S&P 500 volume prediction with Wavenet
A technical report for a github project

February 10, 2020

This document covers the first steps of experimenting with S&P 500 stock volume prediction. The aim is to beat the baseline of 'tomorow is gonna be the same' (which seemingly cannot be beaten with a simple models on a first try like documented here), and analyzing the found model. References to other works are used mostly for inspiration (and are either generally not recommended to follow or only talk about failed attempts like here and here).

For reproducing this experiment and analysis, see readme.

1 Initial data exploration

We will work with daily data composed of trading days in the interval '1999-01-01' to '2018-12-31'. The training set will start with '2000-01-03' (data prior to that date are used only as historical reference), validation starts with the date '2017-01-03'. The date is actually the first day in the year in data - we can observe, that the timeline does not contain all days - presumably only workdays, when the stock exchange is open.

Final testing of the found model should be done on a testing set, we do not cover that here, since this technical report aims to just demonstrate the methods and techniques to beat the baseline on a validation set.

The goal is to predict the S&P Volume. The first step is normalising the data - since just min-max normalisation (see 1 below) oscilates around a moving mean value over the time, we will be predicting the change to the next day (2):

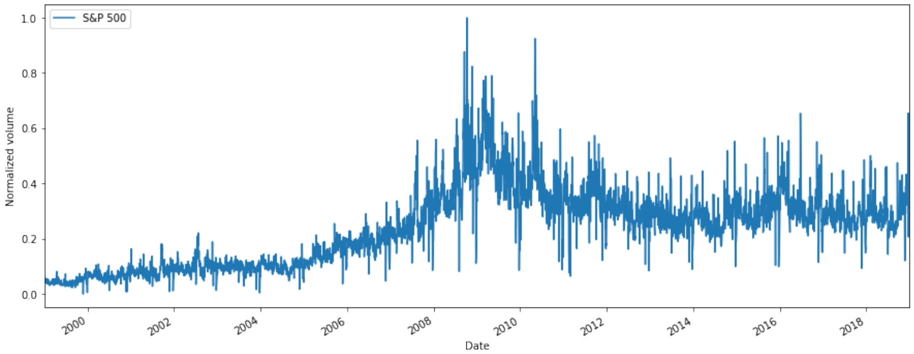


Figure 1: S&P 500 normalized volume over time.

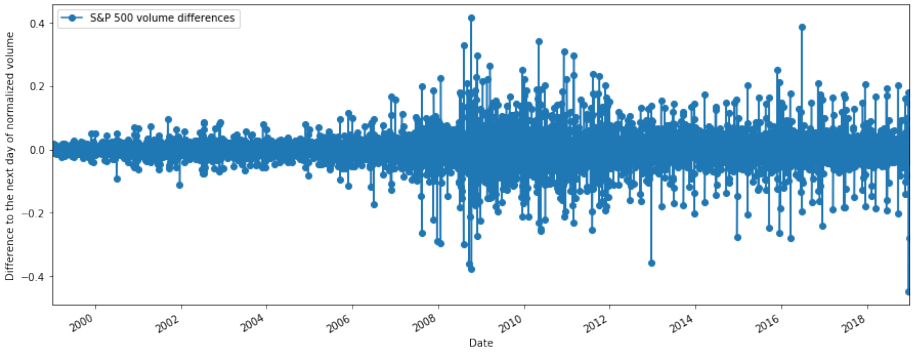


Figure 2: S&P 500 differences in volume to the next day.

(We could predict normalised change and not change of normalised value also). The changes have the following distribution (3) with most values centered around zero (only 700 datapoints are further away than one standard deviation):

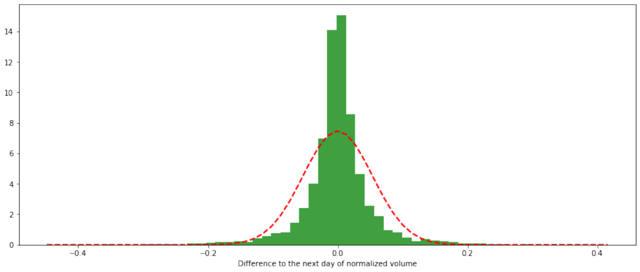


Figure 3: The distribution of volume differences

Lets find the correlations in the data itself - what could a linear model tell us when trying to predict next day from the past data? When we plot the correlations with itself, but shifted, we get this graph (4) :

