

Project Team:

Magdalene Cheong (23A467J)

Carel Teoh (22A187U)

Chim Jia Wen (23A462U)

Table of Contents:

Introduction.....	1
Data Preparation.....	1
Exploratory Data Analysis.....	1
Data Preprocessing.....	2
Model Results.....	2
Custom LSTM.....	3
N-HITS.....	4
NBEATS/NBEATSx.....	6
Temporal Convolutional Network.....	11
Conclusion.....	15
References.....	16

Introduction

In this final project, our team focused on environmental sustainability problem to develop a deep learning model for energy consumption forecasting. Dataset is obtained from Kaggle, which consists of London weather data, household data and energy consumption data (up to half-hourly granularity) from November 2011 to February 2014. The direction is to estimate the daily energy consumption data through available features. The models are deployed on HuggingFace.

Team members workload distribution:

Magdalene - Exploratory Data Analysis, NBEATS/NBEATx model development

Carel - Data Cleaning and Scaling, Temporal Convolutional Network model development

Jia Wen - Data Preparation, N-HITS exploration, Custom LSTM model development (GitHub link: https://github.com/jiawenchim/ITI110_codes)

Data Preparation

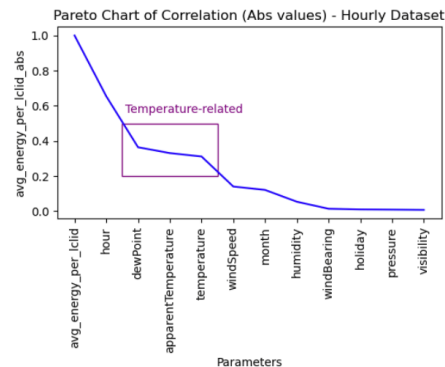
The raw data comes in 112 csv files from 112 blocks in the UK (source: [Kaggle](#)). The data is concatenated and aggregated into hourly energy consumption & number of unique smart metres..

From the available energy consumption data, we aggregated the average hourly energy consumption per smart metre. This step is required as the number of smart metres is not constant throughout the timestamps. Then, the energy consumption data is merged with weather and holiday information.

Exploratory Data Analysis

Main discoveries (Link: [EDA](#)):

- 1) Unusually low meter readings with median 0 which are thus removed as outliers.
- 2) Energy consumption has strongest correlation with temperature-related parameters but the value is not high due to time-dependency.



- 3) Time-dependency. There is statistically significant difference in energy for different-hour-within-day and same-hour-between-months base on One-way ANOVA tests.

Data Preprocessing

Data preprocessing was done on the dataset after it was prepared. Initially, work was done on the daily energy consumption dataset. However, the group quickly discovered that the performance was not desirable, therefore, a switch was made to use the hourly dataset. The normal process of exploration was employed on the dataset:

1. `df.head()` to look at the data structure
2. `df.columns` – identify the columns
3. `df.info()` to check for Dtype and null values
4. `df.isnull().sum()` – to identify the areas of NaN and NaT
5. drop the categories that are not used
6. Filter the time and `tstp` columns to display differences and drop one of them.
7. Checking for duplicates
8. Imputing missing data with mean, where appropriate.

The dataset was split into 80/20 split. After which the train and test data were scaled. The `MinMaxScaler` was used to normalize the values. However, depending on the choice of our models for training, the decision was left to individual members of the group to decide if the model should be scaled using standardization or normalizing method, to give flexibility.

Data Preprocessing code files: <https://github.com/Carel555/ITI110-Project.git>

Model Results

Custom LSTM

Input features selected are "month", "hour", "temperature","humidity","windSpeed","holiday".

* Output from Keras tuner hyperparameters search

Figure 1: Custom LSTM Model Training Data

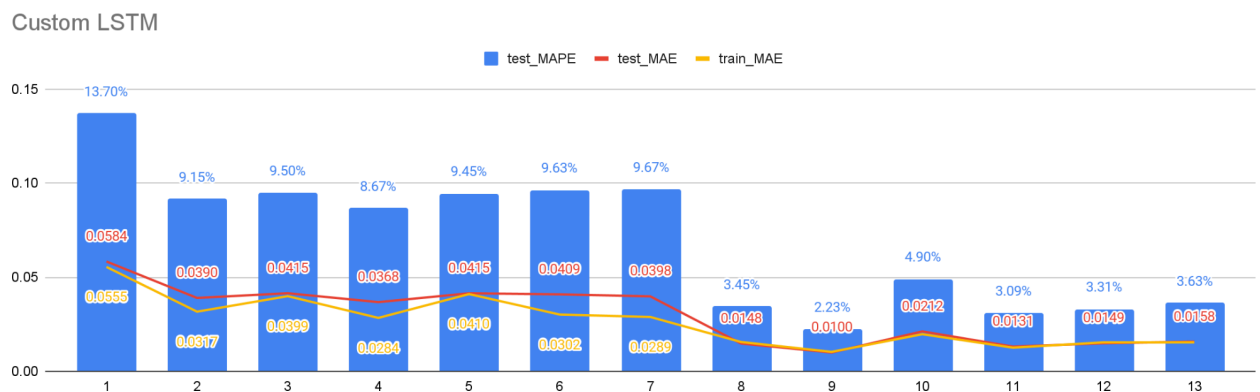


Table 1: Custom LSTM Model Experimentation Details

Run number	1	2	3	4	5	6	7	8	9	10	11	12	13
# of output	1	1	1	1	1	1	1	1	1	24	1	1	1
# layers	1	1	2	2	3	3	3	2	2	2	2	3	2
# units	50	256*	50,50	96,96*	50,50,50	128,128,128*	128,224,352*	96,96	96,96	96,96	480,480*	224,32,128*	96,96
Learning rate	0.001	0.01*	0.001	0.001*	0.001	0.001*	0.001*	0.001	0.001	0.001	0.01*	0.01*	0.001
Past X hours data as features	0	0	0	0	0	0	0	7	24	24	7	7	1
MAE	0.0584	0.039	0.0415	0.0368	0.0415	0.0409	0.0398	0.0148	0.01	0.0212	0.0131	0.0149	0.0158

