HYBRID TRANSFORMER AND HOLT-WINTER'S METHOD FOR TIME SERIES FORECASTING

Nhi N. Truong[1], **Duc Q. Nguyen**[1], **Jeffrey Gropp**[2], **Sang T. Truong**[2, 3]
[1]HCMC University of Technology - VNU-HCM [2]DePauw University [3]Stanford University
sttruong@cs.stanford.edu

ABSTRACT

Time series forecasting is an important research topic in machine learning due to its prevalence in social and scientific applications. Multi-model forecasting paradigm, including model hybridization and model combination, is shown to be more effective than single-model forecasting in the M4 competition. In this study, we hybridize exponential smoothing with transformer architecture to capture both levels and seasonal patterns while exploiting the complex non-linear trend in time series data. We show that our model can capture complex trends and seasonal patterns with moderately improvement in comparison to the state-of-the-arts result from the M4 competition.

INTRODUCTION

Time series forecasting spans various fields like medicine and economics, employing methodologies such as the transformer architecture for Influenza-like illness (ILI) prediction (Wu et al., 2020a) and the hybridization of autoregressive integrated moving average (ARIMA) and artificial neural network (ANN) for currency exchange rate forecasting (Khashei & Bijari, 2011). These methods fall into categories like linear or non-linear, parametric or non-parametric, statistical or machine learning, and single or multi-model approaches, each extracting different insights from data. The M4 competition (Makridakis et al., 2020) demonstrates the superiority of multi-model frameworks over single-model ones. Two prevalent multi-model forecasting paradigms are model combination and hybridization (Makridakis et al., 2018). Model combination constructs a panel of models, leveraging a linear combination of their predictions. Conversely, hybridization passes data through a sequence of models, such as the exponential smoothing - recurrent neural network (ESRNN), which excelled in the M4 competition (Smyl, 2020), capturing levels, seasonality, and trends effectively.

Although RNNs are integrated into state-of-the-art model combinations, they come with significant drawbacks. Firstly, the sequential nature of RNNs makes it challenging for them to effectively learn long-term dependencies (Greaves-Tunnell & Harchaoui, 2019). Additionally, RNNs often encounter numerical instability issues. Due to the repeated multiplication of weights by values at different time points, gradients can either vanish or explode, making it difficult to update the weights effectively for optimal results (Ribeiro et al., 2020). Finally, since RNN computes one-by-one values through unrolled loop, the training and predicting process cannot be parallelized hence much slower.

The multi-head attention transformer architecture (Vaswani et al., 2017) aims to address the limitations of RNNs. This transformer model employs a multi-head self-attention mechanism to handle sequence data, allowing it to learn dependencies across sequences simultaneously and identify the most pertinent data points for generating outputs. Therefore, the training and prediction of this architecture is more efficient without numeric instability problems.

In this paper, we hybridize multi-head attention transformer architecture and the Holt-Winter's method to improve the state-of-the-arts forecasting method. Our model captures the pattern of levels and seasonality, while exploits the nonlinear trend via the neural autoencoder. We conduct experiments and ablation study on M4 to compare our model with various baselines.

RELATED WORKS

Exponential smoothing - recurrent neural network (ESRNN) is the state-of-the-art (SOTA) hybrid model which is fast and applicable to different time series dataset. This model composes two dif-

ferent layers which is the pre-processing for exponential smoothing and a Long-short term memory (LTSM) layer that updates the parameters of Holt-Winter model through each series.

Pre-processing layer Following the application of Holt-Winters' multiplicative trend and seasonality as described earlier, the pre-processing step removes seasonality and normalizes the series to derive the trend for the recurrent neural network layer. This is achieved through the calculation of $b_t = \frac{y_t}{l_t s_t}$, where y_t represents the input vector, l_t signifies the level, s_t denotes the seasonality, and b_t represents a vector of trend.

Deep Learning layer The neural network structure, as outlined in (Smyl, 2020), integrates LSTM layers with skip connections to create the Dilated LSTM network. This design enhances computational efficiency and enables the retention of information across multiple past time intervals. By introducing dilation in the second layer, the initial LSTM hidden weights, input to successive cells, and bias weights can extend to two consecutive cells. Moreover, residual connections from the RNN layers contribute to stable training, while a straightforward linear layer at the conclusion assists in aligning RNN output with the residual prediction window in the normalized and deseasonalized data. A notable feature of this architecture is its ability to simultaneously train both RNN and classical Holt-Winters parameters.

Hybrid model Combining multiple models has been employed across various time series datasets, as seen in the integration of models like ARIMA-ANN and ESRNN (Khashei & Bijari, 2011; Rahimi & Khashei, 2018; Singh, 2015). One approach involves amalgamating two distinct neural network models to discern intricate patterns within multivariate time series data (Wu et al., 2020b). Another common strategy is to employ a statistical model to detect linearity within the time series data, followed by the utilization of a neural network to capture non-linear patterns (Khashei & Bijari, 2011; Smyl, 2020). (Smyl, 2020) demonstrated the effectiveness of this hybrid ESRNN model across various types of series, including those with high levels of randomness.

METHOD

Let $w \in \mathbb{R}$ be the window size from time step t to time step $t + w$ (Hota et al., 2017). Our method is to select the next segment from the end of the last segment. We repeat this process after going through all data points within a series. The input matrix for our model is $Y_{t,t+w} \in \mathbb{R}^{|D| \times w}$ which contains $|D|$ series with the length of each series is w (i.e. from time t to $t + w$).

