Stage 3 ML Project

Thomas DANIEL Théo CHICHERY Jean MARCHEGAY Julien DE VOS

December 8, 2024
DIA 3

Contents

1. Previous Methods and Limitations

In stage 2 we explored 3 more advanced models: XGBoost, LSTM and NN.

1.1. XGBoost

XGBoost (Extreme Gradient Boosting) is a powerful and efficient tree-based algorithm that is well-suited for tabular data like this dataset. XGBoost has the following limitations:

- **Limited Extrapolation Capabilities**: Can't extrapolate as it's based on a tree principle.
- **Risk of Overfitting**: The model may be prone to overfitting if not properly parameterized.
- **Time**: Time-consuming training on large datasets.

1.2. Neural Network and LSTM

Artificial Neural Networks (ANNs) are flexible models capable of approximating complex functions.

Long Short-Term Memory (LSTM) networks are a type of Recurrent Neural Network (RNN) specifically designed to handle sequential data.
They have similar limitations :

- **Risk of Overfitting**: The model may tend to overfitting if the dataset is too small.
- **Resource-Intensive Training**: Training LSTMs requires a lot of resources, especially in terms of memory and time.
- **Model Interpretability**: The model is difficult to interpret. It is therefore difficult to understand where the results come from.

2. Improvement Assumptions

Based on the results of the previous methods, the following improvements are proposed :

2.1. Multivariate and Univariate

We will now try to process the data in a univariate way, taking into account only one variable, or in a multivariate way, taking into account several columns.

2.2. Advanced Modeling Techniques

In order to get a better accuracy, the following models will be explored:

- **Statistical model**: A statistical model such as ARIMA can efficiently predict stationary series by taking time series into account.

- **Deep Learning**: Neural networks, including LSTM, CNN or VAE, will be reworked for their interesting initial results and predictive capabilities.

By parameterizing these models effectively, we should be able to improve the accuracy of our results.

3. Problem Formalisation and Methods

3.1. New Algorithm Description

3.1.1 TCN

TCN (Temporal Convolutional Networks) is a deep learning model designed to analyze time series.
Temporal Convolutional Networks: A Unified Approach to Action Segmentation
It offers many advantages for our solution :

- **Memory requirements**: TCN needs less memory capacity than LSTM because each layer has only one kernel.
- **Performance**: Thanks to parallelization, the model achieves good processing times for a neural network and presents good results, particularly in capturing temporal dependencies.

3.1.2 CNN

Convolutional Neural Networks (CNNs) are powerful neural networks designed to capture patterns in grid-like data, making them suitable for tasks like residual load forecasting.

- **Capturing Non-Linear and Spatial Relationships**: CNNs detect complex interactions between variables such as H sun, T2m, and WS10m through convolutional layers, enabling them to model intricate patterns in the data.
- **Temporal Pattern Recognition**: Using 1D convolutions, CNNs can recognize short-term temporal dependencies in sequential datasets, making them effective for time-series features like weather conditions.
- **Efficiency and Regularization**: With techniques like weight sharing and pooling, CNNs are computationally efficient, while dropout and batch normalization improve generalization and prevent overfitting.
- **Automatic Feature Extraction**: CNNs learn hierarchical features directly from raw inputs, reducing the need for manual feature engineering.

3.2. Limitations

TCN

- **Domain transfer**: TCN has difficulty handling domain transfers, especially when moving from a short to a long history.

CNN

- **Dataset size**: CNNs require larger datasets because of its complexity
- **Temporal efficacy**: CNNs are less effective at modeling long-term temporal dependencies compared to RNNs or LSTMs.

4. Methodology

For Stage 3 we decided to optimize the algorithms we had, but also to try out another TCN.

We have therefore chosen to focus on the most promising algorithms: Neural networks with LSTM, CNN and TCN.
To improve our results, we began by implementing the model repeat function from stage 2, which had given us much better results.

