1 Models being considered

Many papers have shown that GARCH, ANN and LSTM models can work together to provide a more nuanced view of volatility than single model. We focus on this proposition, and investigate the following models:

- **Machine Learning Models**
  - **NBEATSx** - an improvement of NBEATS with exogenous variables
  - **LSTM** which is typically combined with GARCH to improve performance
  - **TCN** which captures long range dependencies more efficiently than LSTM and have shown to be competitive with LSTMs
- **GARCH and their variations**
  - **GARCH** to capture stylised fact of volatilty clustering
  - **GJR-GARCH** to capture the stylised fact that negative returns are followed by more volatility compared to positive returns
  - **HAR** where an asset's conditional volatility is modelled as a linea function of (1) Previous day's realized volatility (2) Previous week's realized volatility (3) Previous month's realized volatility

    As quoted by Corsi[2009], the main idea is that agents with different time horizons perceive, react to, and cause different types of volatility components. Simplifying a bit, we can identify three primary volatility components: the short-term traders with daily or higher trading frequency, the medium-term investors who typically rebalance their positions weekly, and the long-term agents with a characteristic time of one or more months.

However, an important consideration for training these models is the use of GANS. Unfortunately, there is limited repositories out there with working code for how these methods can be integrated with WGAN for time series prediction. This gap is significant because WGAN offers potential benefits in generating realistic synthetic data, which could enhance the robustness of forecasting models, by uncovering novel features of the data set.

Given that most of the models are well-known, we will only give a brief discussioin on NBEATSx which is arguably the most complex model here.

2 NBEATSx

We first give the overall architecture of the NBEATSx model below. As a broad overview, the NBEATSx framework separates local nonlinear projections of the target data onto basis functions across each of its various blocks to decompose the objective signal. Here the objective signal is represented by the vector $\mathbf{y}$, the inputs for the model is the backcast window vector $\mathbf{y}^{back}$ of length L and the forecast window vector $\mathbf{y}^{for}$ of length H. Here L denotes the length of the lags available as class autoregressive features, and H is the forecast horizon treated as the objective.

The main difference between this and the original NBEATS model is that it also incorporates covariates in its analysis denoted by the matrix $\mathbf{X}$. This is the exogenous matrix that will be included on top of the backcast vector.

