Financial Prediction Using LSTM and Multivariable Factor Regression During the Covid-19 Pandemic

Yann Hoffmann – University of Luxembourg
MICS - Master in Computer Sciences
Specialization: Data Sciences

Abstract

The objective of this paper is to demonstrate the adequacy of LSTM models in predicting stock prices within a rigid framework of stationarity. Stationarity is achieved by restricted our study to price movements during recessions and in particular during the stock market rout and recovery following the covid-19 pandemic. The LSTM model is used conjointly with a multivariable factor analysis on the fundamentals of the companies to manufacture strong trading signals and garner deeper insights on how investors value companies during economic uncertainty. The project achieves promising results through the necessary use of distribution over a cluster. From there, the code can easily be scaled up even further. The LSTM architecture manages to effectively featurize price formation while the multivariable factor analysis offers an alternative way of thinking about future performance of companies when liquidity invariably becomes an issue.

Introduction

The stock market is the pricing mechanism for various instruments, equities and their derivatives. Prices are known to all economic actors and act as a proxy for the health of the economy, the scarcity of a commodity or the expected value of common stocks and derivatives. The economy follows cycles of booms and busts hardly predictable that induces non-stationarity. In 2020 we are again faced with an unprecedented crisis that has uprooted global supply chains, impacted labor movement and suppressed demand. Can we learn from past recessions to anticipate recovery and penalize companies that don't build sufficient buffers in bullish times or is it already priced in the market?

In this paper, I train two models with the objective of measuring which assets are likelier to underperform or beat the market exclusively in times of crisis. The first model is trained on the NYSE prices from 2012 to 2016. It serves as a baseline for price prediction as well as a conceptual improvement on the classical Brownian Motion modelization. In addition to this predictive model, I train a multilinear regression model on various fundamentals from this period. The fundamentals include accounting figures such as the bottom line, cash flows, as well as usual liquidity ratios, and measures of indebtedness. With a 0.3%-accuracy LSTM model and the data from the fundamentals, I leverage the power of Apache Spark and the Iris cluster at the *University of Luxembourg* to (1) preprocess the data, (2) tune the hyperparameters of the architecture and (3) train and validate the models on prices from the covid-19 crisis.

SOTA Review (LSTM)

The standard approach in econometrics for time series analysis and forecasting is to use ARIMA models (Auto Regressive Integrated Moving Average). This effective model detrends any series and

studies its residuals. ARIMA has already been coupled with LSTM models to predict correlations between securities to improve portfolio management [1]. LSTM models have been widely used in NLP and to great success in the context of seq2seq and language translation. Moghar et alii used the LSTM and have proven that it can indeed achieve accurate predictions of stock return and unearth complex multivariable correlation between different features [2].

For all their strengths in capturing states and fostering long term dependencies, asset prices are notoriously hard to predict even with the best technology. LSTM have several weaknesses, one of which is that financial data are non-stationary [3]. To mitigate this problem I decided to focus only on bear markets in the hope that they might be sufficiently stationary. Sirignano and Cont have suggested that an increase in the quality and the quantity of the data (ie. an access to undisclosed data and higher granularity with different data sources) can give the model a real edge on the market – regardless of stationarity [4].

In the next sections, I will explore the idea of using an LSTM and multivariate linear regression, better known as factor modelling, on fundamentals data.

Data sources and processing

As alluded to in the introduction, the choice and the quality of the data is the main determinant of success of a machine learning application. For this project, I have drawn structured data from the Kaggle platform.

1. ANN data set

The first dataset used in the project is the ANN data set. It weighs 2GB and comprises stock histories from AMEX, NYSE and the NASDAQ stock exchanges. The entry dates were continuously reported on a daily basis since 1970. It thus encompasses the covid pandemic period, crucial for our analysis. The csv itself contains the close price (dependent label) as well as the open price, the volume and the intraday low and high mark (independent features).

2. Fundamentals

This second data set contains up to 400 accounting numbers disclosed in the earnings reports that describe the overall financial health of the listed companies. The fundamentals are purposefully taken four years before the crisis (2012 - 2016) so as strengthen the model.

As a first pass through the data, the notebook data_exploration.ipynb is dedicated to understanding the correlation between the data and was used to narrow down on the two presented data sets.

