$.

The task is forecast the realized volatility over the next month (assuming 21 trading days). I will be evaluating the forecast using the root mean square error (RMSE) as the default metric, but I will also consider mean absolute percentage error (MAPE) as an alternative measure.

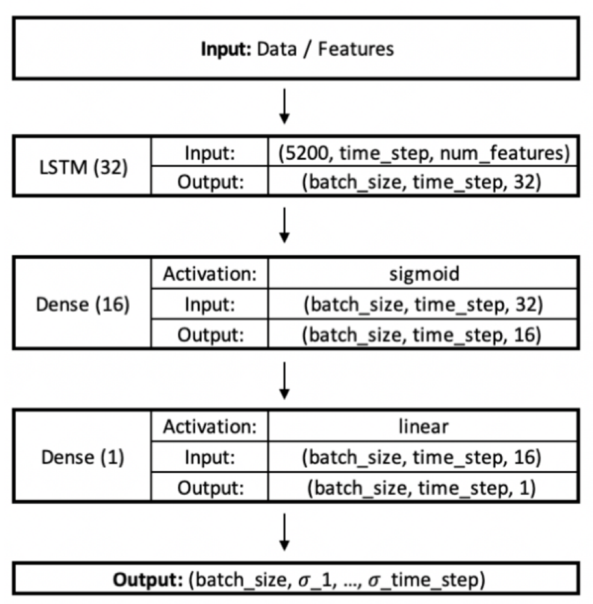
Our objective is to **outperform the common baselines of historical realized volatility (exponentially-smoothed with a half-life of 21-days) and the well-known GARCH(1, 1) volatility model.** Ideally, we should also get close to the oracle of implied volatility (VIX) de-biased using full-period information (i.e. some perfect foresight).

Method: RNN/LSTM Architectures

Compared to traditional feed-forward neural networks, recurrent neural networks (RNNs) have feedback loops which allow information to persist, which worked well in tasks involving dynamic temporal patterns. In particular, I've decided to use Long Short Term Memory (LSTM) [1], a special kind of RNN that has the ability to learn "long-term dependencies" via its four interacting layers/gates. Without going into the details (found in the paper), the additional parameters of these memory blocks are learnt using standard backpropagation.

There are numerous neural network architectures incorporating LSTM. I've implemented two different approaches. The first, inspired by Xiong (2015) [2], is a **single-layer LSTM followed by two fully-connected dense layers**, which uses a `time_step` of 10 and a `batch_size` of 32, trained using `adam` optimizer and MAPE as the objective loss function. The second, modified from Kim (2018) [3], is a **3-layer LSTM with 3 dropout layers**, before passing them through two fully-connected layers at *t* to predict volatility at time *t+1*.

Single Layer LSTM from [17]



Input: Data / Features

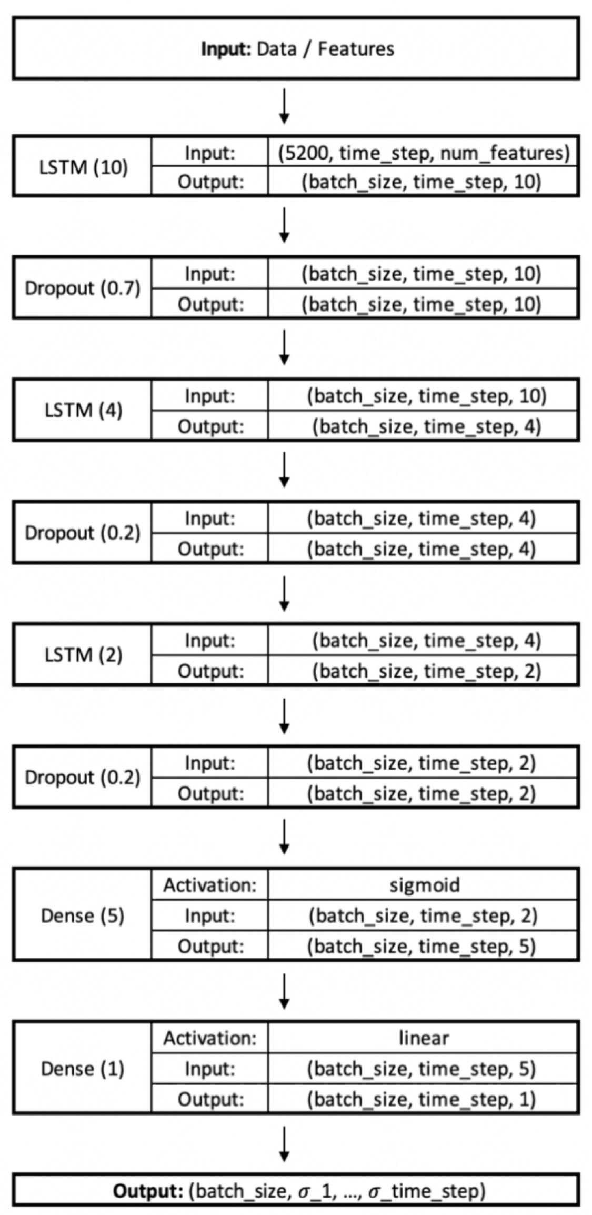
LSTM (32)	Input:	(5200, time_step, num_features)
	Output:	(batch_size, time_step, 32)