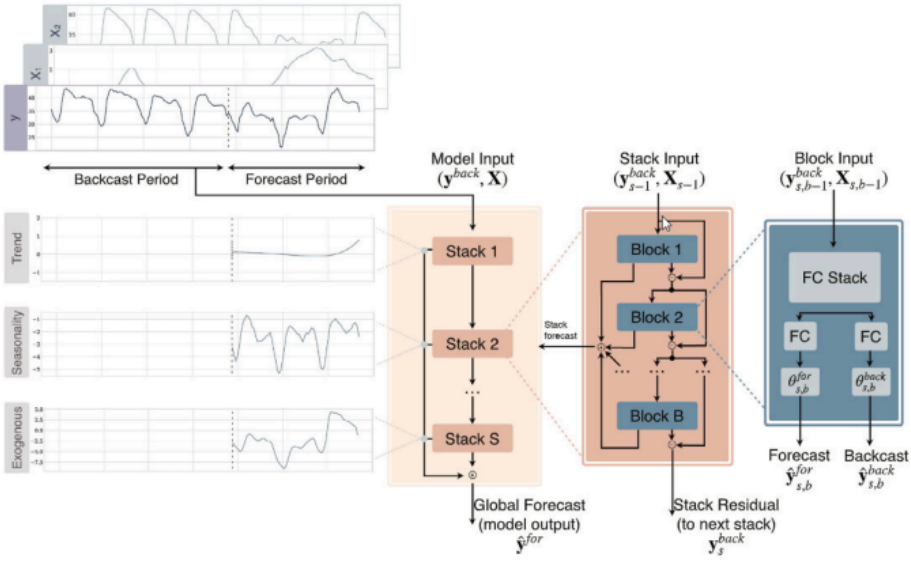


Figure 1: NBEATSx architecture

2.1 Blocks

For a given s-th stack and b-th block within it, the NBEATSx model performs two transformations, depicted in the blue rectangle of the figure above. The first transformation takes the input data $\left(\mathbf{y}^{\text{back}}_{s,b-1}, \mathbf{X}_{s,b-1}\right)$ and passes it through a **Fully Connected Neural Network** to learn hidden units $\mathbf{h}_{s,b} \in \mathbb{R}^{N_h}$. We then perform a simple linear transformation to obtain the forecast $\theta^{for}_{s,b} \in \mathbb{R}^{N_s}$ and $\theta^{back}_{s,b} \in \mathbb{R}^{N_s}$ coefficients that will be used in the basis expansion, where N_s is the dimension of the stack basis.

$$\mathbf{h}_{s,b} = \text{FCNN}_{s,b}\left(\mathbf{y}^{s,b-1}_{\text{back}}, \mathbf{X}_{s,b-1}\right)$$
$$\theta^{s,b}_{\text{back}} = \text{LINEAR}^{\text{back}}(\mathbf{h}_{s,b}), \qquad \theta^{s,b}_{\text{for}} = \text{LINEAR}^{\text{for}}(\mathbf{h}_{s,b})$$

The second transformation is a basis expansion operation. Here the basis functions can vary depending on the kind of pattern being captured, for example we may use **polynomial basis functions** to capture

trends, or **fourier basis functions** to capture seasonality. Using the block's basis vectors $\mathbf{V}_{s,b}^{\text{back}} \in \mathbb{R}^{L \times N_s}$ and $\mathbf{V}_{s,b}^{\text{for}} \in \mathbb{R}^{H \times N_s}$, we have the following transformations to produce our backcast $\hat{y}_{s,b}^{\text{back}}$ and our forecast $\hat{y}_{s,b}^{\text{for}}$

$$\hat{y}_{s,b}^{\text{back}} = \mathbf{V}_{s,b}^{\text{back}} \theta_{s,b}^{\text{back}} \quad \text{and} \quad \hat{y}_{s,b}^{\text{for}} = \mathbf{V}_{s,b}^{\text{for}} \theta_{s,b}^{\text{for}}$$

2.2 Stacks and residual connections

Looking at the first block, we see that the predicted backcast $\hat{y}_{s,1}^{\text{back}}$ is subtracted from the earlier input $y_{s,1}^{\text{back}}$ before being fed into the next block. Repeating this process is similar to the concept of **boosting**, where the next block focuses on modelling residuals not explained yet by the current block. Simultaneously, we also aggregate the partial forcasts $\hat{y}_{s,b}^{\text{for}}$ from each block to get the stack forecast y_s^{for}

$$y_{s,b+1}^{\text{back}} = y_{s,b}^{\text{back}} - \hat{y}_{s,b}^{\text{back}} \quad \text{and} \quad \hat{y}_s^{\text{for}} = \sum_{b=1}^{B} \hat{y}_{s,b}^{\text{for}}$$

2.3 Model Prediction

The final prediction $\hat{\mathbf{y}}^{for}$ of the model can then be obtained by summation of all the stack predictions

$$\hat{\mathbf{y}}^{for} = \sum_{s}^{S} \hat{y}_s^{\text{for}}$$

The original NBeats model contains the trend stack followed by the seasonality stack, each containing three blocks. NBEATSx extends this by adding 1 more exogenous stack, where we focus on the basis having a **generic configuration**. This means that the basis vectors are not based on well-understood, predefined basis functions as mentioned earlier, but instead is learned in a more flexible manner as the model adapts the data.

$$\hat{y}_{s,b}^{\text{exog}} = \sum_{i=1}^{N_c} C_{s,b,i} \theta_{s,b,i}^{\text{for}} \equiv C_{s,b} \theta_{s,b}^{\text{for}} \quad \text{with} \quad C_{s,b} = \text{TCN}(\mathbf{X}) \tag{1}$$

Specifically, the TCN is effectively expanding the basis by learning a transformation of the exogenous variables (like future or historical external factors) into a useful form that adjusts the forecast. This is different from classic N-BEATS, where the basis expansion is done directly on the time series itself.