In order to feed the data to the neural network, I divided the ANN data into two periods: Jan 2012 – Dec 2016 to train the algorithm and Jan 2020 – Aug 2020 to test it in conjunction with the regression. I made sure to remove IPOs from after 2012 and companies that might have been delisted too. Since we care about the deviation from the price of the aggregate index, I applied differencing on the stock prices and then scaled the series to facilitate backpropagation, all of this using the PySpark MLlib library. The fundamentals were in turn collected into row vectors until the training phase.

Model explanation and intuition

The purpose of the end model is to predict if any basket of stocks outperforms the market index during an economic shock. Technical recessions follow, by definition, two semesters of GDP contraction and will be used as benchmarks in this project. The model is composed of two parts, the first one is a comprehensive stock price-predicting neural network biased towards large cap businesses. The second component is a multivariable linear regression on the fundamentals of major companies a few years prior to a recession. The combination of both is then optimized to fit the recession stock prices (2020).

The resulting factor analysis lends itself to qualitative interpretation of how these fundamentals affect the performance of a stock. One can surmise that during a liquidity crunch companies with large margin buffer such as Apple outperform the broader market. Blue chip and consumer staples are also known to be recession-proof and the model can perhaps capture this dynamic.

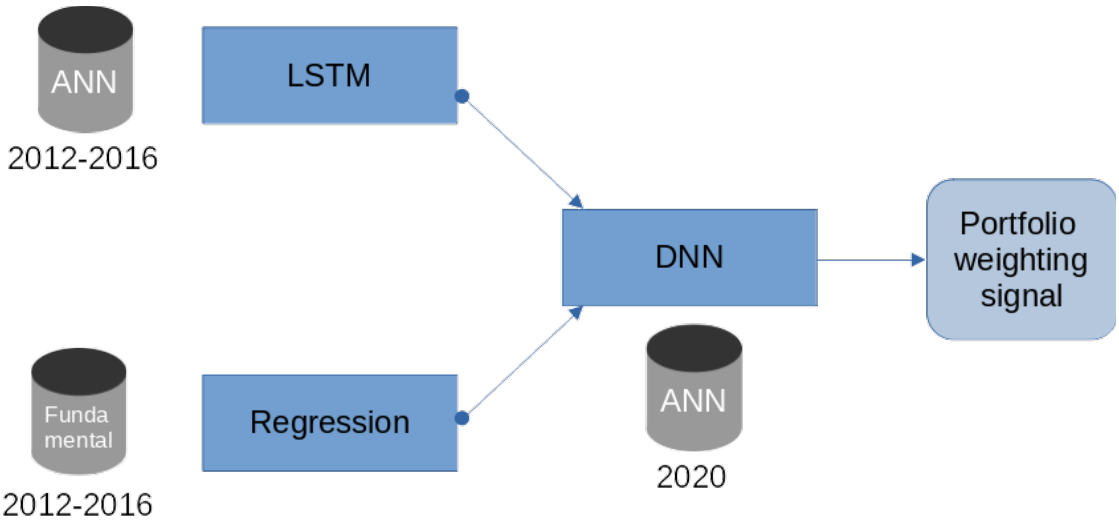
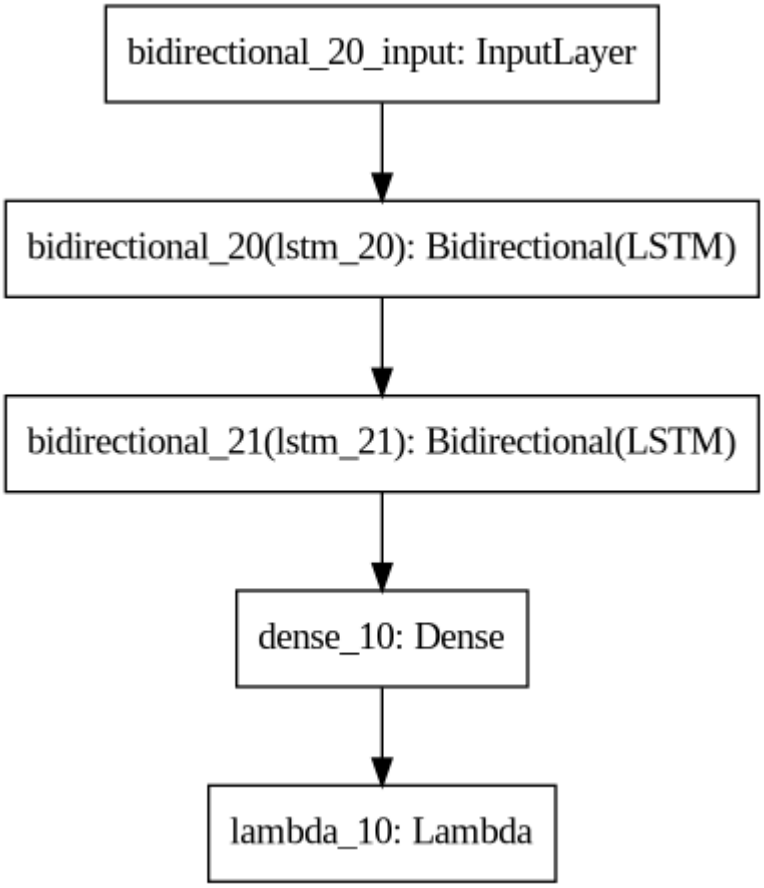