EXPONENTIAL SMOOTHING DECOMPOSITION

There are nine types of exponential smoothing (ES) models depending on variations in the combinations of trend and seasonality, as outlined by (Hyndman & Athanasopoulos, 2018). (Smyl, 2020) adopts the multiplicative seasonality approach due to the widening seasonal pattern observed in M4 data across various time frames (Hyndman & Athanasopoulos, 2018; Makridakis et al., 2020). The Holt-Winter's multiplicative ES model are specified as $\hat{y}_{t+h} = l_t b_t^h s_{t-m+h}$ with

$$l_t = \alpha \frac{y_t}{s_{t-m}} + (1-\alpha) l_{t-1} b_{t-1}; b_t = \beta \frac{l_t}{l_{t-1}} + (1-\beta) b_{t-1}; s_t = \gamma \frac{y_t}{l_{t-1} b_{t-1}} + (1-\gamma) s_{t-m}, \quad (1)$$

where l_t, b_t, and s_t are the levels, trends, and seasonality at time t with the corresponding smoothing parameters α, β, and γ, respectively. All smoothing parameters have the inclusive interval between 0 and 1. $\hat{y}_{t+h}$ is the predicted value at time $t+h$ where t is the current time frame that we use to analyze and h is the forecast horizon. m is the windows size of the series. Decomposition layer extracts the seasonality and levels from our series using multiplicative seasonality with no trend:

$$L_{t,t+w} = \alpha \left(\frac{Y_{t,t+w}}{\prod_i S^i_{t-s,t+w-s}} \right) + (1-\alpha) L_{t-1,t+w-1}; S^i_{t,t+w} = \gamma (\frac{Y_{t,t+w}}{L_{t,t+w}}) + (1-\gamma) S^i_{t-s,t+w-s} \quad (2)$$

where $L_{t,t+w} \in R^{|D| \times w}$ is the levels matrix, $S^i_t \in R^{|D| \times w}$ is the seasonality matrix in the seasonality set S. $\alpha \in [0,1]^{|D|}$ and $\gamma \in [0,1]^{|D|}$ are the smoothing vectors parameters of $L_{t,t+w}$ and $S^i_{t,t+w}$ respectively. α goes through a sigmoid layer, while γ is through an exponential layer.

In this ES layer, since the model no longer has local linear trend, we take out the levels and seasonality from our truth data to obtain the in-sample prediction $\hat{Y}_{t,t+w} \in \mathbb{R}^{|D| \times (w+|M|)}$ as follows.

$$\hat{Y}_{t,t+w} = \frac{Y_{t,t+w}}{L_{t,t+w} \odot \prod_i S^i_{t,t+w}}, \quad (3)$$

where $\odot$ denotes element-wise multiplication. We concatenate the residual trend and five binary variables indicating the domain $m \in M$ (e.g. macro or finance) from $\mathcal{D}$. Then, the non-linearity residual passes through the transformer for next training process.

TRANSFORMER

The effectiveness of the Transformer architecture in sequence modeling has been demonstrated in various studies (Li et al., 2019; Wu et al., 2020a;b). In the realm of time series forecasting, (Wu et al., 2020a) utilize a Transformer architecture, which outperforms the current SOTA models in predicting ILI. Our model follows the multi-head attention Transformer architecture (Vaswani et al., 2017), with encoder-decoder as the primary layers and multi-head scale dot-product attention as the sublayer. Additionally, we employ a linear transformation with sigmoid activation instead of the softmax function, which is more suitable for time series forecasting tasks. Let $TFM()$ denotes our transformer layer, the final prediction $\hat{Y}^{out}_{t,t+w} \in \mathbb{R}^{|D| \times d_{output}}$ is obtained by $\hat{Y}^{out}_{t,t+w} = TFM(\hat{Y}_{t,t+w})$ where d_{output} is the dimension of output vector and depends on how much steps we want to predict. Details of the transformer is presented in Appendix *Transformer layer*.

EXPERIMENTAL RESULTS AND DISCUSSION

THE DATASET

The M4 competition dataset, sourced from the ForeDeCk database at the National Technical University of Athens (Makridakis et al., 2020), originates from a comprehensive database featuring 900,000 series for business forecasting collected from various sectors such as education and government. From this extensive database, the M4 competition selected a subset of 100,000 series randomly, spanning six different frequencies (Hourly, Daily, Weekly, Monthly, Quarterly, and Yearly) and six diverse domains (Micro, Industry, Macro, Finance, Demographic, and Other). Details regarding the number of observations are provided in Table 2, while Table 3 presents a summary statistic of the dataset.

EVALUATION METRICS

Similar to the M4 competition (Makridakis et al., 2018) which use predictions from the Naive2 model as a baseline model, we use the Overall Weighted Average (OWA) between ESTransformer and Naive2 as the baseline model. The OWA is constructed as:

$$\text{OWA} = \frac{1}{2}\left[\frac{\text{sMAPE}}{\text{sMAPE}_{\text{Naive2}}} + \frac{\text{MASE}}{\text{MASE}_{\text{Naive2}}}\right], \tag{4}$$

where Symmetric Mean Absolute Percent Error (sMAPE) and Mean Absolute Scaled Error (MASE) are represented as:

$$\text{sMAPE} = \frac{200}{h}\sum_{i=1}^{h}\frac{|y_{t+i} - \hat{y}_{t+i}|}{|y_{t+i}| + |\hat{y}_{t+i}|}; \text{MASE} = \frac{1}{h}\sum_{i=1}^{h}\frac{|y_{t+i} - \hat{y}_{t+i}|}{\frac{1}{t+h-m}\sum_{j=m+q}^{T+H}|y_j - y_{j-m}|}. \tag{5}$$

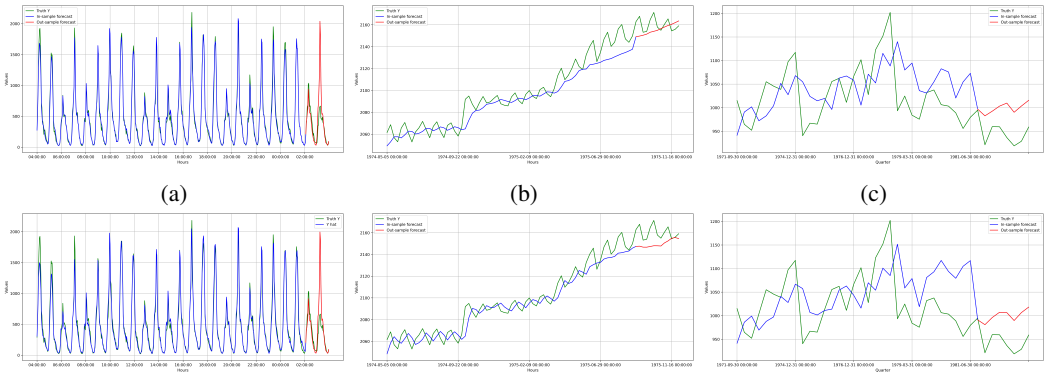
RESULTS AND DISCUSSION

The results of ESTransformer and ESRNN are reported in Table 1. From all frequencies data' s OWA, ESTransformer has comparable result with ESRNN with better performance in the Daily and Quarterly data. Moreover, four out of six frequency in M4 data (i.e. Hourly, Weekly, Monthly, and Yearly) has lower OWA value after training in ESRNN.

