Forecasting Energy Demand to Improve Decarbonized Grids

Chang Sun changsun@gatech.edu
Pei-Chuan Chao pchao33@gatech.edu
Matt Carney mcarney@gatech.edu

Georgia Institute of Technology

Abstract

The ability to accurately forecast energy demand is a crucial step towards improving decarbonized grids. Traditional statistical time series models have historically been the de facto approach but deep learning methods offer new possibilities. This paper presents a study of three different deep learning architectures, Spacetimeformer, DeepAR, Temporal Fusion Transformer, on a multi-scale load dataset with benchmark models. The results show that while we were able to improve several of the benchmark models, we did not surpass the previous most performant algorithm - exponential smoothing..

1. Introduction

1.1. Motivation

One of the biggest problems facing the transition to renewable energy sources is the rapid connection of new renewable projects to the grid. Unlike petroleum or natural gas, renewable sources such as solar and wind are not always available, which means that it can be difficult for grid operators to predict grid load on a consistent basis. This is important, because managing load is a critical part of operating a vast, complex electrical grid. If actual renewable generation is lower than forecasted, then it will be very difficult and costly to spin up reserve generation at the last minute. If it's higher than forecasted, then energy is wasted, as the grid will be at capacity and you'll have too much electricity to send to the grid. Accurately forecasting demand will improve the economics of renewable energy investments, which would help contribute to the fight against climate change. While predicting wind and solar energy generation are key components to renewable grid balancing, they both rely on an accurate forecast of load demand which is the focus of our project.

1.2. Background

Traditionally, load forecasting was done using statistical time series-based approaches such as ARIMA and ETS. These methods were specifically adapted for the power engineering domain, and have worked pretty well through today.

However, the deep learning revolution has unlocked the possibility of a step change in forecasting possibilities. Of course, blindly throwing fancy deep learning algorithms at a problem is not sufficient to make major breakthroughs. Tackling domain-specific problems requires methods and approaches honed by experience and exposure to that specific domain.

1.2. Dataset

In order to facilitate advances in both the renewable energy and machine learning communities, it's crucial to develop calibrated open-source datasets that are relevant to real-world power engineering problems[1]. Given the gravity of the climate and energy crisis we face, researchers at Texas A&M University developed PSML, a first of its kind open source multi-scale dataset. PSML consists of datasets and baseline machine learning results to serve as benchmarks for three use cases: 1.) Early detection of dynamic disturbance events, 2.) Load demand and renewable energy supply forecasting, and 3.) Synthetic generation of physical-law-constrained time series.

Given our chosen problem, we focused on improving on the performance benchmarks given for the second use case: Load demand and renewable energy supply forecasting. The provided dataset consists of minute-level measurements for the years 2018, 2019, and 2020 from 66 different sub-grids across the United States and includes the following key features: Time, Load Power (our target), plus various weather-related features such as: DHI (Direct Horizontal Irradiance), DNI (Direct Normal Irradiance), GHI (Global Horizontal Irradiance), Dew Point, Solar Zenith Angle, Wind Speed, Relative Humidity and Temperature. While there are 66 locations available, we chose to focus on CAISO (the California Independent System Operator Corporation) as this is the location used in the benchmark models.

2. Approach

Our primary approach is to experiment with new deep learning architectures and adapt them to the problem space. We note that traditional time series models such as the ETS family of models benchmark very well, performing at or near the top of the pack across all three years' worth of data. Zheng et al posit that deep learning methods have so far failed to match traditional time series methods due to their inability to remember historical information for long periods of time absent substantial feature engineering. We chose our approaches based on our initial hypotheses that they might help address some of the aforementioned issues:

a Spacetimeformer, a modified DeepAR, and a Temporal Fusion Transformer.