2.4 Exogenous Variables

We distinguish exogenous variables by whether they reflect static or time-dependent aspects of the modeled data. *Static* exogenous variables carry **time-invariant information**, meaning they do not change over the time of the forecasting period.

These variables are used when forecasting multiple time series simultaneously, especially when some time series share certain characteristics. For instance, if you are forecasting the sales for different regions, the **region identifier** would be a static exogenous variable. This means that the model can generalize patterns across series with the same static variable value, improving the overall accuracy of the forecast.

As for the *time-dependent* exogenous covariates, we discern two subtypes. First, we consider **seasonal covariates** from the natural frequencies in the data. These variables are useful for NBEATSx to identify seasonal patterns and special events, both within and outside the historical lookback window. Secondly, we consider **Domain-specific temporal covariates**. These are specific to the problem being solved. For example, in energy forecasting, this might include variables like electricity load or renewable energy production.

3 Integration of Machine Learning and Time Series Models

3.1 SP500

We first test our models on the S&P500 data from 2000 to 2023, where the realized volatility proxy chosen for this is Yang & Zhang's realized volatility since this proxy is independent of the drift, unbiased in the continuous limit, and consistent in dealing with opening price jumps (Yang & Zhang, 2000). Its choice is also motivated by its accessibility thanks to the fact that it only requires High, Low, Open, and Close prices for its estimation.

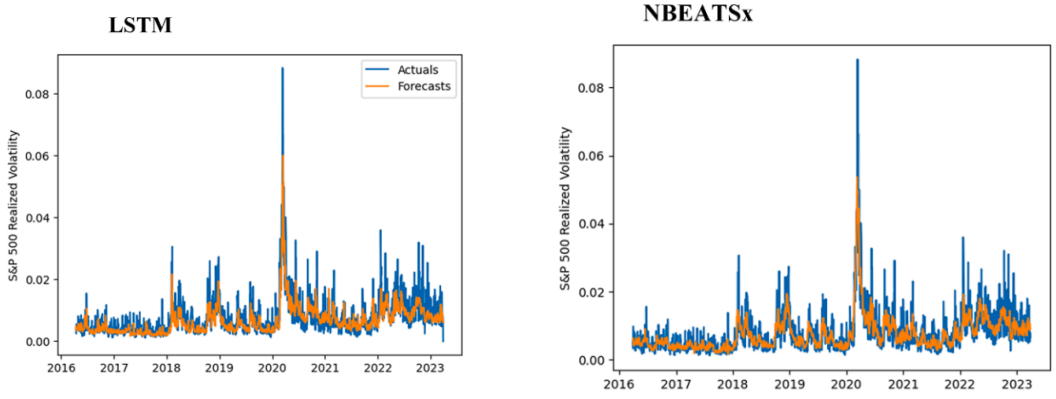
The important part here is that we introduce the **VIX** closing prices as part of our exogenous variables that we feed into the NBEATSx/TCN/LSTM model. This is similar to using the GARCH inputs for the LSTM model, which is more commonly known as GARCH-LSTM. However, the results on the VIX-LSTM model yielded better results, probably because the VIX is an index of implied market volatilities which has higher correlations with the realised volatility compared to predicted GARCH inputs. Here RMSE stands for the Root Mean Squared Error, while the MA is calculated as 1 - MAPE, where MAPE is the Mean Absolute Percentage Error. We run the models several times with different random seeds to get the following error measures.

Error Measures	NBEATSx-VIX	TCN-VIX	LSTM-VIX	HAR	GARCH(1,1)	GJR-GARCH(1,1,1)
RMSE	$0.246\% \leq X \leq 0.253\%$	$0.248\% \leq X \leq 0.280\%$	$0.235\% \leq X \leq 0.264\%$	0.265%	0.295%	0.295%
MA	$63.78\% \leq X \leq 65.37\%$	$52.34\% \leq X \leq 66.39\%$	$59.67\% \leq X \leq 70.06\%$	60.70%	59.83%	59.63%

Main Sample Results for $H = 1$ Forecast

The range of values for the error measures for NBEATSx, LSTM and TCN is due to the different random seeds we use for weight initalisation each time, which leads to diff error measures on each run. Regardless, we can see that the NBEATSx model is more robust and consistently performs with an accuracy of about 64%, which the LSTM-VIX model achieves similarly but with much greater variance. The TCN model however performs subpar compared to both the LSTM model and NBEATSx model and has a much wider range of values for accuracy (MA).