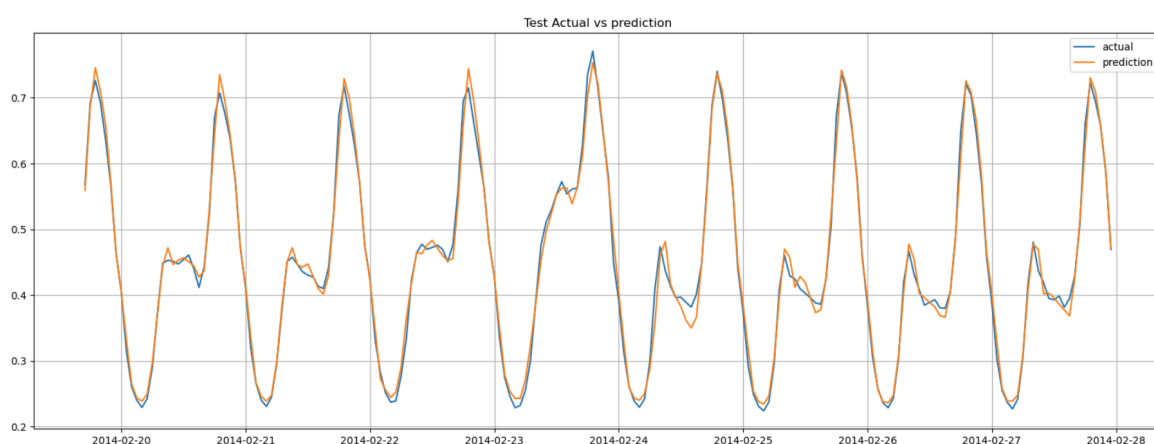
Custom LSTM experimentation runs were done on 500 epochs with early stopping patience at 20.

The experimentation starts with a single LSTM layer with 50 units, which has MAE of 0.0584. With the adoption of Keras Tuner hyperparameters search library, the best single layer units count is 256 (MAE 0.039). Increasing to two layers with 96 units shows further improvement

(MAE 0.0368). However, more attempts to adjust the layers count, units count and learning rate do not yield significant improvement.

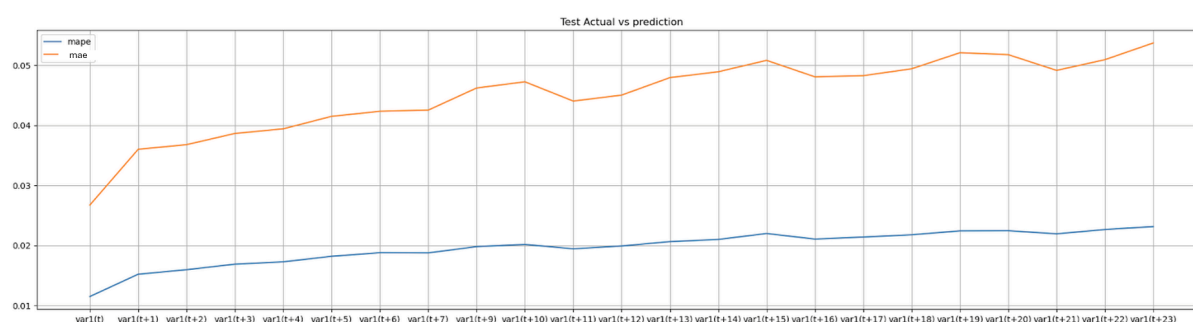
A different approach is attempted by creating more input features using historical data as input features. This approach is discussed in this article [“How to Convert a Time Series to a Supervised Learning Problem in Python”](#). This approach has shown significant improvement to our model, bringing MAE down to 0.0148. Increasing the historical data used as features to 24 further brings down the MAE to 0.01. Error analysis shows MAE and MAPE do not have seasonal correlation. Instead, hourly correlation is observed. (5am to 6pm generally shows 2-3% MAPE while the remaining hours show 1-2% MAPE).

Figure 2: After fine tuning, the custom LSTM model is able to capture the day-to-day and hour-to-hour variations and nuances well.



While the previous runs are meant to predict 1 future datapoint (i.e. next one hour energy consumption), Run 10 shows the model can perform relatively well to predict multiple outputs (24). It's observed that for multiple output prediction, the further predicted timestamps tend to show higher MAE and MAPE.

Figure 3: MAE and MAPE for the 24 predictions



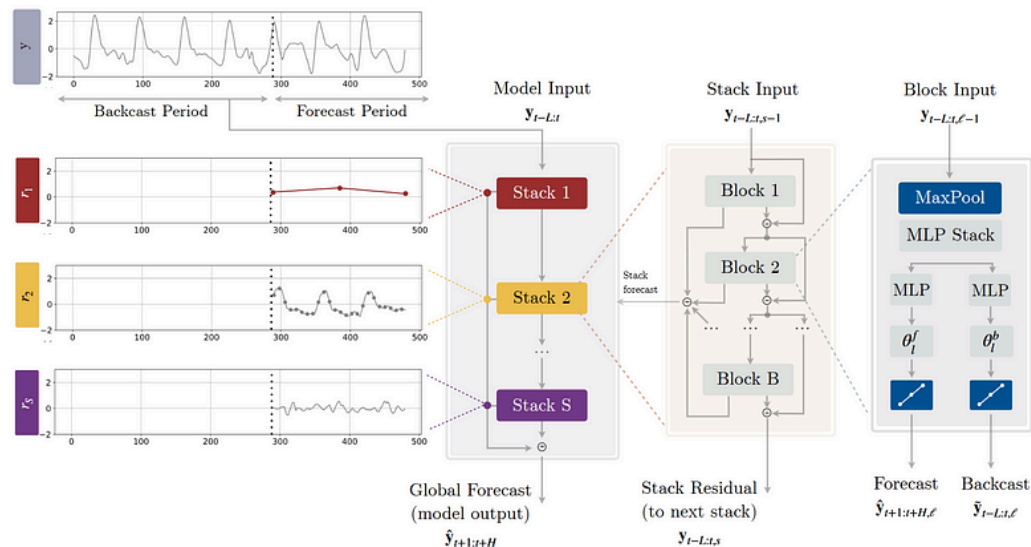
N-HITS

The custom LSTM model is compared against a pretrained time series model called N-HITS. N-HITS stands for **N**eural **H**ierarchical interpolation for **T**ime **S**eries forecasting.

