Published in Towards Data Science

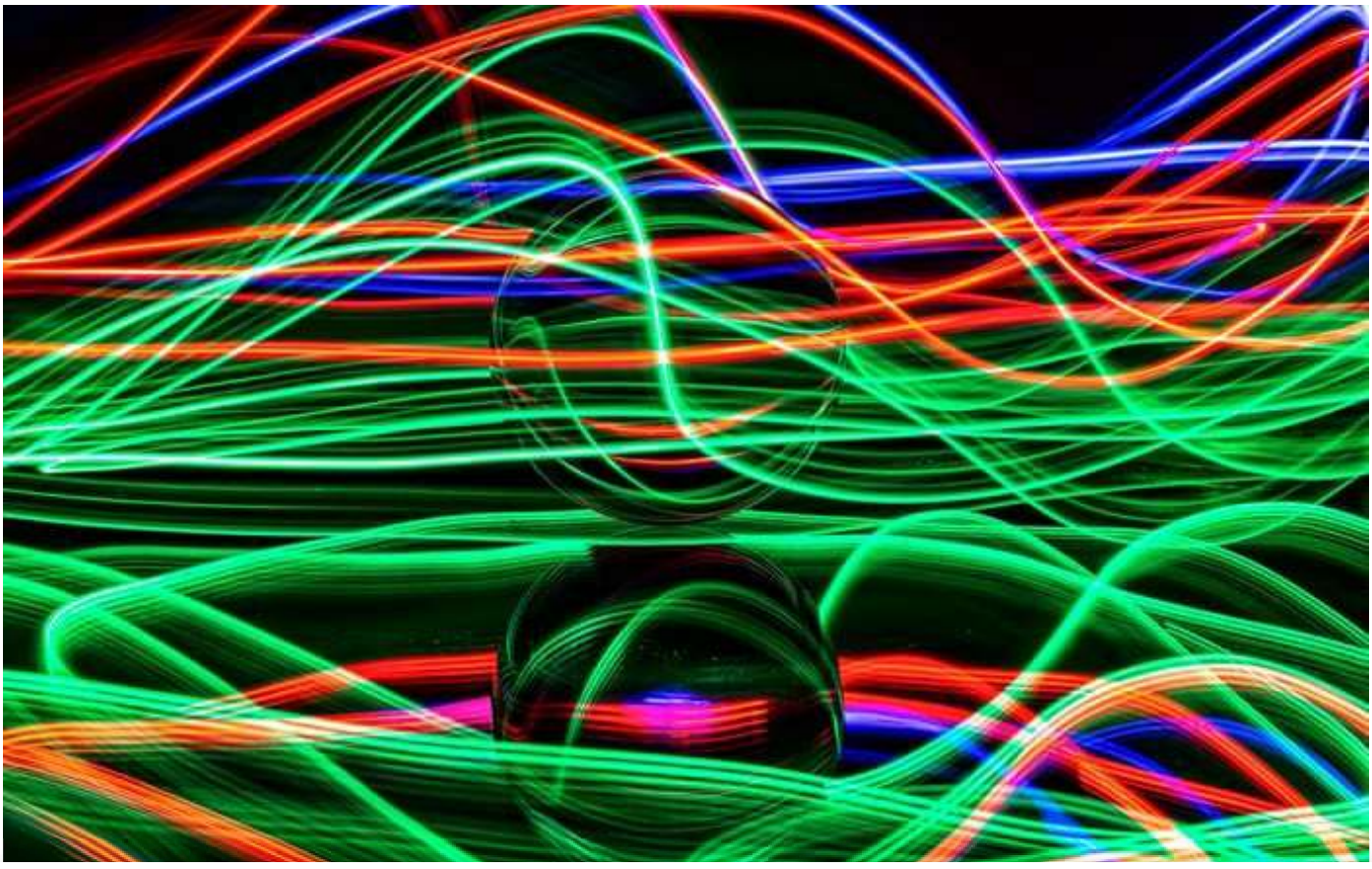
Nikos Kafritsas Follow

Dec 20, 2021 · 10 min read · · Listen

Save

The Best Deep Learning Models for Time Series Forecasting

Everything you need to know about Time Series and Deep Learning



> **Be sure to SUBSCRIBE here to never miss another article on data science topics, projects, guides and more!**

Preliminaries

The landscape of Time Series forecasting has changed dramatically in the span of two years.