We depict the training and prediction results of three series in Figure 1. For Hourly series no.149 (Figure 1a and 1d), both ESTransformer and ESRNN perform well, closely resembling the testing data due to the series' strong cyclic pattern. However, for Weekly series no.106 (Figure 1b and 1e), ESRNN over-forecasts while ESTransformer under-forecasts. Despite the sophistication of ESRNN's predictions, they significantly underestimate the true values. In contrast, our model's out-of-sample predictions are closer to the actual data. Lastly, for Quarter series no.996 (Figure 1c and 1f), although both models' in-sample predictions are close to the actual values, ESTransformer's prediction is visibly closer. Overall, our ESTransformer architecture demonstrates comparable or superior performance to the state-of-the-art ESRNN model on the M4 dataset, as evidenced by the loss metrics reported in Table 1 and Table 5.

Table 1: OWA loss metrics' results for testing data.

Model	Frequency	Hyperparameter	OWA↓
ESTransformer	Hourly	$\|D\|$ = 80730, d_{model} = 128, N = 2, $\|M\|$ = 6, w = 24, d_{output} = 48, τ = 0.5, LP = 30	0.66
	Daily	$\|D\|$ = 4227	1.00
	Weekly	$\|D\|$ = 359, w = 10, d_{output} = 13, LP = 100	1.23
	Monthly	$\|D\|$ = 24000, w = 12, d_{output} = 18, LP = 50	0.90
	Quarterly	$\|D\|$ = 14400, w = 4, d_{output} = 8, LP = 100	0.90
	Yearly	$\|D\|$ = 23000, w = 4, d_{output} = 6, LP = 100	0.81
ESRNN	Hourly	$\|D\|$ = 80730, $\|M\|$ = 6, w = 24, τ = 0.5, LP = 30	0.53
	Daily	$\|D\|$ = 4227	1.01
	Weekly	$\|D\|$ = 359, w = 10, LP = 100	1.10
	Monthly	$\|D\|$ = 24000, w = 12, LP = 100	0.90
	Quarterly	$\|D\|$ = 14400, w = 4, LP = 100	0.92
	Yearly	$\|D\|$ = 23000, w = 4, LP = 100	0.79



(a) (b) (c)

(d) (e) (f)

Figure 1: Demonstration of ESTransformer (top - (a), (b), and (c)) and ESRNN (bottom - (d), (e), and (f)) on Hourly series no.149 (left - (a) and (d)), Weekly series no.106 (middle - (b) and (e)), and Quarterly series no.996 (right - (c) and (f)). The blue line represents the in-sample forecast. The red line represents the out-sample forecast. The green line represents the real input value.

Conclusion and Future Works

In this paper, we propose ESTransformer, a hybridization approach for time series forecasting. We utilize ES to extract the linear pattern and transformer architecture to explore the non-linearity pattern in time series data. We show that our model can capture complex trends and seasonal patterns with moderately improvement in comparison to the SOTA result from the M4 competition.

Various approaches can be explored to extend our work. For instance, instead of relying solely on statistical methods for preprocessing, hybridizing different non-linear learners can delve deeper into complex time series data patterns and elucidate variable impacts. As evidenced by (Guo et al., 2021), combining a multi-scale residual convolutional neural network with long short-term memory reveals intricate feature interactions, outperforming single neural network models. Investigating the rationale behind the superior performance of multi-model forecasting presents a compelling research avenue, especially considering the unclear reasons behind its success despite demonstrated effectiveness in competitions like M-competition. Additionally, addressing the absence of variance estimation in models like ESRNN and ESTransformer by developing a hybridization framework that provides confidence levels for predictions would be a significant contribution to the forecasting domain. Despite limited time and computational resources, our model achieves comparable or improved forecasting performance compared to ESRNN, signaling a modest yet foundational step towards enhancing forecasting capabilities within the research community.

REFERENCES

Alexander Greaves-Tunnell and Zaid Harchaoui. A statistical investigation of long memory in language and music. In Kamalika Chaudhuri and Ruslan Salakhutdinov (eds.), *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pp. 2394–2403. PMLR, 09–15 Jun 2019.

Qiutong Guo, Shun Lei, Qing Ye, and Zhiyang Fang. Mrc-lstm: A hybrid approach of multi-scale residual cnn and lstm to predict bitcoin price. In *2021 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8, 2021. doi: 10.1109/IJCNN52387.2021.9534453.

HS Hota, Richa Handa, and AK Shrivas. Time series data prediction using sliding window based rbf neural network. *International Journal of Computational Intelligence Research*, 13(5):1145–1156, 2017.

Rob J Hyndman and George Athanasopoulos. Forecasting: Principles and practice. *OTexts: Melbourne, Australia*, 2018.

Mehdi Khashei and Mehdi Bijari. A novel hybridization of artificial neural networks and arima models for time series forecasting. *Applied Soft Computing*, 11(2):2664–2675, 2011. ISSN 1568-4946. doi: 10.1016/j.asoc.2010.10.015. The Impact of Soft Computing for the Progress of Artificial Intelligence.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

Shiyang Li, Xiaoyong Jin, Yao Xuan, Xiyou Zhou, Wenhu Chen, Yu-Xiang Wang, and Xifeng Yan. Enhancing the locality and breaking the memory bottleneck of transformer on time series forecasting. In *Proceedings of the 33rd International Conference on Neural Information Processing Systems*, Red Hook, NY, USA, 2019. Curran Associates Inc.