“In short, N-HiTS is an extension of the N-BEATS model that improves the accuracy of the predictions and reduces the computational cost. This is achieved by the model sampling the time series at different rates. That way, the model can learn short-term and long-term effects in the series. Then, when generating the predictions, it will combine the forecasts made at different time scales, considering both long-term and short-term effects. This is called *hierarchical interpolation*.”

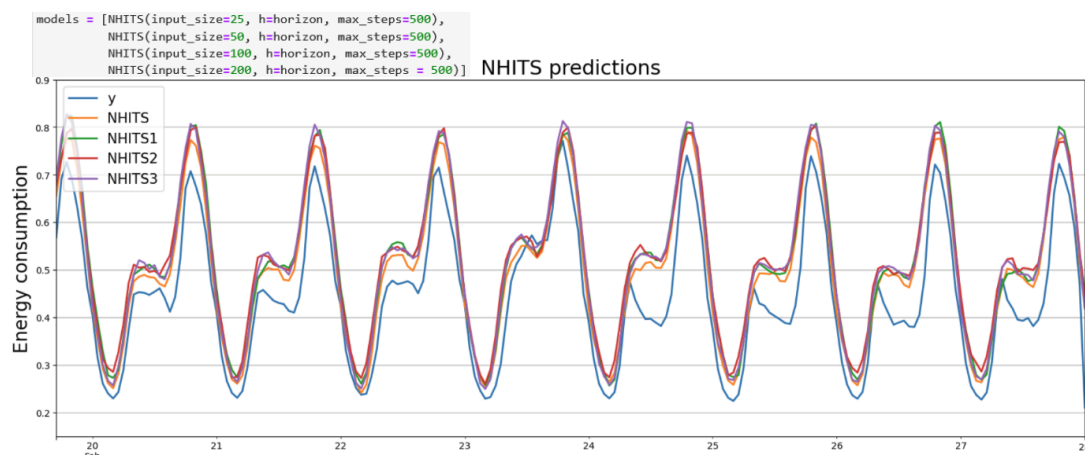
Source: All About N-HiTS: The Latest Breakthrough in Time Series Forecasting, by Marco Peixeiro (Nov, 2022)

Figure 4: Architecture of N-HiTS. Again, the model is made of stacks and blocks, just like N-BEATS. Image by C. Challu, K. Olivares, B. Oreshkin, F. Garza, M. Mergenthaler-Canseco and A. Dubrawski from N-HiTS: Neural Hierarchical Interpolation for Time Series Forecasting



There are a number of hyperparameters for N-HiTS, such as input_size, activation function, stack types, number of stacks, number of blocks, downsampling frequency e.t.c. In this project, we explored whether changing input size impacts the model's accuracy. While input size of 50 seems to show slightly better performance, all runs ranging from 25-200 input size show comparable MAE of around 0.04. There is much room for improvement for future work.

Figure 5: N-HiTS performance with minimal tuning



GitHub link to code files: https://github.com/jiawenchim/ITI110_codes

Deployment on HuggingFace (LSTM):

https://huggingface.co/spaces/jiawenchim/energetic_forecaster_london

NBEATS

Evaluation of the PyTorch NBEATS model started with the vanilla model as shown in Figure 6. The model parameters are first evaluated on input_size, loss function and max_steps to get a working model before evaluating on dataset input parameters (temperature, humidity and windSpeed and if the day is public holiday). MAE and MAPE are used to measure performance. 2 datasets are evaluated: daily and hourly.

Figure 6: NBEATS Vanilla Model

```
NBEATS(h, input_size, n_harmonics=2, n_polynomials=2,
      stack_types=['identity', 'trend', 'seasonality'],
      n_blocks=[1, 1, 1], mlp_units=[[512, 512], [512, 512], [512, 512]],
      dropout_prob_theta=0.0, activation='ReLU', shared_weights=False,
      loss=MAE(), valid_loss=None, max_steps=1000, learning_rate=0.001,
      num_lr_decays=3, early_stop_patience_steps=-1, val_check_steps=100,
      batch_size=32, valid_batch_size=None, windows_batch_size=1024,
      inference_windows_batch_size=-1, start_padding_enabled=False,
      step_size=1, scaler_type='identity', random_seed=1,
      num_workers_loader=0, drop_last_loader=False,
      **trainer_kwargs)
```

Figure 7 and 8 are examples of how error changes with different input_size and max_steps. The optimal setting is derived by fitting a best fit line and identifying the minimum point.