Of our three experimental approaches, two of them (Spacetimeformer and the Temporal Fusion Transformer) utilize attention mechanisms to retain relevant historical information that may be critical for forecasting farther-out time horizons. Using attention mechanisms allows the network to learn what context is relevant for a particular time horizon. DeepAR serves as a complement to the transformer based approaches as it combines Deep learning with traditional probabilistic forecasting. While we are likely not the only ones to try these new architectures, we believe that applying them to the specific problem of load forecasting is a relatively novel approach.

We implemented three different deep learning models on the provided dataset to forecast energy load demand. The performance of the models will be compared with the benchmarks presented in the paper (Table 1). We follow the same methodology (train/test campaign, performance metrics) outlined in the paper and discussed in the researcher's methods section below.

Our first approach was to try implementing each architecture within the existing framework and development environment provided in the PSML repository. However, we quickly realized this was untenable for the TFT, given the complexity of that model and the structure of the PSML code we started with. Similarly, while the PSML code was able to run for DeepAR making any modifications led to the code package breaking down. Thus for TFT and DeepAR, we pivoted to using a new pytorch-forecasting implementation[2, 3] and creating new development environments focused solely on the TFT and DeepAR experiments.

3. Experiments and Results

Our standard for success was whether we could improve on the benchmarks for models in the same category as the architectures we were testing. That is, we compared our two transformer-based architectures against the transformer-related benchmarks, and our modified DeepAR network against the existing DeepAR benchmark. We used the PSML dataset repository, plus other code packages' implementations (pytorch-forecasting) of our chosen models. For optimizers and loss functions, we typically used the suggested defaults in the libraries we used for implementation.

3.1. Spacetimeformer

One of the approaches we tried for 1-hour ahead load power prediction using a spatiotemporal transformer model called Spacetimeformer [4].

Multivariate time series forecasting (MTSF) is a challenging task that involves predicting future values for multiple variables based on historical data. State-of-the-art methods for MTSF rely on either sequence-to-sequence models or graph neural networks (GNNs). Sequence-to-sequence models are able to learn temporal relationships between variables, but they do not consider spatial relationships. GNNs are able to learn spatial relationships between variables, but they do not consider temporal relationships.

The Spacetimeformer model is a novel approach to MTSF that combines the strengths of sequence-to-sequence models and GNNs. Spacetimeformer represents the MTSF problem as a spatiotemporal sequence, where each Transformer input token represents the value of a single variable at a given time. This allows Spacetimeformer to learn interactions between the information jointly along this extended sequence. Figure 1 shows the comparison between only (a) temporal attention, (b) hard-coded spatial graph with temporal attention, and (c) spatial-temporal attention.

Similar to many Transformer models, the model consists of three main components: a spatial encoder, a temporal encoder, and a decoder. The spatial and temporal encoders are both implemented using Transformer blocks. The decoder is implemented using a Transformer decoder with attention between the spatial and temporal representations. Figure 2 shows a one-layer encoder-decoder architecture for Spacetimeformer.

What we experimented with for Spacetimeformer is to represent the load power prediction problem as a spatiotemporal sequence, where each input token represents the value of the load, wind, and solar power at a given time for the Spacetimeformer model.

The model is adopted from "Spacetimeformer Multivariate Forecasting" Github repo [5] and we utlized pytorch-lightning library for training/validating and testing.

The reason we think Spcetimeformer would succeed it's that due to its architecture, it learns both temporal and spatial attention and performs well on long-term time series forecasting. First 120 minutes of the load, wind, and solar power history for each batch from the PSML dataset described above is used as the input "Context points" to forecast 1 hour-ahead load "Target point".

Following the PSML experiments' train/validation/test split, we use the first 11 months of minute-level energy data for training and validation with train_valid_ratio=0.8 and December data each year (2018 to 2020) for testing.

The model has trained for 20 epochs and used "RMSE" as the loss function, as in the PSML paper.