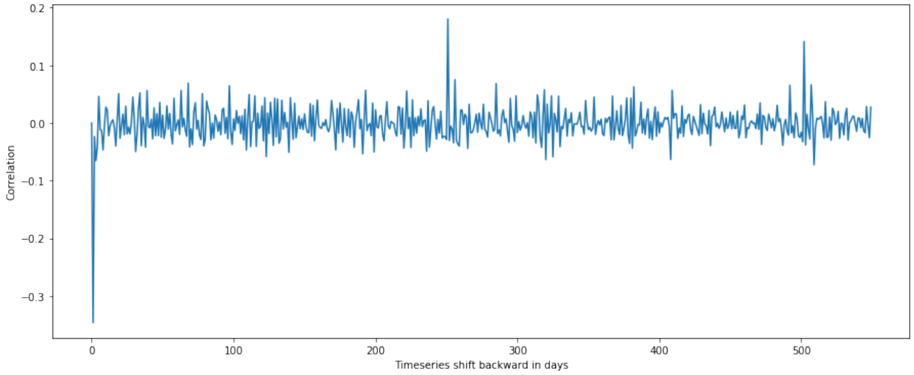


Figure 4: Graph of correlations of past days differences with the current

We find big peaks with correlations one day ago and (multiples of) 251 days before, which correspond to a situation of one calendar year before. See 5:

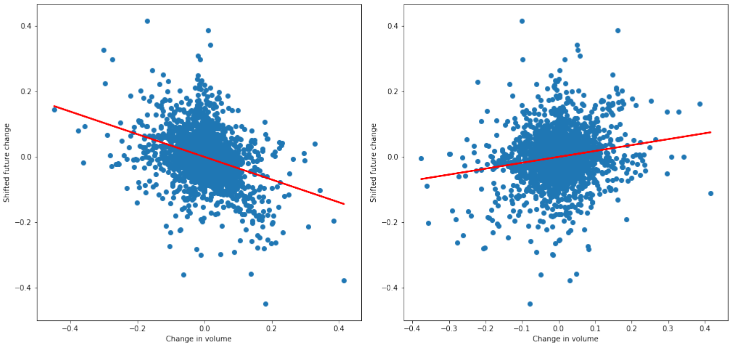


Figure 5: Scatter plot of differences, differences with one day shift and one year shift.

Fouriers transformation also shows (6) that there are peaks in the frequency domain that could be used for predictions in standard approaches (as in Stock predictions using fourier transforms Github notebook).

For convenience, the data exploration could be easily replicated using this google colab notebook.

1.1 Data sources used

Note, that when it comes to data sources for a machine learning model, we generally think, that we are limited only by the computing power and that the

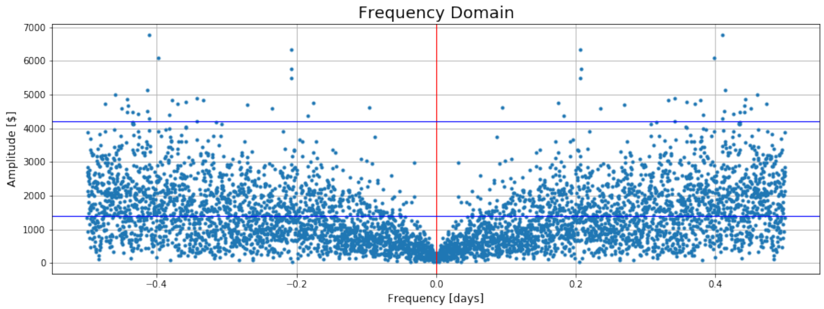


Figure 6: The fourier-transformed differences.

model si able to use all the features inthe right way. As a rule of thumb at a first glance it is true, nevertheless, there are methods that do feature selection to improve the prediction quality. A neural network is only an optimization algorithm walking through the multidimensional landscape, so any insight we could bring does count.

These datasources are used as an example, since we did no feature selection:

- First 3 pages of the most popular daily data on FRED database.
- Yahoo daily tickers: ’ˆDJI’, ’ˆIXIC’, ’ˆRUT’, ’ˆGSPC’ (min-max normalized).
- Policy uncertainity daily data from the US and UK (min-max normalized).
- Reddit worldnews channel.