The forth and fifth in the series of **Makridakis M-competitions** (better known as **M4** and **M5** competitions respectively) took place in 2018 and 2020. For those who are not aware, these M-competitions are essentially a status-quo for the time series ecosystem, offering empirical and objective evidence that guides the theory and practice of forecasting.

The results of the M4 competition in 2018 showed that the pure 'ML' methods were largely outperformed by the traditional statistical approaches. This was unexpected, given that Deep Learning had already left an *indelible* imprint on other fields such as Computer Vision and NLP. However, in the **M5 competition**[1] two years later, with a more creative dataset, the top spot submissions featured only 'ML' methods. To be more precise, all 50 top-performing methods were ML-based. This competition saw the rise of the versatile ***LightGBM*** (used for time series forecasting) and the debut of **Amazon's *DeepAR***[2] and ***N-BEATS***[3]. The *N-BEATS* model, published in 2020, outperformed the winner of the M4 competition by 3%!

The recent **Ventilator Pressure Prediction** Kaggle competition showcased the importance of using deep-learning methods to tackle real-case time series challenges. Specifically, the goal of the competition was to predict the time sequence of pressure within a mechanical lung, given the time series of control inputs. Each training instance was essentially a time series of its own, thus the task was a multiple time series problem. The winning team submitted a multi-level deep architecture, which included, among others, an LSTM network and a Transformer block.

In the past few years, many notable architectures have been published such as the *Multi-Horizon Quantile Recurrent Forecaster (MQRNN) and the Deep Space-State Models (DSSM).*

- **Versatility:** The ability to use the model for different tasks.
- **MLOps**: The ability to use the model in production.
- **Interpretability and explainability:** Black-box models are not so popular anymore.

This article discusses 4 novel deep learning architectures specialized in time series forecasting. Specifically, these are:

1. **N-BEATS** (ElementAI)
2. **DeepAR** (Amazon)
3. **Spacetimeformer** [4]
4. **Temporal Fusion Transformer or TFT** (Google) [5]

The first two are more battle-tested and have been used in many deployments. *Spacetimeformer* and *TFT* are also exceptional models and propose many novelties. They are able to take advantage of new dynamics, beyond the time series context.

N-BEATS

This model came straight from the (unfortunately) short-lived ElementAI, a company cofounded by **Yoshua Bengio**. The top level architecture along with its main components is shown in **Figure 1:**