Spyros Makridakis, Evangelos Spiliotis, and Vassilios Assimakopoulosc. The m-4 competition: Results, findings, conclusion and way forward. *International Journal of Forecasting*, 2018.

Spyros Makridakis, Evangelos Spiliotis, and Vassilios Assimakopoulos. The m4 competition: 100,000 time series and 61 forecasting methods. *International Journal of Forecasting*, 36(1): 54–74, 2020. ISSN 0169-2070. doi: 10.1016/j.ijforecast.2019.04.014. M4 Competition.

Zahra Haji Rahimi and Mehdi Khashei. A least squares-based parallel hybridization of statistical and intelligent models for time series forecasting. *Computers and Industrial Engineering*, 118: 44–53, 2018. ISSN 0360-8352. doi: 10.1016/j.cie.2018.02.023.

Andrew Redd, Kaung Khin, and Aldo Marini. Fast ES-RNN: A GPU implementation of the ES-RNN algorithm. *CoRR*, abs/1907.03329, 2019. URL http://arxiv.org/abs/1907.03329.

António H. Ribeiro, Koen Tiels, Luis A. Aguirre, and Thomas Schön. Beyond exploding and vanishing gradients: analysing rnn training using attractors and smoothness. In Silvia Chiappa and Roberto Calandra (eds.), *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics*, volume 108 of *Proceedings of Machine Learning Research*, pp. 2370–2380. PMLR, 26–28 Aug 2020.

Pritpal Singh. *Big Data Time Series Forecasting Model: A Fuzzy-Neuro Hybridize Approach*, pp. 55–72. Springer International Publishing, Cham, 2015. doi: 10.1007/978-3-319-16598-1_2.

Slawek Smyl. A hybrid method of exponential smoothing and recurrent neural networks for time series forecasting. *International Journal of Forecasting*, 36(1):75–85, 2020. ISSN 0169-2070. doi: 10.1016/j.ijforecast.2019.03.017. M4 Competition.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

Neo Wu, Bradley Green, Xue Ben, and Shawn O'Banion. Deep transformer models for time series forecasting: The influenza prevalence case. *arXiv preprint arXiv:2001.08317*, 2020a.

Sifan Wu, Xi Xiao, Qianggang Ding, Peilin Zhao, Ying Wei, and Junzhou Huang. Adversarial sparse transformer for time series forecasting. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 17105–17115. Curran Associates, Inc., 2020b.

TRANSFORMER LAYER

First, information about position of each token in the sequence needs to be encoded because transformer does not process token sequentially. Positional encoding of token x is

$$\mathrm{PE}(x) = \sin\left(\frac{2\pi \mathrm{x}}{\mathrm{period}}\right) \tag{6}$$

where period is a hyperparameter. Due to different lengths in various series, using this seasonal function can extrapolate the sequence and equalize the length of each series. Also, this positional encoding version is more suitable with time series data than the sine and cosine implemented in (Vaswani et al., 2017).

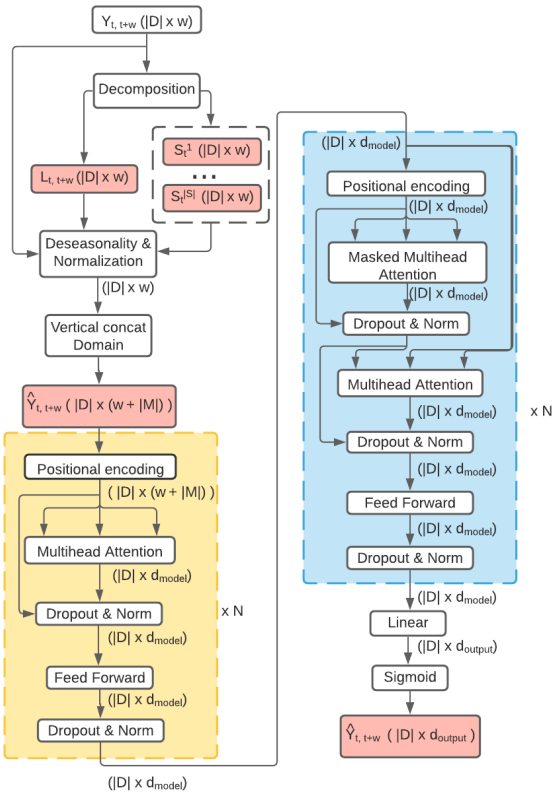


Figure 2: Architecture of the ESTransformer model. The left upper uncolor fraction represents the ES layer where the matrix levels L_t, seasonality S_t^i where i starts from 1 to $|S|$, and trend B_t at time t which are the outputs of this ES model. The yellow and blue fraction represents the encoder and decoder architecture of Transformer layer, respectively.

The next sub-layer in both encoder and decoder is the multi-head scale dot-product self-attention. A self-attention function finds the representation of the sequence through matrix of queries, keys, and values. It connects a query and a set of key-value pairs to an output (Vaswani et al., 2017). The compatibility between the query and the corresponding key will assign the weight to each value. The sum of these weights is the output.

There are two commonly used types of attention: the addictive and dot-product (i.e. multiplicative) functions. The additive attention computes the compatibility between the query and keys through a

feed-forward network with a hidden layer. On the other hand, the multiplicative attention applies the softmax function on the dot product of query with the corresponding keys. Even though these two attention mechanism have similar complexity, the multiplicative method is much faster and utilizes the memory more effectively due to its optimized matrix multiplication. Due to the efficiency in speed and allocating memory, we choose to implement the scale dot-product attention, a version of multiplicative self-attention with the scaling factor, in both encoder and decoder layer. As the keys of dimension grows larger, the magnitude of the dot product QK^T grows bigger, which causes the attention function to fall into extremely small or invalid gradient regions. To counteract against these regions, (Vaswani et al., 2017) scaled the dot-product by the keys of dimension $\sqrt{d_k}$.

The weight are computed through a set of queries, keys, and values packed together simultaneously into matrices Q, K, and V, respectively. We then apply the softmax function on the dot product of queries and keys, scaled by the keys of dimension $\sqrt{d_k}$ for each corresponding values:

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V \tag{7}$$

As the values of d_k gets larger, the magnitude of dot product grows larger. The softmax function may fall into regions where its gradients vanish. Thus, the scaling factor is to prevent attention function from the extremely small gradients.