Figure 7: Deriving Optimal input_size

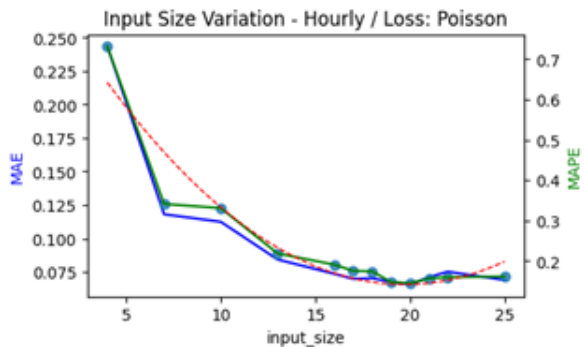
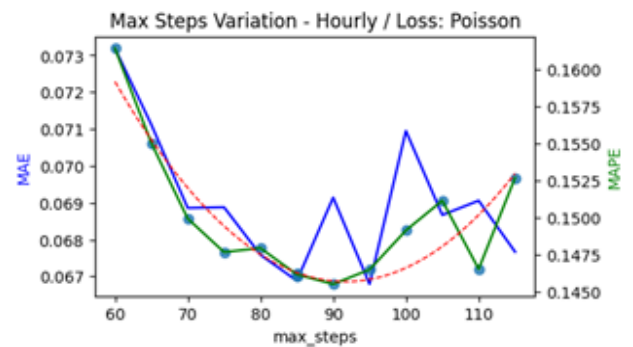


Figure 8: Deriving Optimal max_steps



Summary of the evaluation

Table 2: Optimal Model Parameters Derived

Dataset	Loss Function	Performance on Val data		Optimised Parameters		Remarks
		MAE	MAPE	input_size	max_steps	
Daily	Poisson	0.501	0.048	19	83	Weather and holiday no strong influence
	MAE	0.552	0.052	21	80	Weather and holiday no strong influence
Hourly	Poisson	0.067	0.146	19	85	Weather and holiday no strong influence
	MAE	0.085	0.182	20	150	Weather and holiday no strong influence; at high number of max step of 150, optimal loss is still not achieved -> abort

Figure 9: Impact of Weather Parameters and Holiday on MAE/MAPE

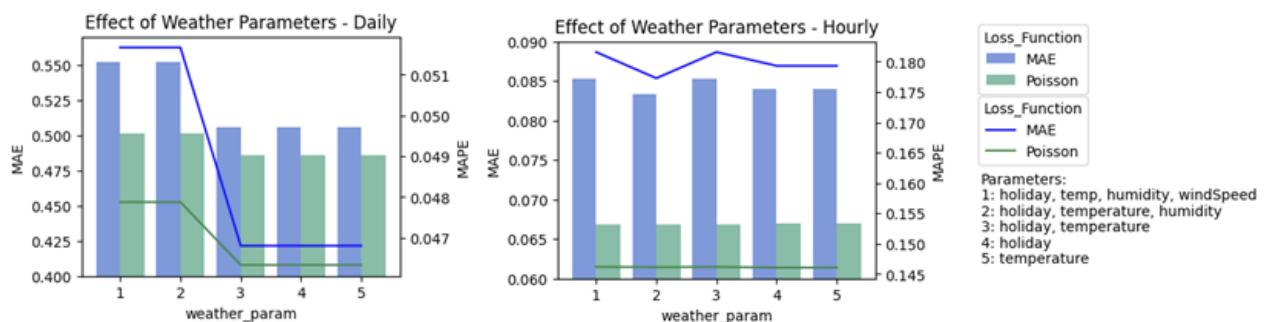
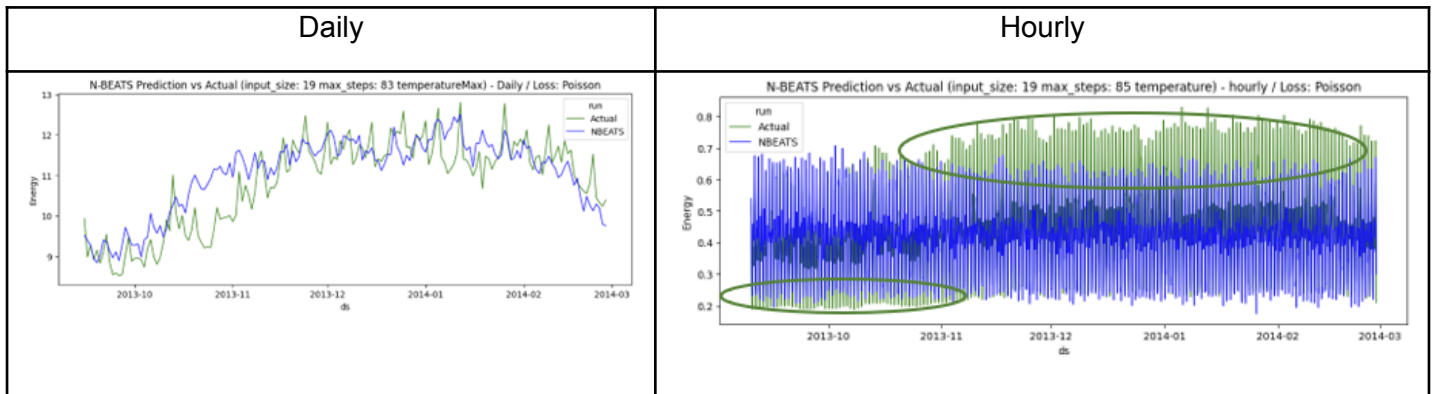


Figure 10: Plot Prediction vs Actual (Validation Dataset)



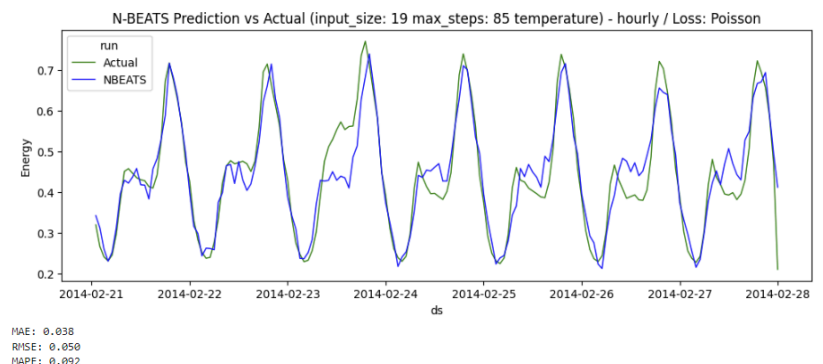
Conclusion from results table (Table 2) and Figure 9 and 10:

- 1) Poisson distribution as loss function, gives better MAE.
- 2) The exclusion of humidity and windSpeed appear to improve the error for daily dataset but only by about 0.5% which is negligible. MAE does not shift for hourly dataset with exclusion of any weather parameter. Hence, weather parameters and presence of holiday do not affect the model performance.
- 3) Figure 10 (left chart) shows high fluctuation in actual and predicted values for daily dataset hence high MAE. Hence daily dataset is not ideal for prediction purpose.
- 4) The best model from the evaluation is thus on hourly dataset using Poisson distribution as loss function and treated as a univariate series (highlighted in blue in Table 2).
- 5) However, peculiar behaviour is observed on the hourly dataset prediction (circled in green in Figure 10). The model slightly over-predicts before Nov and under-predicts from Nov onwards over winter months. This will be further discussed later.

