Model Report: Patch-TST for Time-Series Demand Forecasting

Uday Sapra, Data Science Intern, Promo Analytics A&BC

I. Background:

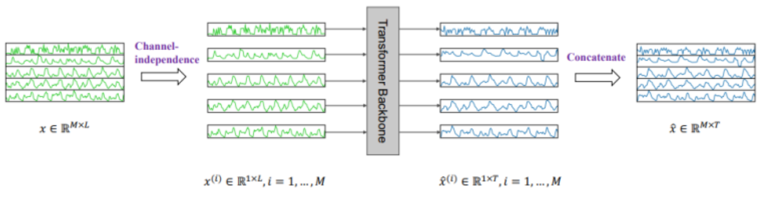
Promo Analytics is currently developing an ensemble of models for front-store demand forecasting with both forest and transformer-based models. While the ensemble undergoes testing before deployment, I was tasked to prototype a new transformers-based model to add to the ensemble. The purpose of this model was to fix two issues inherent to the traditional transformer architecture – slow training speeds due to compute scaling quadratically with context length and poor-performance on time-series that need long context lengths i.e. seasonal data. If performance expectations were met, the model would either become a complement to the ensemble or entirely replace the current transformers-based model.

II. Project Goals:

1. Faster Training Speeds than TFT
2. Par Overall Performance to TFT
3. Better Seasonal Data performance against TFT

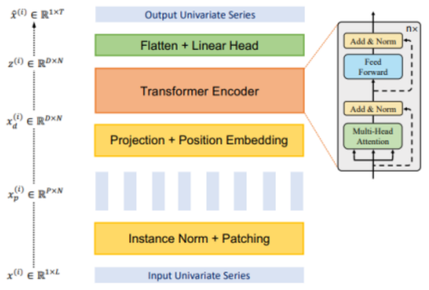
III. Model Choice: Patch-TST

The team was exploring Patch-TST as a solution to these issues due to two major modifications it makes to the transformers architecture:



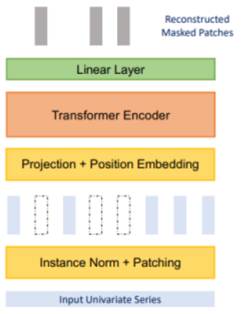
$x \in \mathbb{R}^{M\times L}$
Channel-independence
$x^{(i)} \in \mathbb{R}^{1\times L}, i = 1, \dots, M$
Transformer Backbone
$\hat{x}^{(i)} \in \mathbb{R}^{1\times T}, i = 1, \dots, M$
Concatenate
$\hat{x} \in \mathbb{R}^{M\times T}$

(a) PatchTST Model Overview



$\hat{x}^{(i)} \in \mathbb{R}^{1\times T}$
$z^{(i)} \in \mathbb{R}^{D\times N}$
$x_d^{(i)} \in \mathbb{R}^{D\times N}$
$x_p^{(i)} \in \mathbb{R}^{P\times N}$
$x^{(i)} \in \mathbb{R}^{1\times L}$
Output Univariate Series
Flatten + Linear Head
Transformer Encoder
Projection + Position Embedding
Instance Norm + Patching
Input Univariate Series
n×
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention

(b) Transformer Backbone (Supervised)



(c) Transformer Backbone (Self-supervised)

1. <u>Patching of Input Time-Series</u>

Transformers based models are applied for modelling sequential datasets. First developed for NLP tasks, they were later adopted on other forms of sequential data – like time-series. Traditional transformers encode and process each input token, this is known as point-wise attention, and this enables the model to understand and remember relationships between input tokens and give more attention to stronger relationships. While point-wise attention makes intuitive sense for words, which carry meaning by themselves, it doesn't necessarily extend to time-series. Each time-step doesn't convey a lot by itself, and more context is required for understanding and modelling time-series datasets. Point-wise attention doesn't only prompt a logical conflict when transitioning to time-series tasks, it also leads to computational bottlenecks. The time complexity of transformers-based models scales quadratically $O(N^2)$ with respect to the look-back window of the model or the number of input tokens. For time-series models in general, longer lookback windows are preferred as they relay more information to the model, enabling effective training. Loss for time-series tasks thus, generally, scales inversely to the look-back window. The quadratic increase in compute becomes a bottleneck on time-series' that rely heavily on longer contexts to be understood, especially seasonal data. Long term seasonalities – annual or half-yearly – would intuitively need more context and thus longer look-back windows to be forecasted with sufficient accuracy.

PatchTST modifies the transformers architecture to introduce patching – combining sequential time-steps together to form a longer chunk of time-steps, called a patch. These patches are then processed by the model as input tokens. So instead of time-stamp-wise attention, the model processes the data patch-wise. Such a modification has two significant benefits. Firstly, it enables each input token to hold more semantic information as it is a collection of sequential time-steps. Secondly, it scales down the total number of input tokens as a multiple of patch-size and stride, leading to a quadratic reduction in time-complexity. Thus, with patching, the transformer receives more meaningful input tokens, can harness more context through larger lookback windows, all of which is achieved without significant increases in compute requirements. In theory, this would enable us to train the model way faster while improving performance on seasonal categories – a limitation of the current TFT.