  We took 25 most upvoted entries from each day, excluding most non-english entries and embedded them using GLOVE embeddings (glove.6B.50d.txt). The pre-processing and downloading codes are available at github.
- The actual date information with separated day of week, month and day in month. All embedded using positional embedding from attention is all you need, using sin, cos and linear functions, each embedded into 4 dimensions. Additionally the dayofweek and month are encoded using one-hot encoding.

Just to be sure, for FRED database we have checked the timeseries dates are indeed in the same format as Yahoo provides (FRED provides also S&P 500).

Missing values? Because the stocks trading days do not fully cover the datasources availability, a pre-processing is employed to gain the most from the existing information. When there is a gap present between the last trading day and the next one, the datasources could offer more daily data (based on calendar days). In that case we take minimal and maximal value from the data corresponding to all days in the gap (and the current day) of each non-reddit datasource.

For reddit news we take the most upvoted news from all the days in the gap.

Note, that when there is no gap present, we just copy the available data twice (as a minimum and maximum of 1 length sequence) to hold the format of 2 columns per feature per trading day the same.

2 The modelling setting

As stated before, we have normalized the data and aim predicting a relative change of the volume to the next day.

Note that the baseline of 'tomorow the stock's data are gonna stay the same' then becomes a constant zero prediction with a SSE score of 1.483.

We will use mean squared error as the model's loss function - since we do not change the timeseries lengths, it is equivalent (up to a scaling constant) to summed square error for gradient propagation.

Usually when dealing with large timeseries data, the data could be split into more (overlapping) windows by various strategies, but in our case we can fit everything into the memory at once, tensorflow will take care of proper gradient propagation.

As said, the data can be concatenated in the features axis (the last one) and fed into the network in one batch of size one. Therefore the dataformat is

- $[1, \text{timeseries length}, \text{features length}]$ for sources and $[1, \text{timeseries length}, 1]$ for target value.

The only exception is the embedded worldnews channel, which is fed in the format of:

- [batch, timeseries length, stories ordered, words per story, embedding size]$= [1, 25, 20, 50]$

In the worldnews channel branch of the input, we use:

- Two stacked convolutions over the 'words per story' dimension with 50 neurons (both with kernel size of 5 and second with dilation rate 2)
- A maxpooling and mean over the features
- Another convolution layer with 50 neurons and kernel size 5 (now over 'stories ordered' dimension)
- Followed with maxpooling and mean
- One 15% dropout

The resulting 3 dimensional tensor is then concatenated to other datasources.

All input data are (at the models beginning) also additionaly shifted and concatenated in the feature dimension to include 1 and 2 years older data and 1,2 and 3 weeks older data directly with contrast to current day data.

To faciliate the said shifts the model is present with older data from 1999-01-01 for training, whereas validation data can see all the training data.

Since we do not change the sequence length, training and validation loss is calculated using mean squared error, which is the same up to a constant to desired summed square error. Keras averaging over batches has no influence here, because we have one batch of size 1.

The losses are weighted by zeroes on out-of-set data (the data present only as a historical reference).

Since we want to use multiple datasources, we will not be using techniques predicting further ahead which are based on already predicted values (like here).

2.1 Prior experiments and insights into the modelling:

With all previously stated held fixed, we can vary architectures of the neural network. Some experiments with simpler architectures were conducted before wavenet to gather some insights (or to verify what the theories and common knowhow advises us):

- In the experiments there was a very strong local minimum, in a numerical or experimental sense, of constant zero being the baseline which we wanted to beat. Many experiments converged to that state.

- When 'traditional' set of activation functions is used (relu, tanh, sigmoidal), the network falls to the constant zero minima, presumably by the combination of trainable bias and the activation function's limits.

- On the other side, linear activation function tends to overfit the data.

  The issue can be solved by a strong regularizing dropout (45% in our case) that forces the predictors to oppose and support each other in the resulting sum and act as an ensemble. The effect can be succesfully seen on the learning curves (like at figure 7 on page 10), where the error is lower on validation set (where the dropout is turned off).

- Simple LSTM architectures, when presented with enough inputs, do avoid the 0-prediction trap and visually capture the changing trend of the data, but does not get over the baseline.

- Interestingly when the problem is formulated only as a sign prediction ('does the volume go up or down?'), we achieve 55% accuracy with LSTM, which is a tiny bit higher than the existing result. Unfortunately it cannot be calibrated to beat the baseline in the terms of squared error (because the errors are significant in areas with higher peaks). Some sources would suggest to use attention instead in this case.