Test Model On Test Dataset

Before delving into the peculiar behaviour of under-predicting during winter, the best model is run on the test dataset. Figure 11 on the right shows that actual energy consumption and prediction mostly fit well and has a much better MAE of 0.038. It is likely that the test dataset is moving out of the winter period and is not affected by the under-prediction phenomenon hence better MAE.

Figure 11: Model Prediction On Test Dataset



Error Analysis

Figure 12 zooms in into the period where the under-predicting starts.

The top chart is with temperature as input parameter. The bottom chart is without any input parameter. Both show similar prediction trend. In both, the model does not react as temperature gradually declines.

'A' denotes pre-winter and 'B' winter.

Figure 12: Zoom In Prediction vs Actual Plot (Pre-Winter)

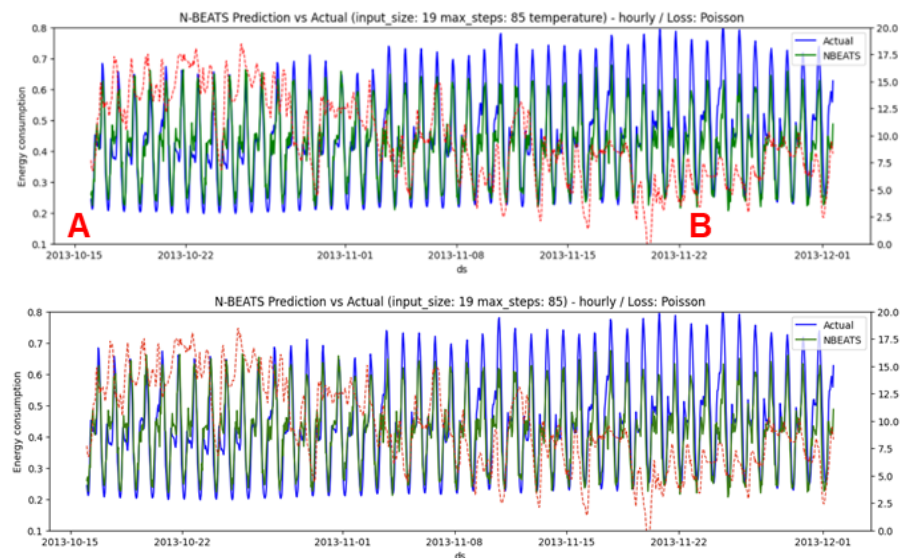


Figure 13 below further zooms in into pre-winter (A) and winter time (B) windows. The left chart below shows that in pre-winter, actual and predicted minimum energy consumption is on point at the hour. The right chart shows a shift in the minimum energy consumption in winter period.

Figure 13: Zoom In Into Pre-Winter and Winter Time Windows

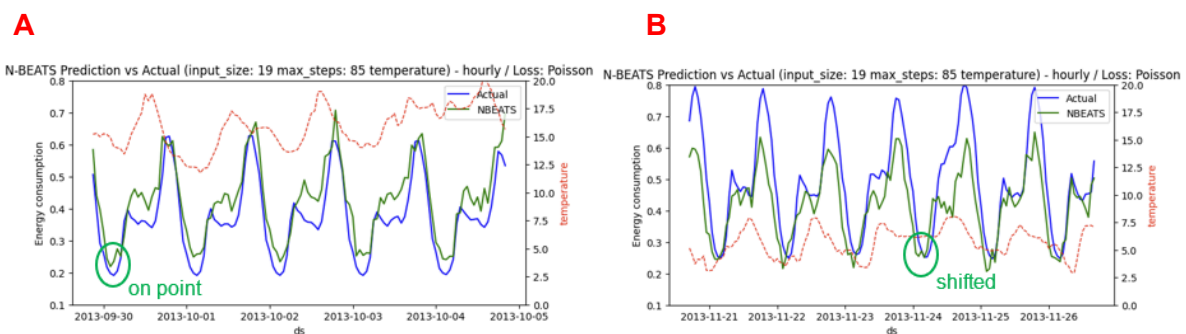
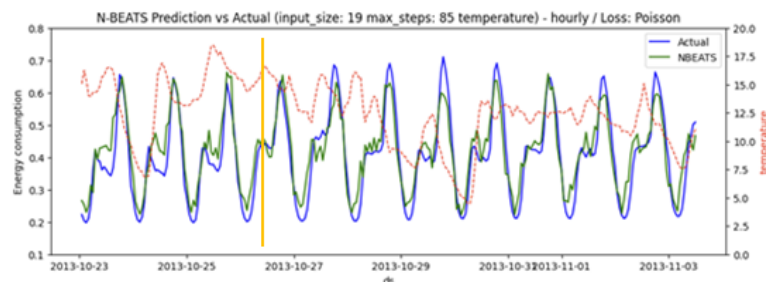


Figure 14: Minimum Point Transition Over Time

Figure 14 shows a wider window period and it can be seen that the shift starts as temperature starts to decline and NBEATS model transits into under-prediction (after the orange line).



Optimisation Attempts

Attempt 1: evaluate shifting the energy by hours before and after

In this evaluation, every shift is 1 hour. While shifting energy values helps align actual and prediction for winter period, the pre-winter period becomes misaligned. Therefore, a zero-sum improvement. Figure 15 shows that 0 shift has the lowest MAE. In all these runs, the under-predict phenomenon is still present.