2. <u>Channel Independence</u>

For multivariate time-series data, traditional transformer-based models mix information from input time-series during training as they share an identical set of attention weights across the model. PatchTST on the other hand, processes each time-series independently, by having a different set of attention weights for each. This, in theory, enables the model to better preserve and and understand local semantic information unique to each input time-series. Channel independence does align with our use case, as each product category has multiple products, the sales for which may not necessarily related. Hence, processing each product independently should lead to more accurate predictions for each product, and hence the entire category. While this limits the model's ability to understand and utilize correlations between different time-series for predictions, our ensemble plans to incorporate that information through Graph Neural Networks.

Conclusively, with these two changes – patching and channel independence - to the vanilla transformer architecture, PatchTST promises both faster train times and improved performance on seasonal data. Given the model's alignment with our use-case, I was tasked to prototype it.

IV. Model Development

1. **Libraries, Tools, and Specs:** The model was developed using the TSAI (Time-Series AI) library and its implementation of PatchTST. FastAI/TSAI's built-in functions were used for most preprocessing steps, including train-test splits, and sliding-window transformations, and normalization. Optuna was used to fine-tune hyperparameters. All models were trained with 1 NVIDIA Tesla V100 16GB GPU.

2. **Dataset:** Oral Hygiene (Adversion B) was used to train the model and benchmark against the TFT.

3. **Forecasting Horizon and Use-Case:** These were kept identical to the TFT for benchmarking and due to business needs.

4. **Models:** Two versions of PatchTST were developed for testing – one to predict a single Joint-ID. The other to simultaneously forecast for all Joint-IDs in a category.

V. Model Limitations

Early in model development, I encountered two major limitations for the model.

1. **Ignoring Exogenous Variables**: Unlike the TFT, PatchTST does not allow for exogenous variables – either static or time-series. This meant we could not train PatchTST with the feature-set we used for the TFT and relied solely on auto-correlation for modeling. Given PatchTST is only designed for multi-variate time-series tasks, we could not have a single target variable and required multiple target outputs. Conclusively, the only logical way to run the model was to input all Joint-IDs for a product category at once and make the model output predictions for all.

2. **Library Limitations:** The TSAI library had poor documentation for PatchTST and its hyperparameters. Specifically, were not able to train the model on a custom-loss function and were limited to functions that TSAI implemented, which were MAPE and MSE.

VI. Model Results

Despite the promising theory, the model's results were below expectations. While the train-times showed significant improvements due to patching and a reduced feature-set, the overall accuracy of the model was not on par with the TFT – the loss doubled. Below are the results from the best Optuna tuned models.

Model	Rolling SMAPE	Rolling SMAPE Weighted by SCAN-QTY	Train Time/Epoch	Train Time/Total
PatchTS Oral Hygeine	60%	32%	**6 seconds**	**90 seconds**
PatchTST Single-Joint-ID	16%	N/A	**<1 Second**	**<10 seconds**
TFT Oral Hygeine	**37%**	**33%**	**15 minutes**	**>2 hours**

VII. Explanation of Limitations

There are 3 major reasons I suspect why the model performance was subpar.

1. **Assumption of Autocorrelation**: We lose a lot of useful information by not being able to utilize the exogenous features that the TFT and Boosting models incorporate. By simply relying on autocorrelation, we ignore the proven impact exogenous variables have on product sales and lose a significant amount of predictive power. Thus, with the lack of 26 other exogenous variables, Patch-TST's performance could not converge with the current models in the ensemble.

2. **High Model Complexity**: Given channel-independence assigns a different set of attention weights to each input time-series, the model when run to predict hundreds of Joint-IDs for a product category, becomes immensely complex. This increase in complexity is met with a severe reduction in input tokens due to patching. With only 155 weeks of total data, we do not have enough information to sufficiently train the model's parameters – this was observed by very unstable training and high overfit.

3. **Poor Modelling of Inter-Relationships**: Patching and Channel Independence isolates each input time-series from the others, forcing the model to treat them as independent and unrelated sets of data. Due to this, the model becomes blind to inter-relations between different input time-series. The TFT, on the other hand, shares attention weights across all time-series, and is able to discover correlations between different Joint-IDs in a category, and utilize them for enhanced predictive power.

VIII. Going Forward

The only reason to revisit PatchTST would be to harness its quick training times for recurring and fast baseline predictions. Given that, here are some ideas that could be implemented to make the model perform better.

1. **Change Data Granularity**: With > 100,000 parameters to train depending on hyperparameters, the model demands more data to sufficiently fit and currently we have an insufficient number of input tokens after patching. Hence, changing the granularity of input time-series from weekly to daily might give the model more data to sufficiently train weights against. This would yield a greater number of input patches that are also intuitively more meaningful – patches as a set of days instead of a currently, a set of weeks. For our use case, the benefits of patching might only surface when we have sufficient data.

2. **Custom Loss Function:** The authors of the paper only used MSE for PatchTST, while the TSAI library only implements MSE and MAPE. For our data, MSE has not performed well on the transformers based TFT models and that nature could be inherited by PatchTST. MAPE is unstable when the targets are close to 0 and thus unideal for our use-case. SMAPE and Weighted SMAPE gave optimal performance for the TFT, but I was unable to implement those on PatchTST due to limitations of the TSAI library. In the future, if the TSAI library is updated or if a more main-stream library implements PatchTST, SMAPE and Weighted SMAPE could be tested to assess improvements.

IX. Project Summary

We have ruled out PatchTST as a potential addition to the ensemble. While the training speeds are exponentially better, the drop in performance is not acceptable for our business use-case. Improvements to the TFT and research into more advanced Time-Series forecasting methods such as Mamba would be ideal next steps.