  - By calibration we mean selecting separately the mean of positive and negative values.

- Simple Convolution architectures (even with acces to historical data), when not falling into the local minima, oscilated around a different mean than zero and thus also failed.

The next step was to apply a more complex architecture, specifically wavenet, known to produce good results in its original area of sound signal processing.

3 Wavenet

We have used Wavenet's architecture with 20 blocks with additional modifications, based on the observed prior behavior and gathered knowledge:

- replaced 'relu' wavenet's activations with 'LeakyReLU' to avoid the zero prediction minima (this activation allows for a small slope instead of zero)
    - LeakyReLU is used also for the convolutions in the reddit worldnews input
- replaced the final layers with 400 neurons dense layer, 45% dropout and final linear regression.
- In total the whole model has 11,334,756 weights.

From some experimental runs and their graphs, we can see that the network either:

- approximates the area on the end of the validation timeline
- predicts a recurring trend
- predicts a nonrecurring trend (but the model is not stable)
- or preferres to predict some big drops

Generally the results vary each run, which is an example of behavior we can get when there are big differences in the dimensionality of the timesteps, weights and features. It can give us a nice hint for opportunity for ensmbling later or for the use of scheduled learning rate.

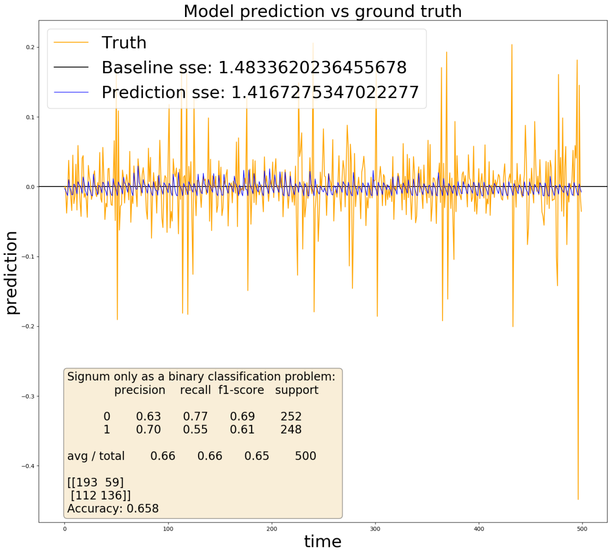
The best, in terms of SSE, was a model exhibiting the last charasteristics, as we can see from figure 8 on the next page. It did beat the baseline's score 1.483 with SSE of 1.4167 therefore we will focus on its analysis and verify the above claim (the claims about the other models can be verified by similar procedure).

wavenet14167 When we look at the predictions only in terms of binary classification (higher / lower), we find out, that not only the model achieves accuracy of 65.8% just by capturing one periodic trend in the data.

Scatterplot of ground truth-prediction (figure 9 on page 9) confirms the model predictions are correlated with the ground truth, but have a shorter range in absolute values.

When we linearly regress the squared error the model makes (figure 10 on the next page), we find out the linear coefficient (7.11e-06) is lower than the regressed error for the baseline (7.03e-06). Following the linear trend we would expect the market to oscilate more in the future and this model to continue capturing the recurring trend.

To see how the model got to these results, we can look at the performance on the training set on figure 11 on page 10. There it behaved similarily, also beat the baseline in terms of sse and direstion too.



```
Signum only as a binary classification problem:
             precision    recall  f1-score   support

          0       0.63      0.77      0.69       252
          1       0.70      0.55      0.61       248

avg / total       0.66      0.66      0.65       500

[[193  59]
 [112 136]]
Accuracy: 0.658
```

Figure 8: Model prediction vs ground truth on validation set (zoom on predictions available in the repository).