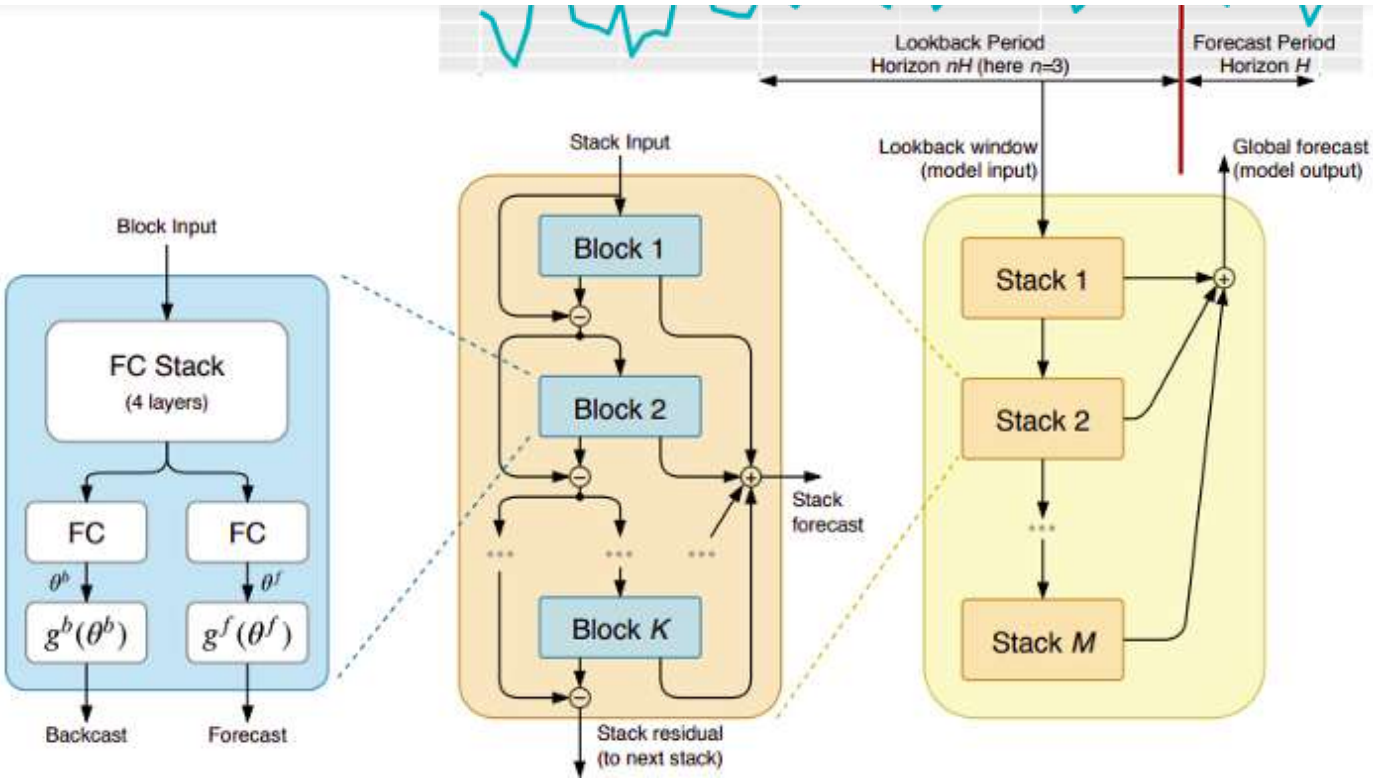


Figure 1: The N-BEATS architecture (Source)

Essentially, *N-BEATS* is a pure deep learning architecture based on a deep stack of ensembled feed forward networks that are also stacked by interconnecting *backcast* and *forecast* links.

> Each successive block models only the residual error due to the reconstruction of the backcast from the previous block and then updates the forecast based on that error. This process mimics the Box-Jenkins method when fitting ARIMA models.

These are the model's key advantages:

Expressive and easy to use: The model is simple to understand and has a modular structure (blocks and stacks). Moreover, it is designed to require minimal time-series

In the *N-BEATS* implementation, this is achieved through *meta-learning*. Specifically, the meta-learning process consists of two procedures: the inner and the outer learning procedures. The inner learning procedure takes place inside blocks and helps the model capture local temporal characteristics. On the other hand, the outer learning procedure takes place inside stacks and helps the model learn global characteristics across all time-series.

Doubly Residual Stacking: The idea of residual connections and stacking is so brilliant that it is used in almost every type of deep neural networks, such as deep *Convnets* and *Transformers*. The same principle is applied in the *N-BEATS* implementation, but with some extra modifications: Each block has two residual branches, one running through the lookback window (called *backcast*) and the other through the predicted window (called *forecast*).

Each successive block models only the residual error due to the reconstruction of the *backcast* from the previous block and then updates the *forecast* based on that error. This helps the model better approximate the useful backcast signals, while simultaneously, the final *stack forecast* prediction is modeled as an hierarchical sum of all partial forecasts. Interestingly, this process mimics the **Box-Jenkins method** when fitting **ARIMA** models.

Interpretability: The model has two variants, *general* and *interpretable*. In the *general* variant, the final weights in the fully-connected layers of each block are learned by the network arbitrarily. In the *interpretable* variant, the last layer of each block is removed. Then, the *backcast* and *forecast* branches are multiplied by specific matrices that mimic **trend** (monotonic function) and **seasonality** (periodic cyclical function).

Note: The original *N-BEATS* implementation only works with univariate time series.