Figure 1: Schema of the final model

LSTM model

After testing a few LSTM architectures for 100 epochs, I found that the below model performs the best. Compared to models with Conv1D layers, and fully-connected layers that follow the recurrent layers, this retained architecture is not very complex. It has nonetheless 2,474,737 trainable parameters and lends itself well to a distributed training.



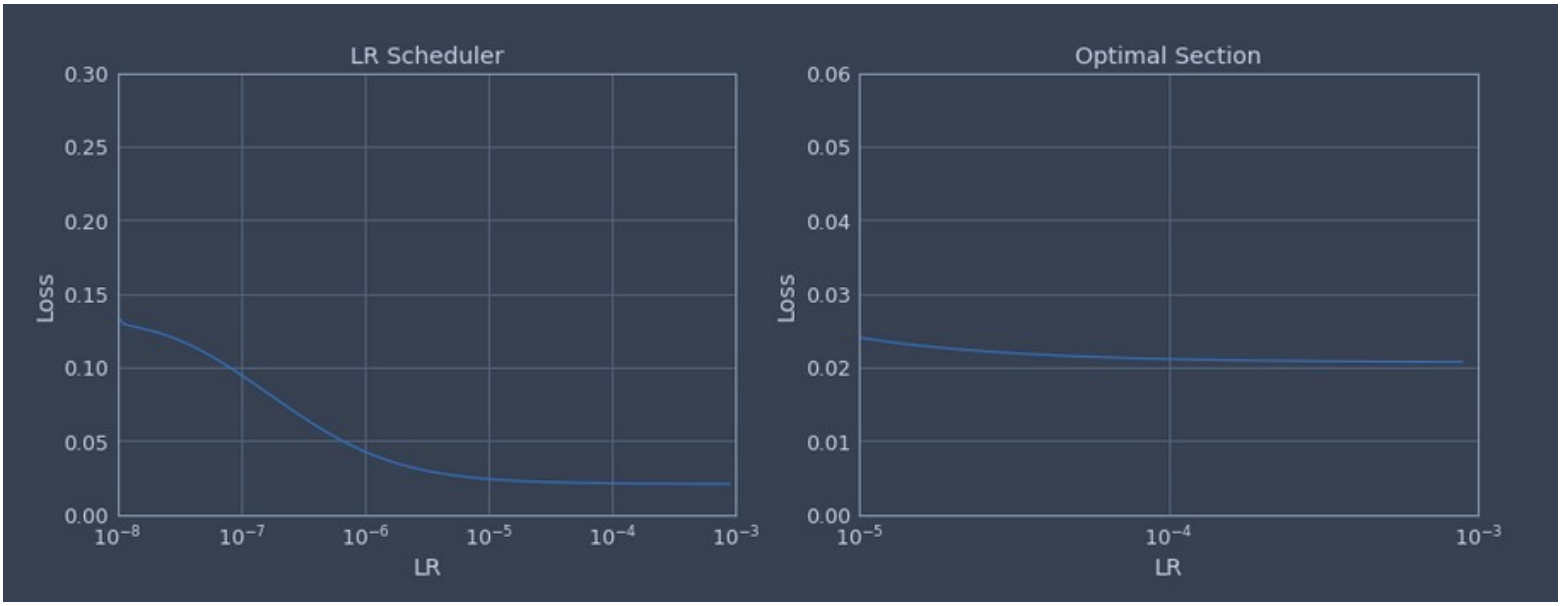
Hyperparameter tuning

I tried two techniques to find the best parameters before training the model on the cluster. The first technique is a conventional trial and error method that gives a quick and dirty approximation of a good starting point [5].