The multi-head attention consists of different scale dot-product attention layers. At the same time, these attention computes the queries, keys and value with h times difference, learned the linear projections to the dimension d_k, d_k, and d_v, respectively. The result then is forwarded in the output dimension d_{model} and dimensional output values d_v. Different positions can also participate in this model at the same. The mechanism of multi-head attention can be represented as:

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, ..., \text{head}_h)W^O \tag{8}$$

$$\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V) \tag{9}$$

where $W_i^Q \in \mathbb{R}^{d_{model} \times d_k}$, $W_i^K \in \mathbb{R}^{d_{model} \times d_k}$, $W_i^V \in \mathbb{R}^{d_{model} \times d_v}$, and $W^O \in \mathbb{R}^{hd_v \times d_{model}}$. Different from (Vaswani et al., 2017), with M4 data, we use $h = 4$ which represents the number of parallel multi-head attentions layers. By default, we employ $d_k = d_v = h = 4$. Figure 3 demonstrates our multi-head attention architecture using scale dot-product attention.

After the multi-head self-attention, encoder and decoder contains a fully connected feed-forward network (FFN) sub-layer, which is also applicable for each position in a series. This sub-layer has two linear transformations with a ReLU activation in between:

$$\text{FFN} = W_1(\text{ReLU}(W_0(\hat{Y}_{t,t+w})))$$

The linear and ReLU transformations remains the same even if the positions change.

All of the components above are organized in two main layers: the encoder and decoder. Our first main layer in the transformer architecture is the encoder layer. It composes a stack of N identical layers. Each of them has two sub-layers. The multi-head attention is the first sub-layer, where it contains all the keys, values, and queries from the output of the positional encoding layer. Each position in the encoder computes the weights assigned to the corresponding positions in the series. The higher the weight implies stronger connections between these locations. The second sub-layer is a simple position-wise function which fully connects to the feed-forward network. Around these two sub-layer, we implement the residual connections, which are added to the dropout and normalization layer. To utilize these residual connections, all sub-layers and generic linear layers return the outputs with the dimension d_{model}.

Decoder architecture has many similarities to the encoder layer. Typically, the decoder as same number N layers. The residual connections are employed around these sub-layers, which is similar to the encoder architecture. But, one major difference is that the decoder has three sub-layers rather than two. The self-attention in the first sub-layer uses the masking method to prevent the positions from attending to subsequent positions. This technique combined with one offset position generated by generic linear output layer allows the predictions to function properly, based on the outputs at the previous positions. The second sub-layer is similar to the ones in encoder architecture. The third performs the multi-head attention on output from the encoder and the previous masked multi-head attention. In this "encoder-decoder attention" layer, only the queries come form the previous

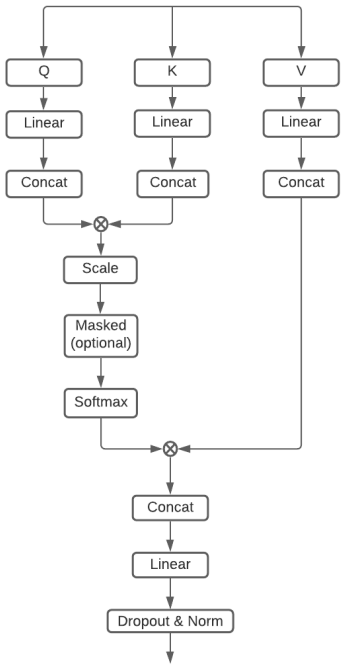


Figure 3: Multi-head attention with scale dot-product attention in Transformer. $\bigotimes$ denotes matrix multiplication. Scale layer is for the scaling factor by $\sqrt{d_k}$. The input is from the positional encoding layer or from the residual connection. The output continues to go through the dropout and normalized layer.

decoder layer, while memory keys and values are from the encoder output. Similarly to encoder, every positions in decoder layer is allowed to attend all positions, including that own position.

After the decoder layer, we add the linear and sigmoid transformation to convert the input and output to the matrix with dimension of output d_{ouput}. Also, we change the embeddings layer in (Vaswani et al., 2017) to linear transformation since our model is for the sequence of real numbers rather than a sequence of words. The output of this step is our residual trends after training through transformer with dimension ($|D| \times d_{output}$).

DATASET PRELIMINARY ANALYSIS

In this section, we provide an overview of the statistics pertaining to the M4 dataset, accompanied by preliminary observations gleaned from the data. Specifically, Table 2 and Table 3 delineate the quantity of samples and their corresponding summary statistics grouped by collecting frequency. Furthermore, Figure 4 and Figure 5 depict the outcomes of partial auto-correlation function (PACF), and multiplicative decomposition conducted on select samples within the dataset.

TRAINING DETAILS

We implemented all neural networks using the Python PyTorch 1.7.1 library. The training-testing ratio was 80:20. We use Adam Optimizer (Kingma & Ba, 2017) for ES and transformer architecture. Table 4 demonstrates the training configuration for ESTransformer. Other training parameters are set as default. More implementation details can be found at https://github.com/sangttruong/hybcast.