DeepAR

A novel time series model that combines both deep-learning and autoregressive

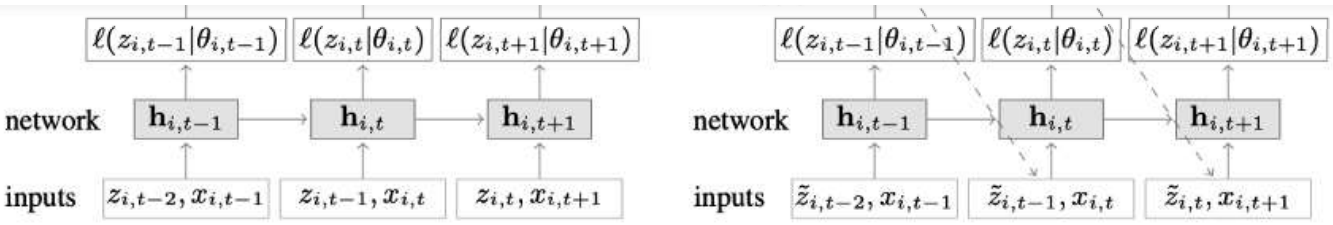



Figure 2: DeepAR model architecture (Source)

These are the model's key advantages:

Multiple time series: *DeepAR* works really well with multiple time series: A global model is built by using multiple time series with slightly different distributions. Also, this property finds application in many real world scenarios. For example, an electric power company might want to launch a power forecasting service for each of their customers. It is almost certain that each customer would have a different consumption pattern (which means a different distribution).

Rich set of inputs: Apart from historical data, *DeepAR* also allows the use of known future time sequences (a characteristic of auto-regressive models) and extra static attributes for series. In the aforementioned power demand forecasting scenario, an extra temporal variable could be the `month` (as an integer, with values between 1–12). Obviously, given that each customer is associated with a single sensor that measures power consumption, the extra static variable would be something like `sensor_id` or `customer_id` .

Automatic scaling: If you are familiar with time series forecasting using neural network architectures like MLPs and RNNs, one crucial preprocessing step is to scale the time sequence using a normalization or standardization technique. In *DeepAR*, there is no need to do that manually since the model under the hood scales the autoregressive input `z` of each time series `i` with a scaling factor `v_i` , which is simply the average value of that time series. Specifically, the equation of the scaling factor used in the paper's benchmarks is the following:

$$t_0 \sum_{t=1}$$

In practice however, if the magnitude of the target time series differs significantly, then it may help if you apply your our own scaling as well during preprocessing. For instance, in the energy demand forecasting scenario, the dataset could contain medium-voltage electricity customers (e.g. small factories, consuming power in the order of MW's) and low-voltage customers (e.g. households, consuming power in the order of KW's).

Probabilistic forecasting: *DeepAR* makes probabilistic forecasts instead of directly outputting the future values. This is done in the form of Monte Carlo samples. Also, these forecasts are used to compute quantile forecasts, by using the **quantile loss function**. For those who are not familiar with this type of loss, quantile loss is used to calculate not only an estimate, but also a prediction interval around that value.

Spacetimeformer

In the context of univariate time sequences, temporal dependencies are all that matters. However, in a multiple time series scenario, things are not so simple. For instance, let's say we have a weather forecasting task, where we want to predict the temperature of five cities. Also, let's assume that these cities belong to a a single country. Given what we have seen so far, we could use *DeepAR* and model each city as an external static covariate.