The second more exhaustive approach was using the Optuna library. Optuna is a hyperparameter optimization framework that utilizes grid search and genetic algorithm within user defined bounds to find the best settings. Below are a few results of the exploration of the hyperparameter space.

Note that the results of the optimization can be found in the optuna.csv file.

1. Scheduler / callback method



With the help of a Keras callback function, I input during one training cycle different learning rates for the Stochastic Gradient Descent (SGD) optimizer and plotted the resulting losses. The right chart shows where the curve stabilizes to its minimum loss value. A learning rate between 10 -5 and 10 -3 achieves the best result with 10 -3 being quickest to converge.

2. Optuna optimization

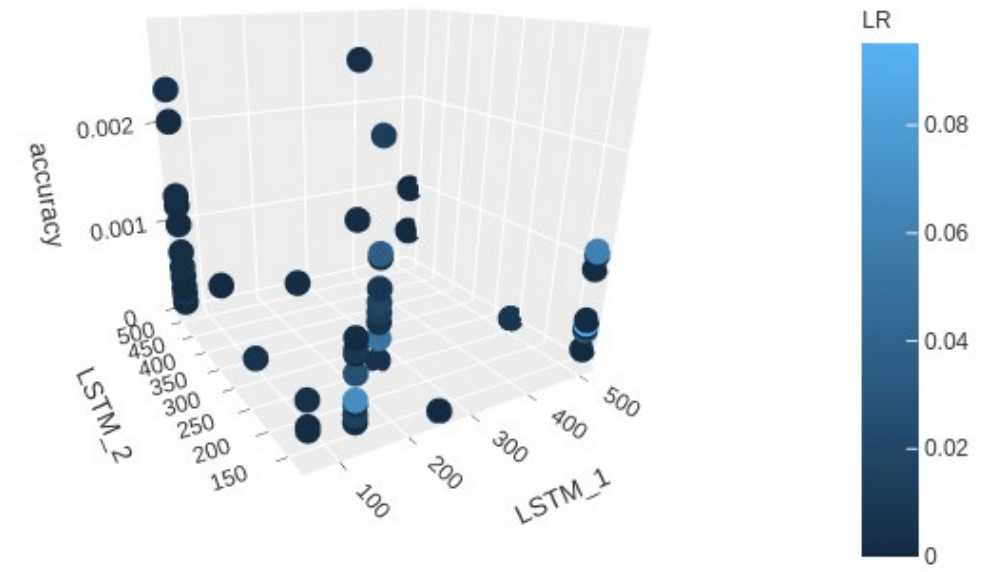
Search space:

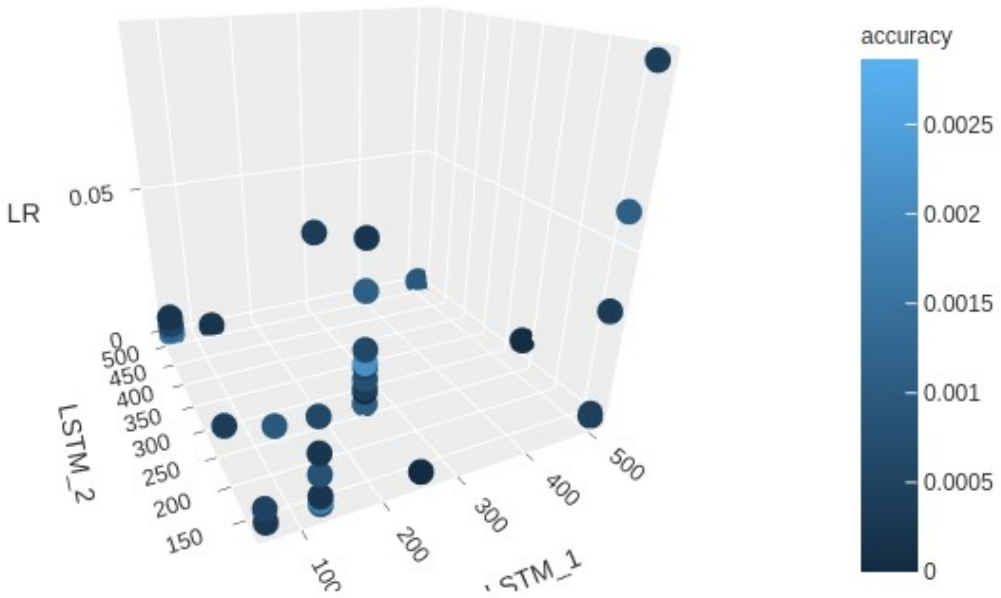
- **LSTM_1** = {64, 128, 256, 512}
- **LSTM_2** = {128, 256, 512}
- **LR** = [10 -6; 10 -3] (log scale)