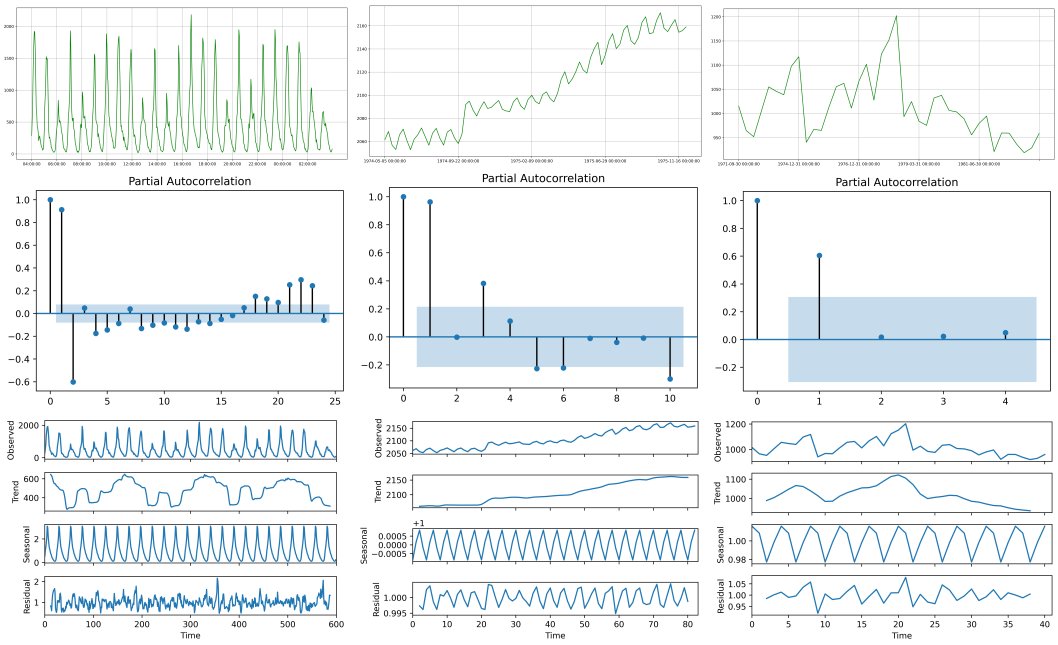
Our first intention for the training loss function is to use the sMAPE and MASE, which both are normalized absolute difference between actual and predicted values. Since the inputs to transformer has been through deseasonalization and normalization layers, the training loss does not need to normalize further. Therefore, we simply implement the difference between the target and predicted

Table 2: Number of samples in M4 data summary with no value less than 10 due to scaling effect. (Makridakis et al., 2020)

Frequency	Micro	Industry	Macro	Finance	Demographic	Other	Total
Yearly	6538	3716	3903	6519	1088	1236	23000
Quarterly	6020	4637	5315	5305	1858	865	24000
Monthly	10975	10017	10016	10987	5728	277	48000
Weekly	112	6	41	164	24	12	359
Daily	1476	422	127	1559	10	633	4227
Hourly	0	0	0	0	0	414	414
Total	25121	18798	19402	24534	8708	3437	**100000**

Table 3: Summary statistic of each frequency in M4 dataset.

Frequency	Count	Mean	Median	Std Dev	Min	Max
Yearly	720458	3689.73	2610.00	3173.55	22.10	115642.00
Quarterly	2214108	4240.60	3144.00	3469.55	19.50	82210.70
Monthly	10382411	4215.54	3250.00	3222.36	20.00	132731.32
Weekly	366912	3814.64	2676.86	3511.14	104.69	51410.00
Daily	9964658	5687.21	5232.20	4238.76	15.00	352000.00
Hourly	353500	5606.17	29.00	35495.30	10.00	703008.00

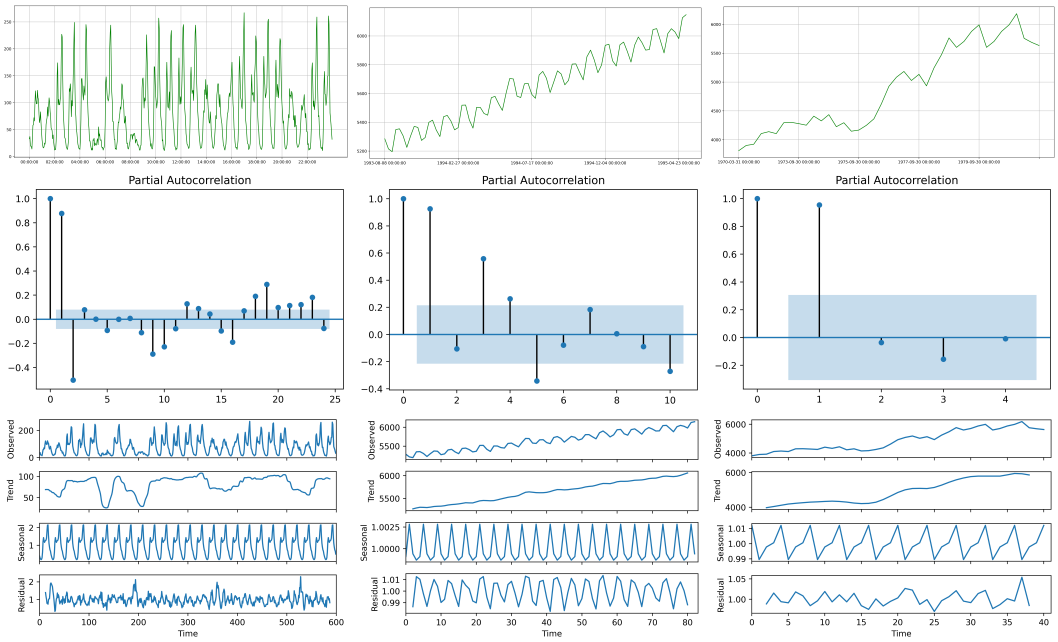


(a) (b) (c)

Figure 4: Preliminary analysis of (a) Hourly series no.149, (b) Weekly series no.106, and (c) Quarterly series no.996 from left to right

values. But, apparently, during backtesting, the positive bias appears due to squashing log function from the ES layer. Our transformer architecture learns from the log space, but our prediction needs to be in linear form. Therefore, for the training loss, we implement a pinball loss PB with the training percentile (τ), operated on the matrix:

$$PB = \begin{cases} \tau \times (Y_{t,t+w} - \hat{Y}_{t,t+w}), & \text{if } Y_{t,t+w} \geq \hat{Y}_{t,t+w}. \\ (1-\tau)(\hat{Y}_{t,t+w} - Y_{t,t+w}), & \text{otherwise.} \end{cases} \tag{10}$$



(a) (b) (c)

Figure 5: Preliminary analysis of (a) Hourly series no.373, (b) Weekly series no.254, and (c) Quarterly series no.8424 from left to right