Dense (16)	Activation:	sigmoid
	Input:	(batch_size, time_step, 32)
	Output:	(batch_size, time_step, 16)

Dense (1)	Activation:	linear
	Input:	(batch_size, time_step, 16)
	Output:	(batch_size, time_step, 1)

Output: (batch_size, σ_1, …, σ_time_step)

Multiple Layer LSTM from [19]



Input: Data / Features

LSTM (10)	Input:	(5200, time_step, num_features)
	Output:	(batch_size, time_step, 10)

Dropout (0.7)	Input:	(batch_size, time_step, 10)
	Output:	(batch_size, time_step, 10)

LSTM (4)	Input:	(batch_size, time_step, 10)
	Output:	(batch_size, time_step, 4)

Dropout (0.2)	Input:	(batch_size, time_step, 4)
	Output:	(batch_size, time_step, 4)

LSTM (2)	Input:	(batch_size, time_step, 4)
	Output:	(batch_size, time_step, 2)

Dropout (0.2)	Input:	(batch_size, time_step, 2)
	Output:	(batch_size, time_step, 2)

Dense (5)	Activation:	sigmoid
	Input:	(batch_size, time_step, 2)
	Output:	(batch_size, time_step, 5)

Dense (1)	Activation:	linear
	Input:	(batch_size, time_step, 5)
	Output:	(batch_size, time_step, 1)

Output: (batch_size, σ_1, …, σ_time_step)