First, we implemented a CNN (Convolutional neural network) with the following structure :

Python Function: `CNN`

Here is the Python implementation of the `CNN` :

```python
input_layer = Input(shape=(X_train.shape[1], 1))

x = Conv1D(filters=80, kernel_size=3, activation='relu')(input_layer)
x = Dropout(0.3)(x)
x = Conv1D(filters=80, kernel_size=3, activation='relu')(x)
x = GlobalMaxPooling1D()(x)
x = Dense(256, activation='relu')(x)
x = Dropout(0.2)(x)

# Output layers for P and load
output_P = Dense(1, name='output_P')(x)
output_load = Dense(1, name='output_load')(x)
```

Several hyperparameters are used:

- **Optimizer**: We chose the adam optimizer for our predictions because it offers greater stability than the other optimizers, and since our data are noisy.
- **Model Layer**: For the layers of our model, GlobalMaxPooling1D makes our model less sensitive to small variations, but also allows us to concentrate on the important data.
- **Kernel size**: The kernel size of 3 allows 3 values to be viewed at the same time in the sequence. This provides a good compromise between computation time and the ability to see local relationships.

We also decide to predict Load and P separately to obtain better results on each prediction, and then subtract the two predictions to calculate the predicted residual load and compare it with the actual residual load.

We then choose to implement the TCN instead of improving the LSTM, as the model seems to have better results and requires less global resources.
In this case we have an additional step before creating the layers: We have to create sequences.

Python Function: `Create sequence`

Here is the Python implementation of the `Create sequence` :

```python
def create_sequences(data, seq_length):
    X, y = [], []
    for i in range(len(data) - seq_length):
        X.append(data[i:i+seq_length, :-1])  # Features
        y.append(data[i+seq_length, -1])     # Target (residual_load)
    return np.array(X), np.array(y)
```

This function can be used to create data sequences of a certain size. In our case, we've chosen sequences of 6 data items, so the function will take 6 rows of data in a row, take the variables used and the associated variables to be predicted, and return them as data items.
The value of 6 was chosen so as not to overload our model with predictions, while still giving it enough to work with.

We then create the layers of our model Here is the Python implementation of the `TCN Layers` :

```python
input_layer = Input(shape=(X_train.shape[1], X_train.shape[2]))

tcn_layer1 = TCN(nb_filters=64, kernel_size=3, dilations=[1, 2, 4],
return_sequences=True, dropout_rate=0.2)(input_layer)
tcn_layer2 = TCN(nb_filters=128, kernel_size=3, dilations=[1, 2, 4],
return_sequences=False, dropout_rate=0.3)(tcn_layer1)

dense_layer = Dense(64, activation='relu')(tcn_layer2)
dropout_layer = Dropout(0.3)(dense_layer)
output_layer = Dense(1)(dropout_layer)
```

These hyperparameters are generally the same as those used for the CNN.
This model has an additional parameter, the dilation. Each element in the dilation represents the number of values taken into account in the sequence:

- **1**: The filter takes consecutive values in the sequence
- **2**: The filter takes 1 value out of 2 in the sequence
- **4**: The filter takes 1 value out of 3 in the sequence

This time we try to predict residual load directly to see if this has an impact on the accuracy of the model.

Following this, we also try to predict only residual load with the CNN.

In this step, we implemented two models (CNN and TCN) in a univariate and multivariate manner.

5. Results

5.1. Metrics

As in Stage 2 to measure the effectiveness of our models, we'll use two metrics: R^2 and negative root mean squared error.

5.2. Overfitting

5.2.1 CNN and TCN

As these models are complex, they can tend to overfit in many cases:

- **Data quantities**: If the number of data items is too small compared with the complexity of the model, overfitting may occur.
- **Number of filters and layers**: If the model has too many filters or layers the model can develop a very high sensitivity to the data.
- **Hyperparameters management**: Wrong choice of hyperparameters can lead to overfitting

5.3. Mitigating Overfitting

To avoid overfitting these models, we use various techniques :

- **Dropout**: Dropout is a layer that can be added to a neural network structure to prevent model overfitting. Its parameter indicates the percentage of neurons that will be deactivated for an iteration. So if we have a dropout of 0.2, 20% of neurons will be deactivated.
- **Early stopping**: This makes it possible to stop the CNN before overlearning. In our case, we choose to stop it when the validation does not improve for 10 epochs.This way, the model doesn't have time to adapt to the data, and we get back a generalist model.
- **Sequence length** : In the case of time series, creating series that are too long can lead to overfitting.This can increase the complexity of the model or make it difficult to see important relationships.
- **Number of epochs**: Too large a number of epochs can lead to overfitting, so we must choose a sufficient number of epochs to have good precision while ensuring no overfitting.