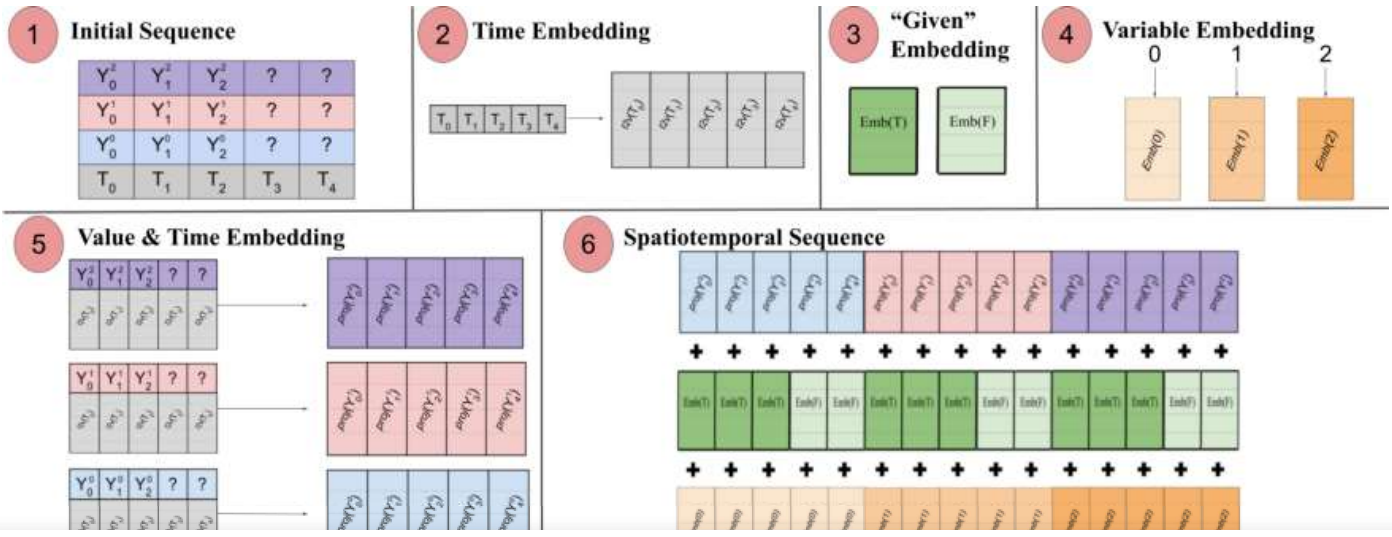
> In other words, the model would consider both temporal and spatial relationships. This is the core idea of Spacetimeformer.

We could also take things one step further and use a model that leverages the spatial relationships between these cities/locations in order to learn extra helpful dependencies. In other words, the model would consider both temporal and spatial relationships. This is

As its name implies, this model uses a transformer-based structure under the hood. In time series forecasting with transformer-based models, a popular technique to produce time-aware embeddings is to pass the input through a ***Time2Vec*** [6] embedding layer (As a reminder, for NLP tasks, a positional encoding vector is used instead of *Time2vec* that produces context-aware embeddings). While this technique works really well for univariate time sequences, it doesn't make any sense for multivariate time inputs. In language modeling, each word of a sentence is represented by an embedding, and a word is essentially a **concept**, part of a vocabulary.

In a multivariate time series context, at a given timestep `t`, the input has the form `x_1,t`, `x_2,t`, `x_m,t` where `x_i,t` is the numerical value of feature `i` and `m` is the total number of features/sequences. If we pass the input through a *Time2Vec* layer, a time embedding vector will be produced. In that case, what does this embedding really represent? The answer is it will represent the whole set of input as a single entity(token). Consequently, the model will learn only the temporal dynamics amongst timesteps, but will miss the spatial relationships among features/variables.

Spacetimeformer addresses this issue by flattening the input into a single large vector, called **spatiotemporal sequence.** If the input consists of `N` variables, organized into `T` timesteps, the generated spatiotemporal sequence will have (`N` x `T`) tokens. This is better displayed in **Figure** 3:



frequency embedding that represents periodic input patterns. (3) A binary embedding indicates whether this value is given as context or needs to be predicted. (4) The integer index of each time series is mapped to a "spatial" representation with a lookup-table embedding. (5) The Time2Vec embedding and variable values of each time series are projected with a feed-forward layer. (6) Value&Time, Variable, and Given embeddings are summed and laid out such that MSA attends to relationships across both time and variable space at the cost of a longer input sequence." Source

In other words, the final sequence has encoded a unified embedding, consisting of temporal, spatial and contextual information.

One drawback of this approach however is that the sequences can become too long, leading to a quadratic increase of resources. This is because according to the attention mechanism, every token/entity is checked against one another. The authors make use of a more efficient architecture suitable for larger sequences, called Performer attention mechanism. For more technical details, check the project's repo on Github.