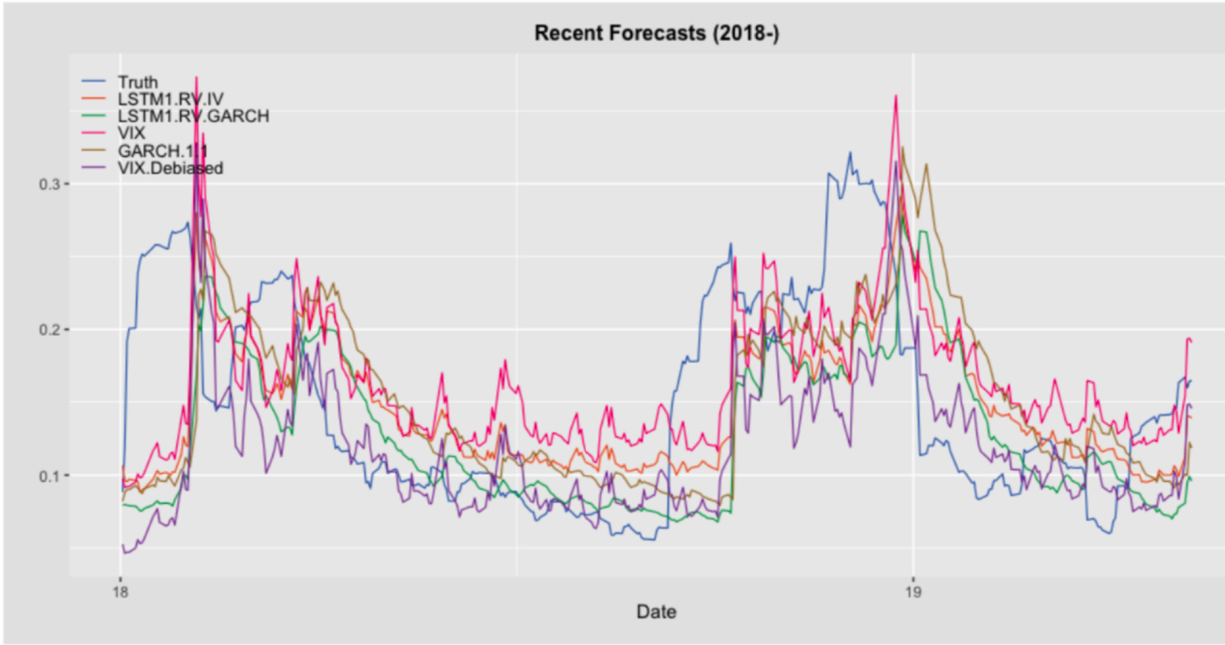
Results and Discussion

I've implemented both architectures above via the python deep learning library Keras and the related R interface. I've also repeated the runs four different sets of input feature listed in the table on the right.

We can see that **LSTM1 using Realized Volatility (RV) and Implied Volatility (IV) have the lowest RMSE** of **5.56%**, beating our baselines and oracles. However, it has a higher MAPE of **35.6%** than our VIX-debiased oracle. Among all 8 variations, LSTM1 using RV and GARCH has the lowest MAPE of **31.4%**, significantly better than both the baseline GARCH(1,1) and the oracle full-period GARCH(1,1).

Another observation is that the **deeper 3-layer LSTM2 did not seem to outperform the simpler LSTM1**, which may suggest that there is either limited non-linearity to be captured or the three layers of drop-out (especially in between LSTM cells) are too punitive.

Model	Features	Params	OOS RMSE	OOS MAPE
Baseline EMSD(21)	-	-	6.78%	37.4%
Baseline GARCH(1,1)	-	-	6.24%	38.7%
Oracle Full GARCH(1,1,)	-	-	6.23%	38.3%
Oracle VIX-Debiased	-	-	5.67%	26.1%
LSTM1	RV	4,897	6.08%	33.1%
LSTM1	RV+IV	5,025	**5.56**%	35.6%
LSTM1	RV+GARCH	5,025	6.07%	**31.4**%
LSTM1	RV+GARCH+IV	5,153	5.66%	38.0%
LSTM2	RV	797	5.94%	35.9%
LSTM2	RV+IV	837	5.66%	36.3%
LSTM2	RV+GARCH	837	6.06%	41.8%
LSTM2	RV+GARCH+IV	877	5.73%	38.7%



Conclusion and Future Works

Overall, while the initial results have been encouraging, I would also note the following caveats:

- While we saw strong relative performance, the absolute magnitude of error (at around **5.56%** RMSE and **31.4%** MAPE) is still very high => **Therefore, this limits the economic significance of any real-world applications using forecasts from our LSTM model.**
- Hidden underneath the summary accuracy measure is the fairly large variations I saw across the different years => **This suggest that our model might not be sufficiently adaptive for a sudden regime changes.**
- The US stock market is one of the most widely traded and "efficient" assets => **As such, we may not achieve similar good performance in other less liquid assets.**

There are several areas for improvements and future works, which I hope to undertake before the final paper:

- Tuning the LSTM architecture, and potentially **re-training the model periodically** (e.g. once every year), which could potentially improve the results and their stability across different years.
- Expand the experiment to other assets (e.g. currencies), and also **include additional features** (e.g. the asset's own volume and related assets' returns)
- Explore an **attention mechanism** in the LSTM model to give higher weights to more important features.

Key References

[1] Sepp Hochreiter and Jürgen Schmidhuber "Long Short-Term Memory". *Neural Computation*, Volume 9(8). MIT Press, 1997.

[2] Ruoxuan Xiong, Eric P. Nichols and Yuan Shen "Deep Learning Stock Volatility with Google Domestic Trends". *arXiv preprint arXiv:1512.04916*, 2015.

[3] Ha Young Kim and Chang Hyun Won. "Forecasting the volatility of stock price index: A hybrid model integrating LSTM with multiple GARCH-type models". *Expert Systems with Applications*, Volume 103. Elsevier, 2018.

Screencast Video Link: https://youtu.be/3iXJ6Sp1rPE

Stanford University