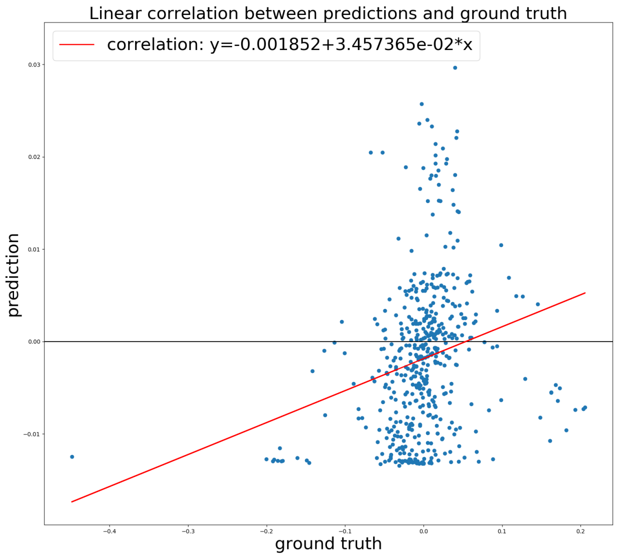


Figure 9: Scatterplot of ground truth-prediction shows the predicted spikes.

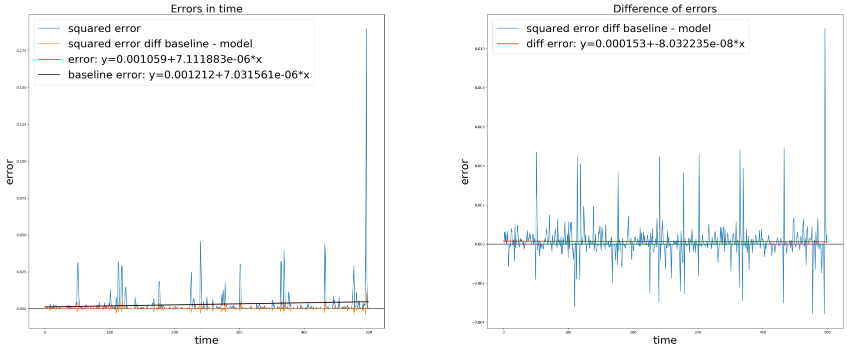
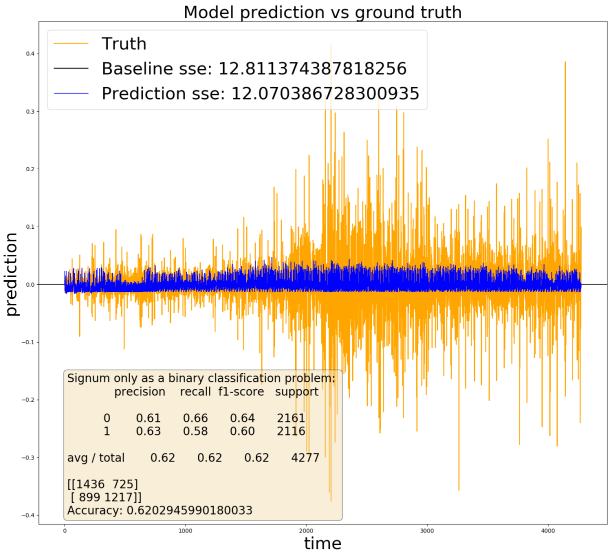


Figure 10: Errors that the baseline and model make, together with their difference (where positive values mean that our model did beat the baseline)



```
Signum only as a binary classification problem:
             precision    recall  f1-score   support

          0       0.61      0.66      0.64      2161
          1       0.63      0.58      0.60      2116

avg / total       0.62      0.62      0.62      4277

[[1436  725]
 [ 899 1217]]
Accuracy: 0.6202945990180033
```

Figure 11: Model predictions on training data (zoom on predictions available in the repository)

3.1 Model analysis

Apart from what we saw in the predictions, we will inspect the model's robustness, flexibility and behavior with extremal values to a specific level of detail.

We should take a note to also inspect two observed phenomena, that are not done in this report:

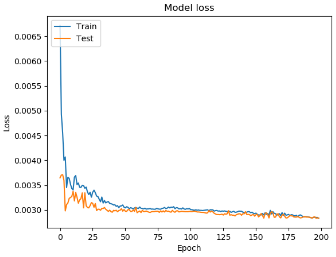


Figure 7: The graph of the loss during training

- What is causing the model to catch the specific frequency observed (by trying to influence just that prediction by modifying the source data, for example by removing that frequency, while holding the model fixed)
- An interesting observation is that the model's error also oscilates. Another analysis could be employed to analyse if the data has some periodic components.

Usually there are two methods, that evaluate how a model is tied to its input:

- observing the model's prediction based on changed / corrupted inputs.