Figure 15: Change in MAE With Shift



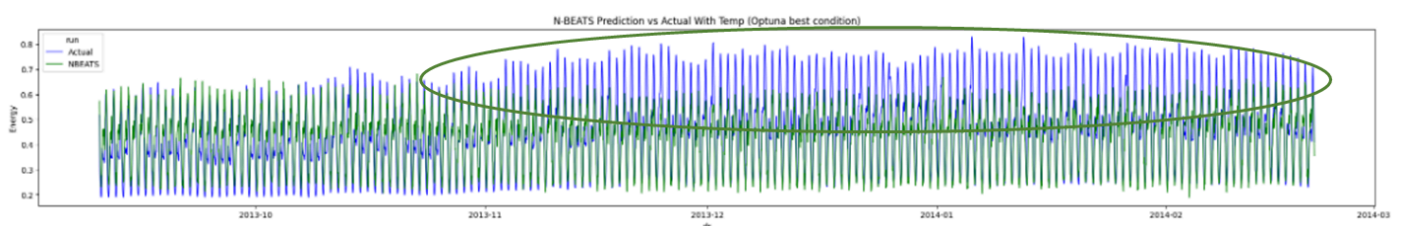
Attempt 2: Use Optuna to tune hyperparameters.

Table 3 below shows the list of hyperparameters tuned and the settings used (loss function: Poisson). The best model derived by Optuna has not improved MAE. The under-predict phenomenon is still present (circled in green in Figure 16).

Table 3: Optuna Hyperparameter Best Parameters (Loss Function: Poisson)

Optuna Hyperparameter Tuning	Best Param without input parameters (MAE: 0.069)	Best Param with temperature as input (MAE: 0.065)
<pre> input_size = trial.suggest_int('input_size', 1, 60) n_blocks_season = trial.suggest_int('n_blocks_season', 1, 3) n_blocks_trend = trial.suggest_int('n_blocks_trend', 1, 3) n_blocks_identity = trial.suggest_int('n_blocks_identity', 1, 3) mlp_units_n = trial.suggest_categorical('mlp_units', [32, 64, 128, 256, 512]) num_hidden = trial.suggest_int('num_hidden', 1, 3) n_harmonics = trial.suggest_int('n_harmonics', 1, 5) n_polynomials = trial.suggest_int('n_polynomials', 1, 5) scaler_type = trial.suggest_categorical('scaler_type', ['standard', 'robust']) learning_rate = trial.suggest_loguniform('learning_rate', 1e-5, 1e-1) </pre>	<pre> input_size = 48 n_blocks_season = 1 n_blocks_trend = 3 n_blocks_identity = 2 mlp_units_n = 32 num_hidden = 3 n_harmonics = 2 n_polynomials = 4 scaler_type = 'robust' learning_rate = 0.0074547496109055545 </pre>	<pre> input_size = 41 n_blocks_season = 1 n_blocks_trend = 1 n_blocks_identity = 2 mlp_units_n = 512 num_hidden = 2 n_harmonics = 2 n_polynomials = 3 scaler_type = 'standard' learning_rate = 0.0019886613692826844 </pre>

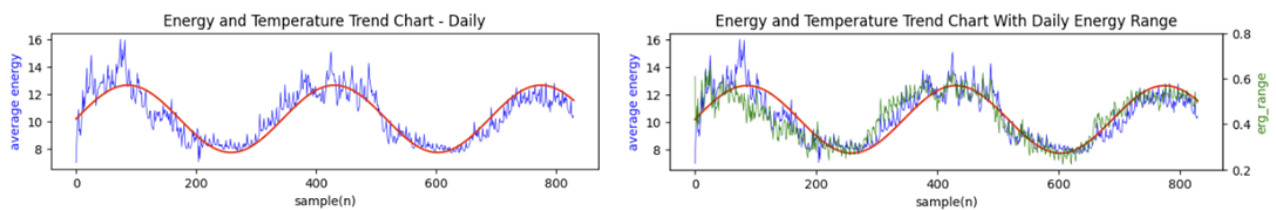
Figure 16: Prediction vs Actual for Optuna Best Parameters (input param: temperature)



Attempt 3:

The energy trend shows high fluctuations at the top. It is plotted against sine curve to consider evaluating a mathematical transformation to boost energy prediction at the top (Figure 17 left). From EDA, we learn that energy fluctuation during winter is larger. Within-day energy range is derived from hourly dataset and merged with daily dataset (Figure 17 right). Unfortunately it is not immediately clear how a mathematical formulation can be derived. Further difficulty arises as the last sine wave has very stable energy values. This wave is the validation dataset and might give unexpected evaluation results.

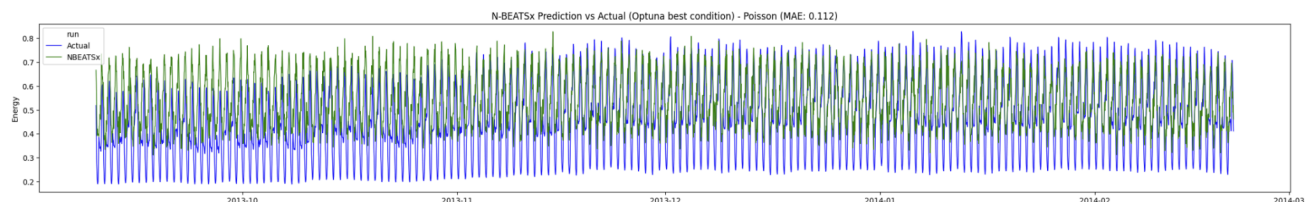
Figure 17: Daily Energy Against Sine Curve



NBEATSx Evaluation

NBEATSx is an improved version of NBEATS that extends its capabilities with external parameters by the inclusion of an exogenous block into the architecture. Optuna is used to derive the best parameters (with all weather parameters in) for both Poisson and MAE loss functions. However, the performance still could not improve - MAE worse at 0.112. Figure 18 shows the performance of Poisson loss function.