Table 4: Training configuration

Frequency	Batch size	Learning rate	Schedule decay	Scheduler step size
Hourly	32	1e-2	0.5	7
Daily	64	1e-2	0.33	4
Weekly	32	1e-2	0.5	10
Monthly	64	7e-4	0.2	12
Quarterly	16	5e-4	0.5	10
Yearly	4	1e-4	0.1	10

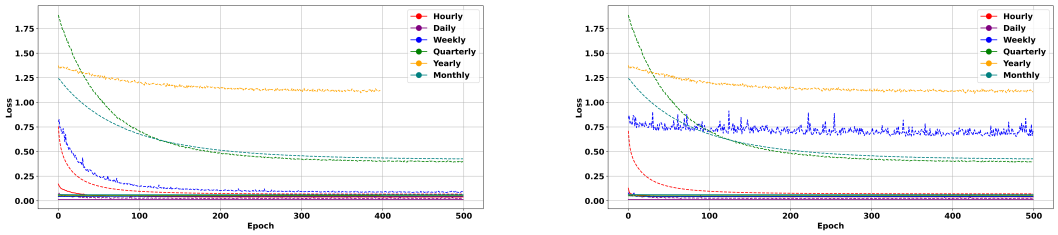
where τ has the inclusive interval between 0.1 to 0.65; the $Y_{t,t+N}$ and $\hat{Y}_{t,t+w}$ are calculated under the log of deseasonalized-normalized of real and predicted matrices. Both matrices have the same dimension which is $|D| \times w$. The purpose of training percentile is to prevent the linearity of the final forecast error. The Pinball loss takes the average $\overline{P}$ of each position's highest value in $\tau \times (Y_{t,t+w} - \hat{Y}_{t,t+w})$ and $(\tau - 1)(Y_{t,t+w} - \hat{Y}_{t,t+w})$ matrix which both use dot wise operation. Therefore, this pinball function can adjust for the penalty with different quantile, dealing with biases of various series efficiently.

The pinball loss from the training loss could have been implemented as testing loss as well. Nonetheless, the prediction intervals metric in the M4 competition is not based on different upper or lower pinball loss coverage percentage, but rather on the mean scaled interval score (Smyl, 2020; Makridakis et al., 2020). Also, even though the positive bias have been counteracted from the training loss, it can still happen if the upper interval exceed less frequently than the lower. Therefore, we design the testing loss aligned with the accuracy metrics in the M4 Competition, which is the Pinball loss divided by the levels variability penalty. The smoothness of the level plays an important part in forecasting accuracy. It also emphasizes the absorbance of seasonality on its components. Thus, we implement the levels variability loss LVL function from (Smyl, 2020) defined below:

$$LVL = ((\log L_{t+1,t+w+1} - \log Lt, t + w)^2)LP \tag{11}$$

where LP is the hyperparameter of level variability penalty which has the interval of 0 to 100, $L_{t+1,t+w+1}$ and $L_{t,t+w}$ are the levels matrices of each series. Similarly to the training loss, using dot wise operation on $\log L_{t+1,t+w+1} - \log L_{t,t+w}$, we square and take the average of each levels log differences matrix for each series. As (Smyl, 2020), the average and squares is a effective penalty computation in handling the wiggliness of a curve. Then, we divide the average of Pinball loss by the average of level variability loss $\overline{LVL}$ to obtain the testing loss:

$$TL = \frac{\overline{PB}}{\overline{LVL}} \tag{12}$$



(a) (b)

Figure 6: Training (dashed line) and Testing (straight line) loss of ESTransformer (a) and ESRNN (b)

ADDITIONAL RESULTS AND VISUALIZATIONS

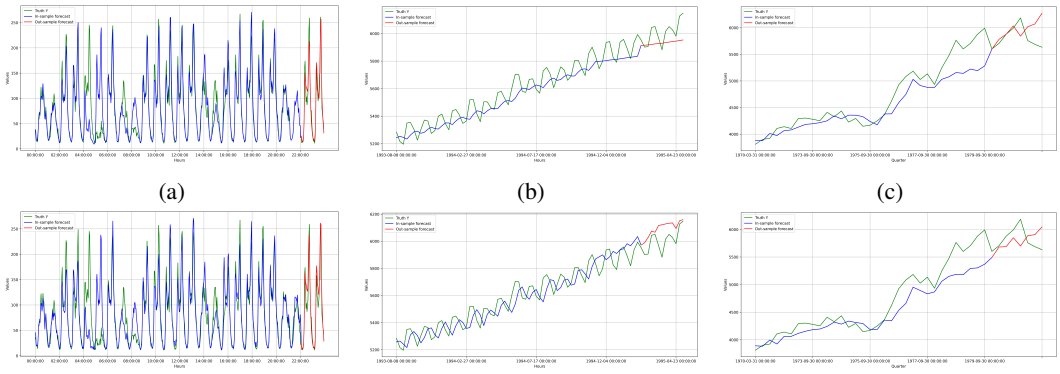
Additional train and test results are reported in the Table 5 which compared our ESTransformer with ESRNN (Redd et al., 2019). Figure 6 depicts the training and testing loss of these models, which all frequencies data converge after 300 epochs. The training and testing loss of ESTransformer are visibly similar to ESRNN, except for Weekly data. In Weekly data, the training process of ES-Transformer is smoother and suffers from less noise than ESRNN. The noisy loss surface of RNN architecture make it much more difficult for the optimizer to converge to a global optimum. The loss surface of Transformer is often much smoother, hence converge faster as shown in the Weekly data in this case. This is a major advantage of our framework in comparison to ESRNN. We also visualize additional prediciton results of ESTransformer and ESRNN in Figure 7.