The next plot shows the accuracy of the generated model on the z axis. We can see that for an identical column of points having the same architecture (L1, L2), the accuracy varies widely depending on the learning rate and the random initialization. Overall, models with less nodes on the first LSTM layer and more nodes on the second perform better than more complex networks. After 50 runs of 150 epochs over 3 hyperparameters the best model ends up being the (128, 128) LSTM model with a learning rate on the upper end of the scale. This configuration is used for training on the cluster.

The second plot shows that there is no clear relation between the learning rate and the accuracy of the model. Accordingly, within one (L1, L2) column, the most accurate model will cluster around a certain learning rate, but this convergence optimum differs for different columns.

On account of the optimization taking 10 hours to approximately determine only 3 hyperparameters, this step is the most conducive to speed-up gains through parallelization. However, Optuna does not support Spark as of yet.


accuracy
0.002
0.001
0
LSTM_2
500
450
400
350
300
250
200
150
100
200
300
400
500
LSTM_1
LR
0.08
0.06
0.04
0.02
0


LR
0.05
0
LSTM_2
500
450
400
350
300
250
200
150
100
200
300
400
500
LSTM_1
accuracy
0.0025
0.002
0.0015
0.001
0.0005
0

Parallelization

1. TensorFlowOnSpark

The rational behind using Spark and TensorFlowOnSpark to train models in a distributed fashion is the following. The use of massive data has been shown to improve the accuracy of financial models. In particular, Sirignano and Cont's data set is a "a high-frequency record of all orders, transactions and order cancellations for approximately 1000 stocks traded on the NASDAQ between January 1, 2014 and March 31, 2017" and deals with "order books of certain stocks [that] can update millions of times per day. This leads to **TeraBytes** of data, which we put to use to build a data-driven model of the price formation process." In short, the scaling up of machine learning training and inference is crucial to obtaining cutting edge results.

With the aim of implementing this strategy albeit at smaller scale, I turned my attention to the yahoo/TensorFlowOnSpark library and its conversion guide. There are many other open-source frameworks like sparkDL and DL4J. Unfortunately the former is in development and works only with CNNs while the latter focuses on entreprise deployment. TensorFlowOnSpark uses a synchronous, data-parallel paradigm that splits batches of data among the worker nodes and returns the gradient at each step.

The library provides a 4-step conversion guide:

1. *Develop and test your TensorFlow application as a single-node application on small scale data*.

The introductory first step, from modelling to hyperparameter tuning, can be found in the lstm_model.ipynb notebook.

2. *Convert your single-node TensorFlow application into a distributed TensorFlow application*.

The second step is facilitated through the use of the high-level Keras API and can be found in the parallelization.ipynb notebook. For this project, I used the experimental *MirroredStrategy* from TensorFlow that uploads directly the data to the worker nodes during training.

3. *Convert your distributed TensorFlow application to TensorFlowOnSpark.*

The third and final step takes place in part in parallelization.ipynb and in the launcher script submit.sh. It takes care of the environment variables and uses the discouraged but faster *Input.SPARK* data transfer method.

4. *Gradually scale up your cluster*.

Step four is left for a real life entreprise application.

I decided to allocate 2 worker nodes along with one "master-worker" node to run the submit.py training script that is nothing more than the parallelization.ipynb code. The results can be found in the next section.