Figure 18: NBEATSx Performance With Optuna Best Parameters



Python scripts for the experiments: [NBEATS_NBEATSx Py Scripts](#)

Experiment log: [NBEATS Expt Log](#)

Deployment:

The app is designed so that the user can only select legitimate values, e.g. 0 to 23 for hour therefore a slider is chosen. The use of textbox otherwise necessitates coding an error message when invalid number like '26' is entered for 'hour'. A slider automatically constrains the boundaries, is more user-friendly and is also easy to implement. The predicted energy output is rounded to 3 decimal places for better readability.

Link: https://huggingface.co/spaces/magcheong/ITI110_Energy_Prediction

TEMPORAL CONVOLUTIONAL NETWORK (TCN)

The TCN architecture consists of dilated, causal 1D Convolutional layers, with the same input and output length. Traditionally, CNN and RNN were used in sequence for time series data, CNN is able to deal with the **spatial temporal information**, while RNN takes in the **high-level temporal information**

TCN on the other hand, handles the 2 levels of information hierarchically and unifies these 2 approaches, capturing spatial and high-level temporal information.

In the real-world scenario, predictions need to be based solely on past and present information, where we do not have access to the future. Causal predictions ensure this – it ensures that the model does not ‘peek’ into the future, while making predictions. TCN generates future predictions using only past information. This helps to maintain realistic and interpretable outcomes. Causal padding is used for this purpose. It adds zeros at the beginning of the data sequence, to prevent future information from influencing current prediction. If we look at the standard CNN, the filters ‘see’ both past and future elements within the window. This is a problem where time-series data is concerned, because future values have not yet occurred. Causal convolutions address this problem.

Model Summary

TCN Model design:

When we talk about model design, we are referring to the **high-level blueprint** of the model, specifying the different types of layers used and their connections. In my codes used, the sequence of layers are defined (TCN, Dropout, Flatten, Dense), outlining the overall design.

TCN Architecture:

Model architecture on the other hand delves deeper into the **specific configurations and parameters** of each layer. Here in my code, it details the number of filters, kernel size dilations, activation function and return sequence to give instructions on how the information is to flow across the residual blocks.

Although my TCN model does not have, explicitly, ‘residual block’ layers in them, however the model does employ **skip connections** that resemble the core functionality of residual blocks.

When nb_stacks=3 is used (for example), effectively it creates **three cascading stacks with skip connections**, which basically is the essence of a residual block.

The way the layers are combined in the tf.keras.Sequential model invariably stack the TCN layers on top of each other, creating **sequential residual block**.

Below is what a typical TCN model with residual block would look like [6].

```
# Define the TCN model with explicit
residual blocks

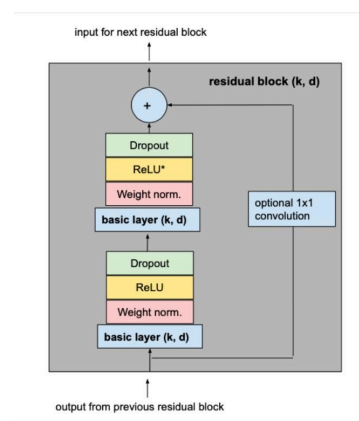
tcn_model = tf.keras.Sequential([

    ResidualBlock(filters=64,
kernel_size=5, dilation_rate=1),

    ResidualBlock(filters=64,
kernel_size=5, dilation_rate=2),

    tf.keras.layers.Flatten(),

    tf.keras.layers.Dense(1) # Output
regression])
```



The code blocks below show my model design and 3 types of architecture I chose to run my experiments with. One as a basic model without residual blocks and one with residual blocks for comparison and evaluation. I also did a 3rd model with forward prediction. The link will be provided below due to space constraint:

Without Residual block:

```
# Define the TCN model WITHOUT residual blocks

input_shape = (X_train.shape[1], X_train.shape[2]) # Input shape based
on the number of features and the additional dimension

tcn_layer = tf.keras.layers.Conv1D(filters=64, kernel_size=3,
dilation_rate=1, padding='causal', activation='relu')

tcn_model = tf.keras.Sequential([

    tcn_layer,

    tf.keras.layers.Flatten(),

    tf.keras.layers.Dense(1) # Output regression])
```

With Residual block:

```
# Define the TCN model WITH multiple residual blocks

tcn_model = tf.keras.Sequential([

    TCN(nb_filters=64, kernel_size=5, nb_stacks=3, dilations=[1,2],
padding='causal', activation='relu', return_sequences=True),

    tf.keras.layers.Dropout(0.2), # Add dropout after each causal
convolution

    tf.keras.layers.Flatten(),

    tf.keras.layers.Dense(1) # Output regression])
```

A third TCN model with forward prediction was also trained with the following :

`nb_filters=64, nbs_stacks=3, dilation=[1,2], kernel_size=5, epochs (300), learning rate (0.001)`. The forward prediction architecture is also with residual blocks. (see link below).

Link to codes in github: <https://github.com/Carel555/ITI110-Project.git>

(7.1 TCN with Residual Blocks & 7.2 TCN for forward predictions)

The differences between the two model designs used are in the `nb_filters`, `nb_stacks` and `dilation`, mainly, in the TCN model with Residual Blocks.

While the term 'residual block' is not used in my code, the key principle of residual connections through stacked TCN layers is utilized, to achieve similar benefits like improved gradient flow and potentially better performance.

By using `return_sequences=True`, this ensures the output of each block is passed to the next, maintaining the flow of information across residual blocks.

The dropout layer after each TCN block helps with regularization and prevents overfitting.