5.4. Evaluation and Comparison with Previous Solutions

We can see that our results weren't quite as satisfactory as we'd hoped. In fact, we didn't manage to get any better results than with the XGBoost/LightGBM, despite using more advanced models.

This leaves us with a kaggle result of 47.61

Model	**RMSE**	**Kaggle Score**
XGBoost/LightGBM (Load-P approach)	17.01 (P), 23.77 (Load)	47.61 (Top 3 leaderboard)
CNN (univariate)	46.9	51.4
CNN (multivariate) w/o model repeat	40.02 (P), 22.31 (Load)	52
CNN (multivariate) w model repeat	23.035 (P), 21.4 (Load)	48 (Top 4 leaderboard)
TCN (univariate)	42.2	62,7

Table 1: Summary of Model Results and Performance

Nevertheless, it can be seen that for neural networks, univariate prediction (residual load prediction) does not greatly increase results compared to multivariate prediction (increase of only 0.4)
On the other hand, we can see that model repeat is essential for P prediction, as it enables us to go from an rmse score of 46.9 without it to 40 with it.

The fact that the results are not better with advanced models can be explained by two points :

- **Model complexity**: The complexity of models has made their use and the choice of hyperparameters complex. In addition, training time was also an obstacle to the various parameter tests.
- **Data quality**: Data quality was also complicated to manage, particularly for P, which required the use of model repeat to be predicted properly. It is therefore highly likely that other lines of data were not viable, rendering the model imprecise. Finally, we believe that the quantity of features was not large enough to predict load and P. Indeed, some of these features were not very useful, such as wind speed, while others could have been interesting to analyze, such as weather-related data like rain or snow.

6. Discussion and Conclusion

During this stage, we were able to see that CNNs were simpler and more effective than TCNs to use, particularly as TCNs require the creation of sequences whereas CNNs do not.

We also saw the importance of pre-processing, particularly with model repeat, which led to one of the biggest score improvements of the whole project. So we don't always have to prioritize model choice, but sometimes also check data quality in depth.

Finally, choosing a "simple" model like XGBoost gives very good results compared to neural network models, which require precise hyper-parameters and good quality data to ensure good predictions. We found that the training time for models like TCN did not allow us to check a large number of hyperparameters.

If we wanted to go further, we'd have to try and find better quality data and hyper-parameters more suited to our models.

7. Final Conclusion

This project showcased a wide range of models, from traditional machine learning methods to advanced deep learning techniques, highlighting their strengths and challenges in the context of residual load prediction. Models like XGBoost and LightGBM stood out for their robustness, ease of use, and ability to handle non-linear relationships. These models consistently delivered reliable results with minimal hyperparameter tuning, making them effective and efficient tools, especially when paired with well-preprocessed and engineered data.

Deep learning models brought a more nuanced approach to the table. Convolutional Neural Networks (CNNs) demonstrated their ability to capture patterns across multiple features, offering a simple yet powerful way to work with time-series data without requiring explicit sequence creation. On the other hand, Temporal Convolutional Networks (TCNs) excelled in modeling long-term dependencies through dilated convolutions, but their reliance on sequence preparation introduced additional complexity. While TCNs showed promise in capturing temporal patterns, their computational demands and sensitivity to hyperparameters made them less practical compared to CNNs in this project.

One of the most valuable insights gained was the importance of data preprocessing and feature engineering. Improvements in these areas, such as better handling of missing values, feature selection, and sequence generation, had a significant impact on model performance. Ultimately, while deep learning models offered advanced capabilities, simpler approaches like XGBoost emerged as reliable choices, particularly when computational efficiency and interpretability were key. Future work could focus on enhancing data quality, exploring more robust temporal models, and fine-tuning hyperparameters to unlock the full potential of these methods.

Accurate residual load predictions are essential for maintaining a stable and reliable energy supply as renewable energy systems, like solar panels, become more prevalent. The residual load, the difference between energy consumption and self-generated energy, depends heavily on weather conditions, making traditional forecasting methods insufficient. Precise forecasts enable energy suppliers to balance supply and demand effectively, ensuring sufficient energy during deficits and managing excess energy fed back into the grid. This supports grid stability, cost optimization, and the successful integration of renewables into the energy system, aligning with the goals of the energy transition.