  A robust model should not react to random non-systematic noise. Ideally we would hope that it would be self-correcting in the manner, that when other data sources are not corrupt, it could ignore the noise. (In a practical setting we could imagine having another model trained that would detect anomalies like that to raise a flag for us.)

- computing the derivatives of the input data with respect to the predictions or loss.

  We can discover more important inputs having higher absolute value of the gradient.

Some works suggest, that both methods are not precise when duplicite or somehow corellated inputs are present, but nevertheless they will give us some insights.

There are, of course, other methods that allow us to inspect robustness (also here), also more sources on interpretability exist.

Data corruption We can evaluate (in terms of SSE) the trained model on data corrupted in many different ways. For our analysis we will look at 3 possibilities:

- Setting a random share of all the input data (with fixed percentage) to zero
- Setting to a plus/minus higher value
- Or to a maximal/minimal value. Since our data are normalised to 0.0-1.0, the negative values also test the model on completely errorneous datapoints.

The plots (figure 12 on the next page) show three trials for each integer percentage from zero to 100 % (just to see the overall shape).

As we can see from the plots, the highest effect of distortion is achieved using the maximal values and ±0.5 methods, where the results are significantly different from the original model's at the level of 6 % of random data distortion (and exhibit the biggest change in SSE when we change all the inputs).

Significant difference condition is decided to be a case when all the trials performance drops by more than 5 % of the difference between baseline SSE and the model's SSE. The criterium is chosen to illustrate the process and could be modified based on the application.

Possibly thanks to the dropout regularization, our model is more resilient to random zero inputs in the source (or markets) data, but sensitive to spikes in the data. Therefore if the inputs begin to differ unusually in magnitudes (while keeping only the volume distribution somehow same) we should not use this model, but retrain, even though we have the 6 % reserve.

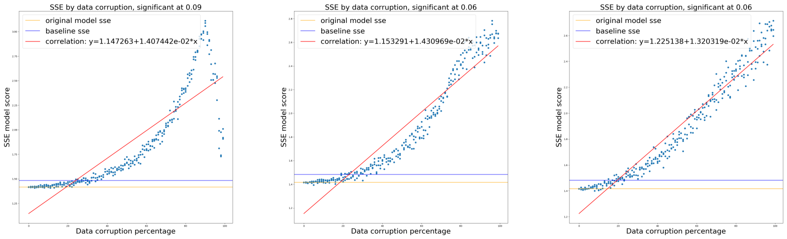


Figure 12: Three modes of data corruption (zeroes, around ±0.5, ±1.0) plotted against the set percentage to corrupt.

Since we have groupped inputs by a datasource, we can investigate the importance of the inputs again using this method of data corruption applied only on some inputs.

- The most important input was discovered (as seen in figure 13 on the following page) to be was from policy uncertainty markers and FRED, as we would suspect the market depend on the political influences.
- Second most important input was the input calendar date embedded using positional embeddings, which hints at stronger dependency on seasonality and/or holidays.

Detailed analysis have uncovered this model depends heavily on specific FRED AAA10Y marker, which, if set to zero, would send immediately the prediction over the baseline's score and make the model mostly predict positive values. Whereas the other markers had zero impact on the prediction alone. That does not have to mean they are not important or not used, but that they might be:

- duplicit with other data
- less important alone as the dropout mechanism supresses the inidividual importance
- helping only the gradient descend during training to reach this optimum

This hypothesis can also be verified by training the model with the AAA10Y marker alone (not done here).

Second method - gradients with respect to inputs By looking at the gradients of the validation's set model's SSE with respect to the inputs and then summing or maxing them out by features or time dimension, we can asses the inputs importance by a different way.

It allows us to see which dates contributed the most to the final SSE, as in figure 14 on page 14.