2. Preprocessing

The preprocessing is done through the help of the MLlib Spark library. Throughout the preprocessing I used the DataFrame Spark API because the DataSet is not available on PySpark and because DataFrames are now the recommended entry point that ensure better performances when considering DAG optimizations. Conversion to RDD are sometimes warranted when row wise transformation are under consideration or when transposing a DataFrame although this is not recommended for real-world performance issues.

Results and interpretation

1. Time and accuracy

	Single node	**Distributed 3 nodes**
Preprocessing time (s)	390.2	75.1
Training time (s)	660.5	2160.9
Epochs	100	500
Accuracy	0.00301	0.00317
890 Inferences (s)	26.2	N/A
Optuna (50 * 150 epochs)	10 hours	N/A

The above table shows the difference between running different stages of the machine learning life cycle on a single node computer and on the iris cluster. We observe that the preprocessing stage enjoys a significant speed-up on the cluster of more than 4x. If we extrapolate the 100 epochs training to 500 we would get a duration of 3300s for a single node computer compared to 2160s on a cluster. This represents a 33% increase in speed. This comparetively poor performance is probably due to network bottlenecks and to the fact that the data set is relatively small so that we do not reach the economies of scale that usually come out of distributed training. Finally the inference takes only 26s on a single node computer and the Optuna optimization is by far the lengthiest operation with a duration of 10 hours.

After training the LSTM model on the HPC, we get an accuracy of 0.317% which is barely an improvement on the 100 epochs training. This could be due to the choice of the learning rate that was optimized for a 100 epochs training and not 500. Furthermore, a more complex model would benefit from the additional training time whereas a (128, 128) model has less room for improvement.

2. LSTM forecasting

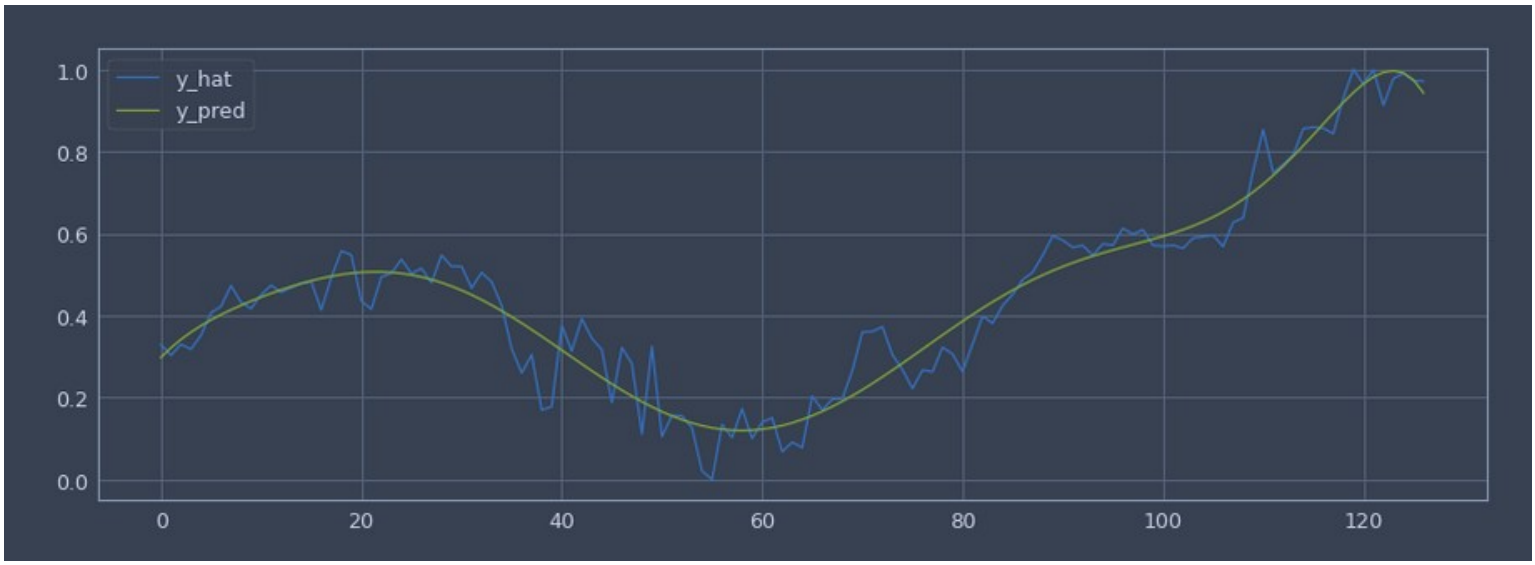
As we can see, the LSTM has a fairly low accuracy. It makes conservative bets on the market tacking onto the average historical price but exhibiting at times swings in the right direction.