TCN hyperparameters and parameter:

The hyperparameters that I used to tune the model were `filter(64, 256)`, `kernel_size (3,5)` and `dilation rate(1,4)` for the TCN model without residual block. The model with residual blocks were tuned with the following hyperparameters: `nb_stacks(3,4)`, `nb_filters(64,128)` and `dilation_rate(1,2)`. During the experimentation, I found that tuning these hyperparameters did very little to improve the performance results.

Epochs

As mentioned above, I did not see much change tuning the hyperparameters. However, changing the number of features used for training, and the number of epochs (10, 100, 300, 500, 1000) as well as the learning rate made a huge difference. The best test MAE and test RMSE results were for 300 epochs (0.0418 and 0.0531, respectively) with the learning rate of 0.00001.

The model with residual blocks that were trained on 100 epochs and 128 filters and learning rate of 0.00001 had the best results of test MAE (0.0245) and test RMSE (0.1567). The RMSE was not the best. Although I have evaluated the model with MAPE, however, the scores were extremely high for the train MAPE. Many studies have been done on the shortcomings and misleading results of MAPE [7].

There are 3 reasons why MAPE may not be a good measure of performance. And one or more of these fit the situation here. Due to time limitation, it was not possible to investigate further into the matter, apart from knowing for a fact that our datasets do have many zero values (eg. non-holiday, and hour):

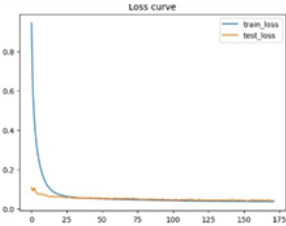
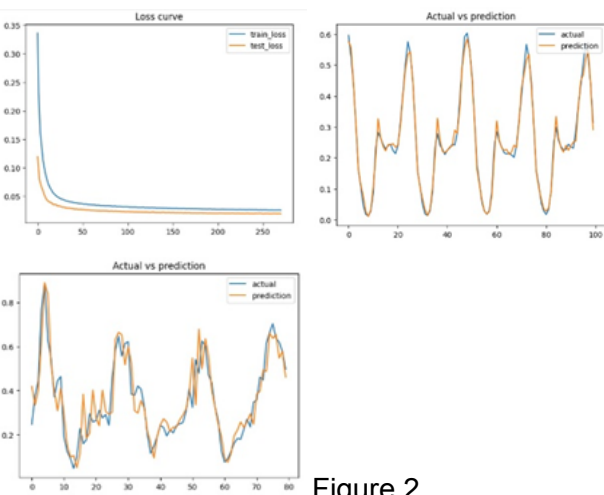
1. It cannot be used if there are zero values, because there would be a division by zero.
2. For forecasts which are too low, the percentage error will not exceed 100%. But where forecasts are too high, there is no upper limit to the percentage error.
3. MAPE puts a heavier penalty on negative errors, than on positive errors.

See experimental log: [ITI110 Model Training data](#)

Learning rate

As mentioned above, tuning the learning rate also resulted in improvements in the performance results.

Best Model And Evaluation

 <p>Figure 1</p>	<p>Train MAE: 0.036</p> <p>Test MAE: 0.0418</p> <p>Train RMSE: 0.0426</p> <p>Test RMSE: 0.0531</p>
 <p>Figure 2</p>	<p>Train MAE : 0.0258</p> <p>Test MAE: 0.0196</p> <p>Train RMSE: 0.0274</p> <p>Test RMSE: 0.026</p>

Code: TCN for forward predictions: <https://github.com/Carel555/ITI110-Project.git>

In Fig.1, we see that the model performs well on both the train and test data, which are reflected in the relatively low MAE & RMSE values. Generally there is good agreement between predictions and actual values in both train and test data.

In Fig.2, we see exceptionally low errors on both train and test data also. The MAE demonstrates nearly equivalent performance on both train and test data. And RMSE confirms and emphasizes the model's accuracy on the test data. The train and test losses are reflected in the MAE values. Overall, these charts show the model's ability to learn underlying patterns.

Deployment

Code for streamlit_app.py : <https://github.com/Carel555/ITI110-Project.git>

Conclusion

NBEATS, despite numerous attempts to optimise has not been able to improve further. The peculiar under-prediction during winter period could not be resolved. One possible reason is NBEATS is first designed to be a univariate model. Despite correlation between temperature and energy consumption, the inclusion of temperature as an input parameter did not manage to nudge the performance up. NBEATSx is subsequently also evaluated. NBEATSx is an improved version of NBEATS with an inclusion of an exogenous block to improve learning on external parameters. Unfortunately, the best parameters produced by Optuna did not produce better performance.

The TCN model without residual blocks and even without any adjustments of hyperparameters seem to perform reasonably well. However, with residual blocks and hyperparameter tuning together with balancing of the number of epochs and learning rate, it dramatically improves the performance results.

Custom-trained LSTM network, despite its simple structure, can perform relatively well on our dataset, due to custom weight adjustments and being able to adapt to the nuances in our data better – a classic example of how a custom network can sometimes be a better choice. However, custom training is more resource intensive and the pretrained model still outshines in many circumstances especially in the case where there is a shortage of training data.

References

- 1) NBEATS: <https://nixtlaverse.nixtla.io/neuralforecast/models.nbeats.html>
- 2) NBEATSx: <https://nixtlaverse.nixtla.io/neuralforecast/models.nbeatsx.html>
- 3) NBEATS: <https://forecastegy.com/posts/multiple-time-series-forecasting-nbeats-python/>
- 4) N-HITS: <https://nixtlaverse.nixtla.io/neuralforecast/models.nhits.html>
- 5) N-HITS: <https://towardsdatascience.com/all-about-n-hits-the-latest-breakthrough-in-time-series-forecasting-a8ddcb27b0d5>
- 6) <https://machinelearningmastery.com/convert-time-series-supervised-learning-problem-python/>
- 7) [Temporal Convolutional Networks and Forecasting | by Francesco Lässig | Unit8 - Big Data & AI | Medium](#)
- 8) [python - Why Keras MAPE metric is exploding during training but MSE loss is not? - Stack Overflow](#)