The visual representations of our forecasts for NBEATSx and LSTM are here:



(a) LSTM forecast

(b) NBEATSx forecast

The NBEATSx model trains significantly faster, with the 150 epochs taking approximately **4 seconds** despite running on the SP500 data spanning 23 years. For our LSTM model, despite there being only 1 LSTM layer of 56 neurons and a single Dense layer, it takes significantly longer to train the model, taking approximately **10 minutes** over 7 epochs. Our TCN model which runs on 15 epochs takes even longer than that, averaging at about **13 minutes per run**. Therefore, we can see that NBEATSx is superior in terms of both **robustness** and **speed** while achieving similar accuracy.

We can also see that the GARCH, GJR-GARCH and HAR models perform worse than our Machine Learning models, suggesting their superiority with the integration of the VIX index as an input.

3.2 Oil Prices

We also look at oil prices which are infamous for having periods of high volatility. We calculate the volatility using a rolling window of size 252 and annualize it for a simpler method of calculating the realized volatility.

We only run the 2 Machine Learning models to confirm their performance on a different dataset which has peaks in the volatility to see how well our model can forecast them. An important note here is that we have included the **OVX** index as part of our inputs since the crude oil volatility index should naturally have a high correlation with the realized volatility. Below we have the results after running both models for 10 runs (more can be done for more accurate bounds but we do 10 in the interest of tiimie)

Error Measures	**NBEATSx-OVX**	**TCN-OVX**	**LSTM-OVX**
RMSE	$6.24\% \leq X \leq 6.30\%$	$6.50\% \leq X \leq 6.89\%$	$6.20\% \leq X \leq 6.70\%$
MA	$79.05\% \leq X \leq 79.43\%$	$76.20\% \leq X \leq 78.65\%$	$77.13\% \leq X \leq 79.80\%$

Main Sample Results for $H = 1$ Forecast

Again we observe that NBEATSx outperforms LSTM in terms of robustness by consistenly producing forecasts with a slightly higher accuracy and a consistently lower RMSE. Looking at the forecast volatility against the true value shows promising results.