Table 5: Training and testing loss

Model	**Frequency**	**Hyperparameter**	PB	TL
ESTransformer	Hourly	$\|D\|$ = 80730, d_{model} = 128, N = 2, $\|M\|$ = 6, w = 24, d_{output} = 48, τ = 0.5, LP = 30	0.06	0.03
	Daily	$\|D\|$ = 4227	0.02	0.01
	Weekly	$\|D\|$ = 359, w = 10, d_{output} = 13, LP = 100	0.08	0.04
	Monthly	$\|D\|$ = 24000, w = 12, d_{output} = 18, LP = 50	0.42	0.06
	Quarterly	$\|D\|$ = 14400, w = 4, d_{output} = 8, LP = 100	0.39	0.05
	Yearly	$\|D\|$ = 23000, w = 4, d_{output} = 6, LP = 100	0.39	0.05
ESRNN	Hourly	$\|D\|$ = 80730, $\|M\|$ = 6, w = 24, τ = 0.5, LP = 30	0.07	0.04
	Daily	$\|D\|$ = 4227	0.02	0.01
	Weekly	$\|D\|$ = 359, w = 10, LP = 100	0.71	0.04
	Monthly	$\|D\|$ = 24000, w = 12, LP = 100	0.45	0.07
	Quarterly	$\|D\|$ = 14400, w = 4, LP = 100	0.39	0.05
	Yearly	$\|D\|$ = 23000, w = 4, LP = 100	1.11	0.06

Residual diagnosis in training three data frequencies are shown in Figure 8. From Hourly to Yearly data, both models have similar residual diagnosis graphs. Although most standardized errors are centered around zero, some have extremely high value (e.g. from 30 to 40) which can be explained by the fact that some series in the dataset have weak or no patterns. For example, Quarterly series no.996 shows a random trend in Figure 4c. These complicated series also cause a remarkable

difference between the in-sample prediction and actual value. Quantile-Quantile plots demonstrate the deviation of the residual from the diagonal line, which implies both overestimated and underestimated prediction. This effect happens in both ESTransformer and ESRNN because of some irregular patterns from the data.



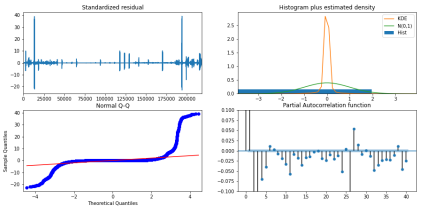
(a) (b) (c)

(d) (e) (f)

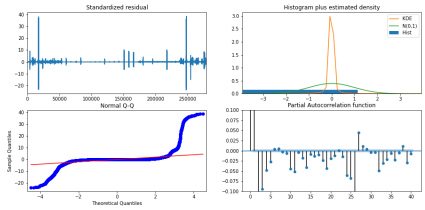
Figure 7: Demonstration of ESTransformer (top - (a), (b), and (c)) and ESRNN (bottom - (d), (e), and (f)) on Hourly series no.373 (left - (a) and (d)), Weekly series no.254 (middle - (b) and (e)), and Quarterly series no.8424 (right - (c) and (f))

We analyze the first layer in attention mechanism in six different frequencies to understand the prediction made by ESTransformer. Figure 9 demonstrates the attention weights of each positions [1]. The dark color represents weak dependency, whereas the bright color represents strong dependency. Commonly, these six frequencies data only have short, mostly continuous, and constant-length dependencies with the exception of daily data. Daily data in M4 dataset remains to be a challenge for time series forecasting due to its high volativity and irregular pattern. Further investigation of model behavior on Daily data is beyond the scope of this paper. The pattern of dependency learned from the model is well-known as it is often utilized in other statistical model, such as ARIMA. The ability of visualize the learned weights from the model is an advantage of our framework in comparison to ESRNN.

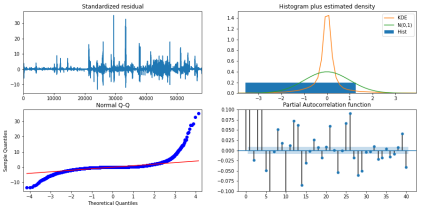
[1] If we think of time series forecasting as weighted average of lagged value, the attention map is a square matrix showing how which lagged value has significant contribution (i.e. large weight) on the prediction



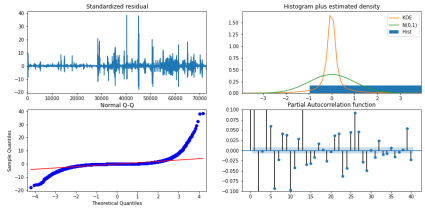
(a) ESTransformer - Hourly



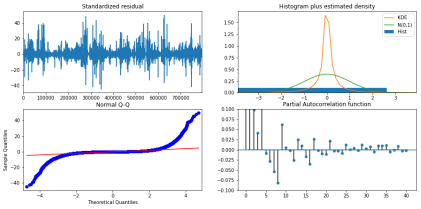
(b) ESRNN - Hourly



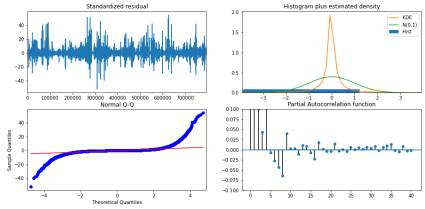
(c) ESTransformer - Weekly



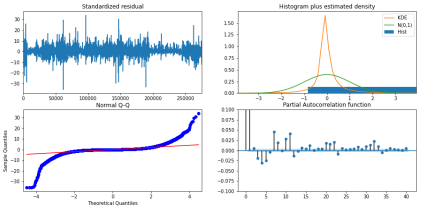
(d) ESRNN - Weekly



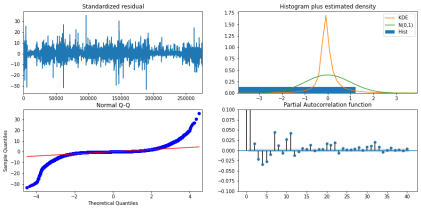
(e) ESTransformer - Quarterly



(f) ESRNN - Quarterly



(g) ESTransformer - Yearly



(h) ESRNN - Yearly

Figure 8: Residual diagnosis of ESTransformer and ESRNN on four different data frequencies (Hourly, Weekly, Quarterly, Yearly)

(a) Hourly data (b) Daily data (c) Weekly data

(d) Weekly data (e) Quarterly data (f) Yearly data

Figure 9: Attention mechanism in all frequencies data. The color bar shows the weight of each data points connection in a series. The bright fractions demonstrate a strong connection or highest weight, whereas the black ones demonstrate no weight or connection to the corresponding positions.