Temporal Fusion Transformer

Temporal Fusion Transformer (TFT) is a transformer-based time series forecasting model published by Google. If you would like a more thorough analysis regarding this awesome model, check this post.

> TFT is more versatile than the previous models. For instance, DeepAR does not work with time-dependent features that are known only up to the present.

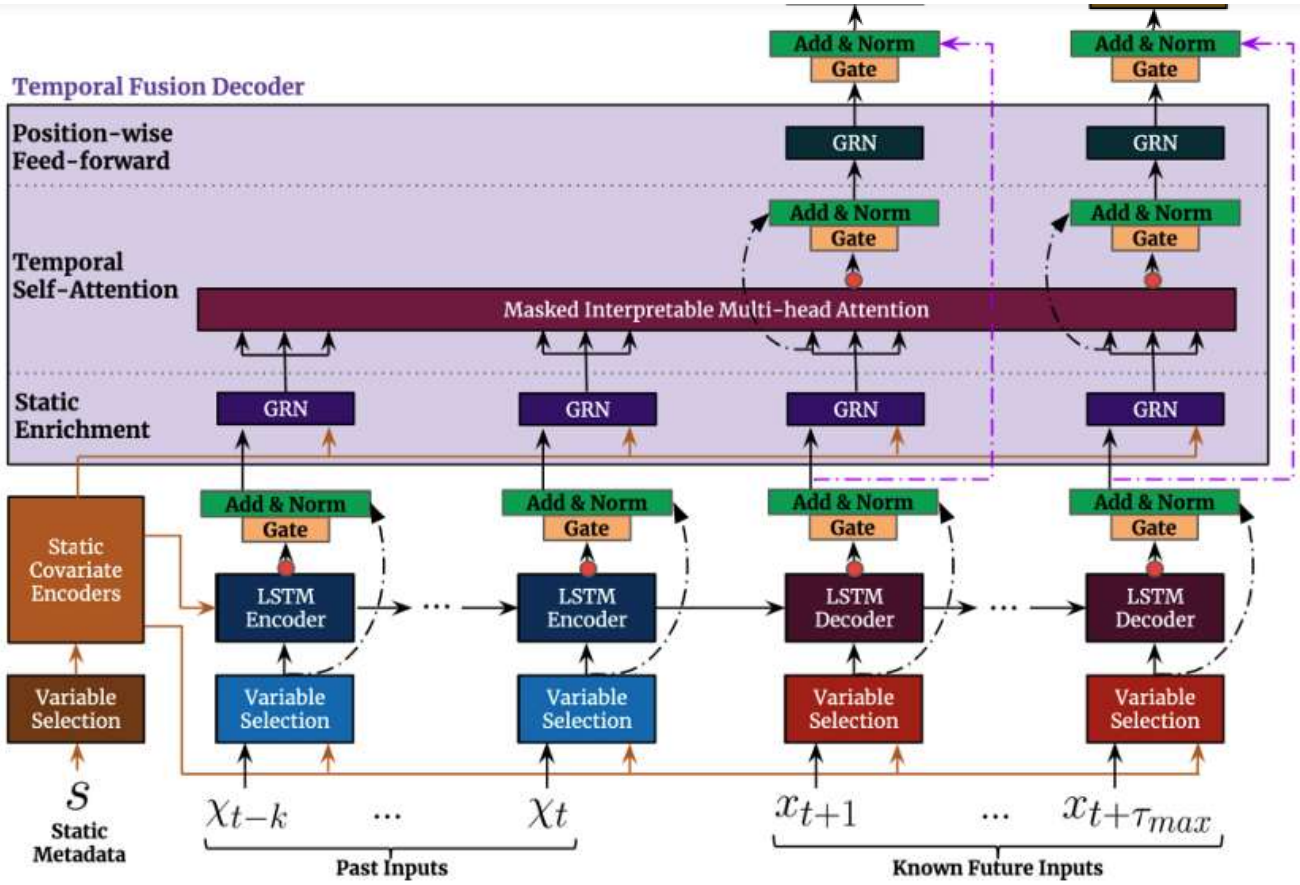


Figure 4: Top level architecture of TFT, along with its main components (Source)

The top level architecture of *TFT* is shown in **Figure 4.** These are the model's key advantages:

- **Multiple time series**: Like the aforementioned models, *TFT* supports building a model on multiple, heterogeneous time series.
- **Rich number of features**: *TFT* supports 3 types of features: i) time-dependent data with known inputs into the future ii) time-dependent data known only up to the present and iii) categorical/static variables, also known as **time-invariant** features. As a consequence, *TFT* is more versatile than the previous models. For instance, *DeepAR* does not work with time-dependent features that are known only up to the present. In the aforementioned power demand forecasting scenario, we would like to use humidity levels as a time-dependent feature, which is known only until up to the present. **This is feasible in *TFT* but not in *DeepAR*.**

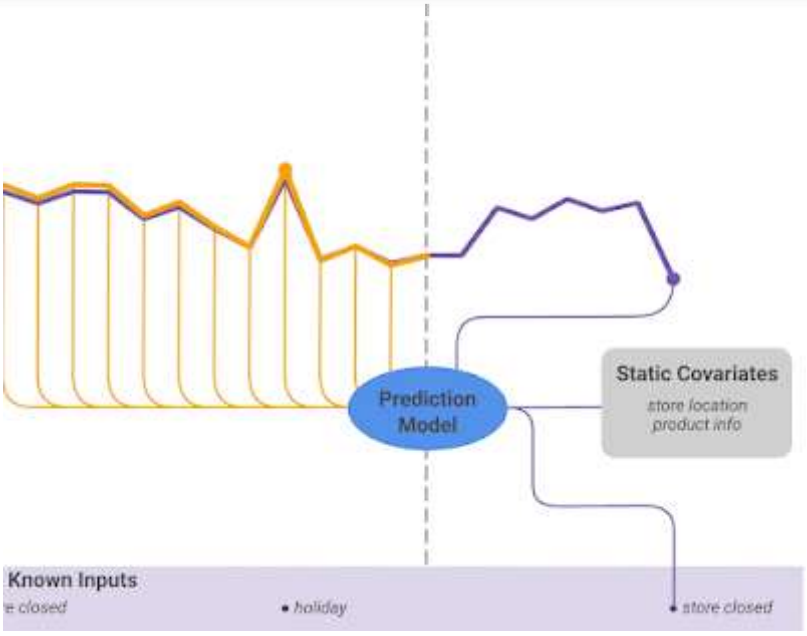


Figure 5: Effect of external static variables on forecasting (Source)

- **Interpretability**: *TFT* gives much emphasis on interpretability. Specifically, by taking advantage of the **Variable Selection** component (shown in **Figure 4**), the model can successfully measure the impact of each feature. As a consequence, the model learns the feature importances.
  On the other hand, *TFT* proposes a novel **interpretable Multi-Head Attention mechanism:** The attention weights from this layer can reveal which time-steps during the lookback period are the most important. Therefore, visualization of those weights could reveal the most prominent seasonal patterns throughout the whole dataset.

- **Prediction Intervals:** Similar to *DeepAR*, *TFT* outputs a prediction interval along with the predicted values, by using quantile regression.

Closing Remarks

Taking all the above into consideration, Deep Learning has undoubtedly revolutionized the landscape of time series forecasting. All of the aforementioned models, apart from unparalleled performance, have one common denominator: They make the best of multiple, multivariate temporal data, while simultaneously they use exogenous information in a harmonic way that boosts forecasting performance to unprecedented

References

[1] Makridakis et al., The M5 Accuracy competition: Results, findings and conclusions, (2020)

[2] D. Salinas et al., DeepAR: Probabilistic forecasting with autoregressive recurrent networks, International Journal of Forecasting (2019).

[3] Boris N. et al., N-BEATS: Neural Basis Expansion Analysis For Interpretable Time Series Forecasting, ICLR (2020)

[4] Jake Grigsby et al., Long-Range Transformers for Dynamic Spatiotemporal Forecasting,

[5] Bryan Lim et al., Temporal Fusion Transformers for Interpretable Multi-horizon Time Series Forecasting, International Journal of Forecasting September 2020

[6] Seyed Mehran Kazemi et al.,*Time2Vec: Learning a Vector Representation of Time*, July 2019