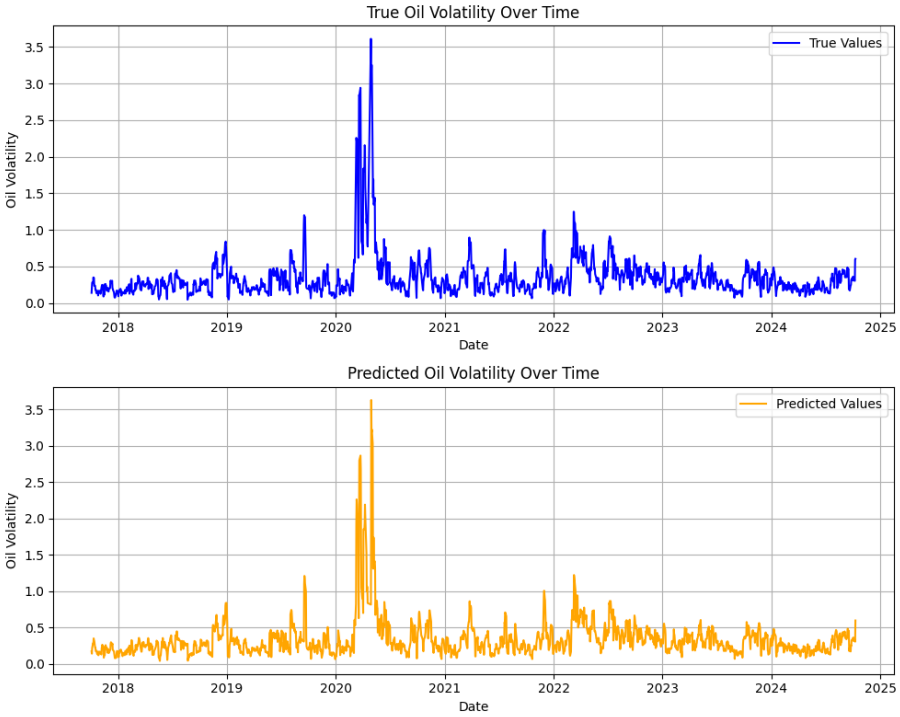


Figure 2: NBEATSx forecast on oil volatility

4 GANS for synthetic Time Series Generation

4.1 WGAN-GP

Training a GAN is highly non-trivial. The actions of the 2 different networks can result in **non-convergence**, which is a very likely scenario. Mode collapse, also known as the *Helvetica Scenario*, is a phenemenon that arises when the training results in failure. It can manifest as a generator that maps all our input noise vectors $\mathbf{z}$ to the same output point, or as a generator that misses certain regions of our sample distribution $\mathbb{P}_{data}$.

We omit a formal discussion of the challenges resulting from the topologies induced by our loss functions, and instead simply state without proof that Wasserstein GAN (WGAN) gradient penalty (GP) is a GAN setup that induces a much weaker topology to allow for a more robust training process.

Ignoring the technical details, we can define the Wasserstein-1 distance as

$$W(\mathbb{P}_{data}, \mathbb{P}_{model}) = \inf_{\gamma \in \Pi(\mathbb{P}_{data}, \mathbb{P}_{model})} \mathbb{E}_{(x,y)\sim\gamma} \left[\|x - y\|\right]$$

where $\Pi(\mathbb{P}_{data}, \mathbb{P}_{model})$ denotes the set of all joint distributions $\gamma(x, y)$ whose marginals are $\mathbb{P}_{data}, \mathbb{P}_{model}$ respectively, and $\|.\|$ is a metric.

Intuitively, it indicates how much *mass* needs to be transported from x to y in order to transform the distribution $\mathbb{P}_{data}$ into $\mathbb{P}_{model}$ (vice versa in this case).

Since we cannot possibly exhaust all joint distributions to comput the above equation, there is a proposed transformation based on the *Kantorovich-Rubinstein Duality*:

$$W(\mathbb{P}_{data}, \mathbb{P}_{model}) = \frac{1}{K} \sup_{\|f\|_L \leq K} (\mathbb{E}_{\mathbf{x}\sim\mathbb{P}_{data}}[f(\mathbf{x})] - \mathbb{E}_{\mathbf{x}\sim\mathbb{P}_{model}}[f(\mathbf{x})])$$

where we demand f to be K-Lipschitz continuous(this just bounds the diff in the expected values by $K(x_{data} - x_{model})$ so that the Wasserstein distance is well-behaved.

Analagously in the case of GANs, we have the following equation

$$L = \mathbb{E}_{x\sim p_{\text{data}}} \left[D_{w(D)}(x)\right] - \mathbb{E}_{x\sim p_{\text{data}}(z)} \left[D_{w(D)}(G_{w(G)}(z))\right] + \lambda \mathbb{E}_{\hat{x}\sim\mathbb{P}_{\hat{x}}} \left[\left(\|\nabla_{\hat{x}} D_{w(D)}(\hat{x})\|_2 - 1\right)^2\right]$$

The penalty is added to make sure the discriminator is 1-Lipschitz continuous. This is important to ensure the discriminator has a controlled gradient limited to a value of 1. If the discrinator's gradient were too big or small, then the generator would receive too much or too little feedback, leading to unstable training. We omit the technical math behind this claim and simply take it at face value.

We test WGAN-GP on the closing prices of GOOG from the year 2013 to 2023 (a 10 year period). We can see that the synthetic data roughly follows the direction of the closing prices, albeit with much more spikes and a seemingly lacklustre performance on the closing prices within the years 2023. Nevertheless, it provides extra training data from which our model can learn.