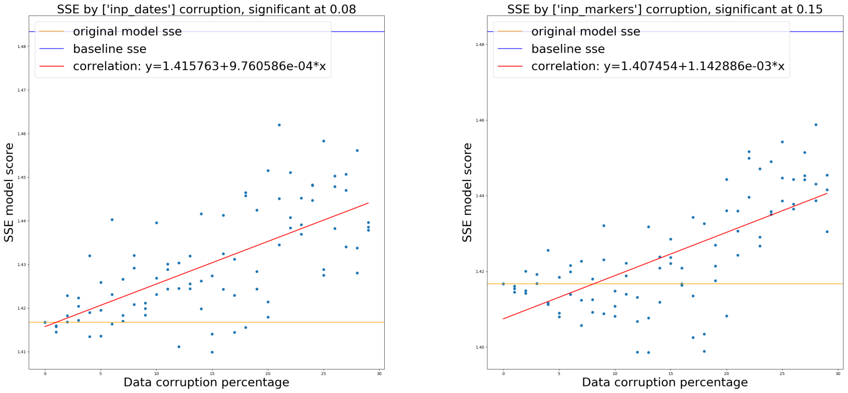


Figure 13: Inputs importance by drop in SSE.

Moreover we could also do this timestep importance analysis per each single day's prediction (and not only the loss) and max/sum it up. We would get a graph of relative importances to tomorow's prediction.

Reducing the gradients along the time dimension shows the model is in fact sensitive to all the input markers. The contrast with the previous method means, it was not able to evaluate it individually or that the model would be sensitive only to smaller changes (as derivatives approximate only the neighbourhood). That is a good news, because smaller changes should count as the model should be sensitive to meaningful data. (Our assumption here is, that smaller changes are more meaningful than random noisy corruption.)

3.2 Conclusion

The model has discovered a specific repeating trend in the data and did not focus on capturing the amplitude much. The main dependency was the FRED AAA10Y marker. Practically the model should be used as a tool to predict ups or downs more than the magnitude.

In this technical report, we have not tested the model on the data from the year 2019, which should be done next, to verify if the procedure hadnt optimized on the validation set also and if the periodic trend does continue.

We should also remember to periodically check the model's basic assumptions like the minmax ranges of values and the percentage of extremal values and retrain if the conditions change.

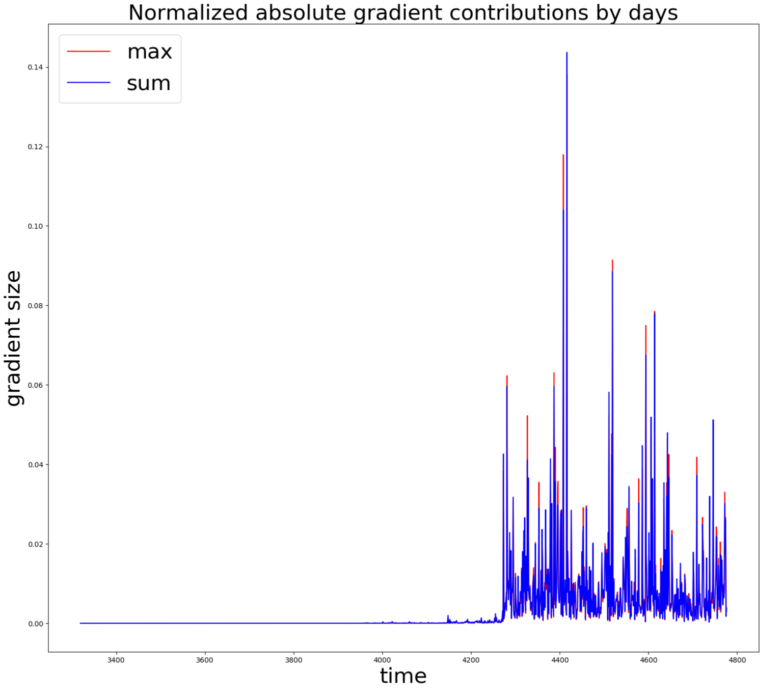


Figure 14: The models sensitivity to inputs in time, by gradients.

4 Possible next steps

So far we have tried multiple architectures and datasources. More datasources can be added - from social platforms like twitter or a calendar of holidays or economic events (as here, or here), that could influence the markets behavior. We could also add another target for the prediction in a hope it would work in a synergy with the trade volume and boost the prediction quality. Or use transfer learning from a different target.

What remains is to try either more advanced architectures (GANs) and/or add standart predictors (ARMA, (G)ARCH(X) models) as features into the network (as is described for example here), or a completely different approach such as predicting changes in a fourier space of a time window.

To create more reliable models, we could incorporate confidence intervals into the prediction (like here) using for example gaussian mixtures, ensembling (based on purely different starting point or even crossvalidation based on the timeseries splits) or anomaly predictors (using, for example, autoencoders).