3. Polynomial interpolation

The polynomial interpolation on the basis of factor analysis of the fundamentals turned out to be impossible to optimize with Scipy.

At first, I tried to create a multivariable regression model to map the fundamentals to the space of the polynomial parameters in the hopes that there would exist a hyperplane that generates a predictive polynomial on prices within reasonable accuracy.

There is however absolutely no guarantee that such a linear relationship even exists between fundamentals and the latent space. The question how to find a set of n depended parameters that would generate a polynomial approximation from the fundamentals remains open.



Conclusion

In summary, in the course of this project I have explored different financial sources in order to construct a model that generates executable trades during economic stress best illustrated by the ongoing covid-19 pandemic. I have restricted the data under study to recessions to obtain a stationary signal. The model composed of a LSTM predictive baseline and a polynomial mutlifactor approximation of the crash and recovery has proven encouraging. After much iteration on the LSTM model, the architecture achieves a 0.31% accuracy which would translate to a net positive trading strategy. It remains to see if the predictive advantage does not vanish after taking into consideration trading costs and the potential overfitting of the model during backtesting. The polynomial modelization must be further refined and tested against the 2020 prices as initially intended.

Finally, the use of the cluster has conceptually proven to be effective at processing large swathes of data and providing significant speed-ups. The cluster can without a doubt justify the use of a larger datset, with higher frequency ticks along with a more complex neural network architecture. The most promising aspect is that of the parallelization of hyperparameter tuning and validation. This task is straightforward to distribute and could substantially improve the performance of the model by widening the search space.

Additional information and explanations about the polynomial modeling among others can be found in the jupyter notebooks.

References

[1] MOGHAR, Adil; HAMICHE, Mhamed. Stock Market Prediction Using LSTM Recurrent Neural Network. *Procedia Computer Science*, 2020, 170. Jg., S. 1168-1173.

[2] CHOI, Hyeong Kyu. Stock price correlation coefficient prediction with ARIMA-LSTM hybrid model. *arXiv preprint arXiv:1808.01560*, 2018.

[3]: https://medium.com/datadriveninvestor/why-financial-time-series-lstm-prediction-fails-4d1486d336e0

[4] SIRIGNANO, Justin; CONT, Rama. Universal features of price formation in financial markets: perspectives from deep learning. *Quantitative Finance*, 2019, 19. Jg., Nr. 9, S. 1449-1459.

[5] Coursera, Sequences, Time Series and Prediction: https://www.coursera.org/learn/tensorflow-sequences-time-series-and-prediction/

Resources

The (Mis)behavior of markets: A fractal view view of risk, ruin and reward – Benoit B. Mandelbrot

The Black Swan: The impact of the highly improbably – Nassim Nicholas Taleb

"The Machine Learning Approach" by Michael Kearns: https://www.youtube.com/watch?v=8P4MfZimMT4&feature=emb_rel_pause

Quantopian Lecture Series: https://www.quantopian.com/lectures

MIT: Topics in Mathematics with Applications in Finance: https://www.youtube.com/watch?v=wvXDB9dMdEo&list=PLaLOVNqqD-2G5SSErHfvGqs7ev7kE8_fj&index=1

Data

AMEX, NYSE, NASDAQ stock histories: https://www.kaggle.com/qks1lver/amex-nyse-nasdaq-stock-histories

New York Stock Exchange: https://www.kaggle.com/dgawlik/nyse

API and libraries

ffn, pandas, scipy, scikit-learn, tensorflow, tensorflowonspark, pyspark, mllib, seaborn, numpy, matplotlib, keras, graphviz, optuna, plotly

Yann Hoffmann, University of Luxembourg
MICS – Master in Computer Sciences
Specialization data sciences