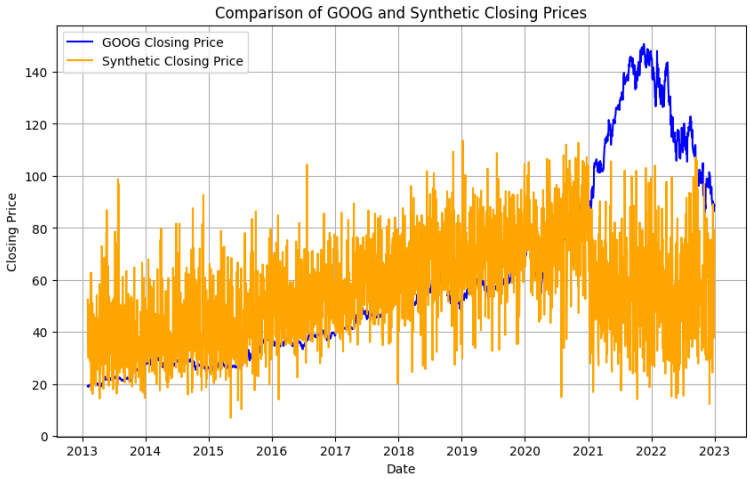


Figure 3: Synthetic data on Google

If we perform the same procedure on our realised volatility of the SP500, the results aren't as promising. I suspect that the values may be too small, and it will be better if we just perform the WGAN-GP on the closing prices of the SP500 and calculate the rolling volatility from there. It could also be attributed to a lacklustre choice of hyperparemeters, but we will have to further test our these theories.

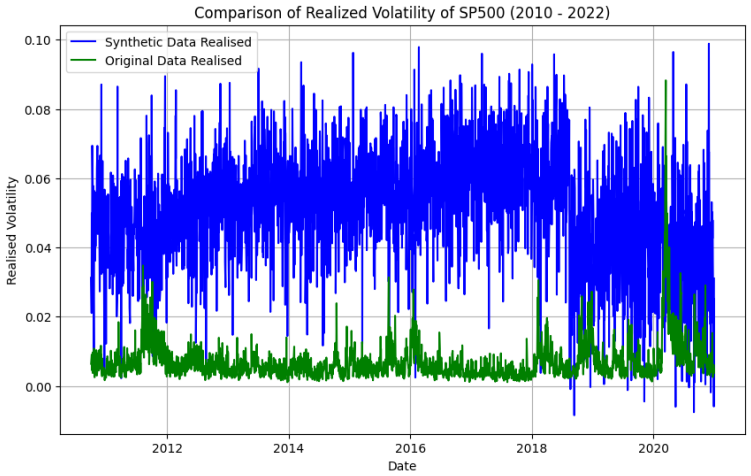


Figure 4: Synthetic data on SP500 Realized Volatility

A recent paper has however showed that the inclusion of WGAN-GP can drastically improve the performance of the models. The code repository is unfortunately not available, but the results look promising and proves that WGAN-GP can be applied to forecasting volatility.

Model	R^2	MSE	MAE	RMSE	MAPE (%)
GARCH	0.074	0.836	0.681	0.914	40.797
ANN	0.237	0.683	0.567	0.826	23.158
LSTM-ANN	0.241	0.673	0.560	0.820	23.913
BLSTM-ANN	0.231	0.687	0.565	0.829	23.931
GARCH-ANN	0.734	0.203	0.295	0.450	14.497
GARCH-LSTM-ANN	0.725	0.206	0.293	0.453	13.872
GARCH-BLSTM-ANN	0.731	0.204	0.294	0.452	23.453

Figure 5: Metrics without WGAN-GP

Table 8. Prediction performances for the oil return data with WGAN Gradient Penalty.

Model	R^2	MSE	MAE	RMSE	MAPE
ANN-WGAN	0.980	0.004	0.049	0.065	0.317%
LSTM-ANN-WGAN	0.977	0.050	0.050	0.069	0.263%
BLSTM-ANN-WGAN	0.977	0.005	0.054	0.070	0.284%

Figure 6: Metrics with WGAN-GP

A thing to note is that the authors have trained the WGAN-GP on 10,000 epochs, and the baseline LSTM model on 100 epochs, which will take a significant amount of time and may thus be infeasible for short-term forecasting. Nevertheless this is worth pursuing and we can reduce some hyper-parameters to see if we can obtain similar results.

There are also other variants of GANs that we have yet to explore, including cGAN, FIN-GAN and TimeGAN. This will be done over the next couple of weeks.