We ran the hyper-parameters grid search to decide the best-performed hyper-parameter set. The hyper-parameters and their final values are listed below: batch size (180), learning rate (1e-2), length of the input time series (context points, 120), Transformer embedding dimension (5), Transformer encoder and decoder layers (2), and the number of self-attention heads (2).

Unfortunately, due to limited computing resources, we had to significantly reduce the model size (trainable parameters) from about 222k to 4.9k.

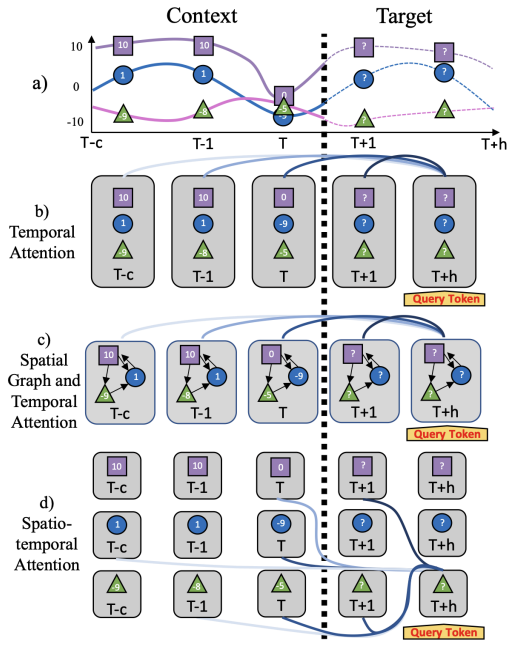


Figure 1. Attention in multivariate forecasting (from Figure 1 in [4])

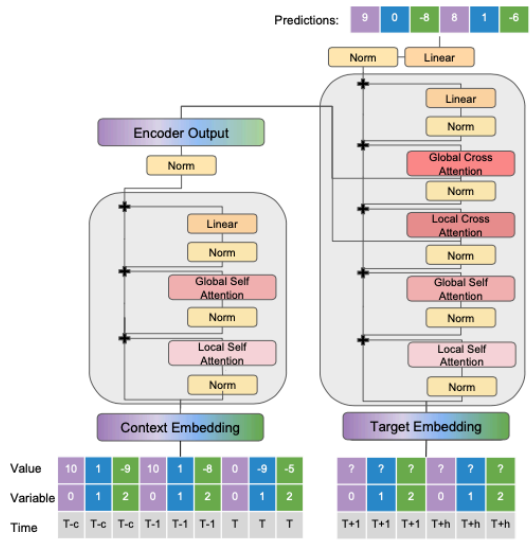


Figure 2. The Spacetimeformer architecture (from Figure 3 in [4])

3.2. DeepAR

DeepAR is a forecasting model based on autoregressive recurrent networks that learns a global model from all During training (Figure 3, top), the inputs to the network are the covariates, target value at the previous step and the previous network output. The network output is then used to compute the parameters of the likelihood which is used for training the model parameters. During prediction (Figure 3, bottom), the history of the time series is fed in for $t < t_0$, then in the prediction range a sample is drawn and fed back for the next point until the desired prediction length is completed.

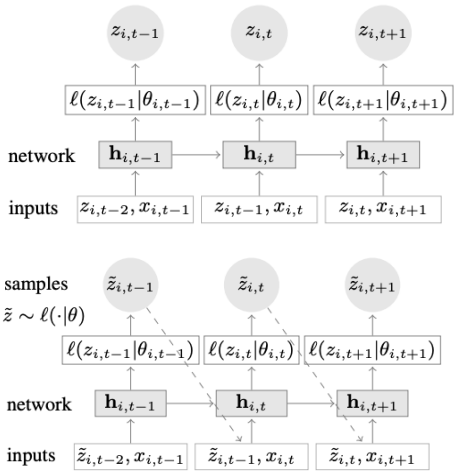


Figure 3. DeepAR Process (from Figure 2 in [6])

The dataset was processed using Zheng et al's provided PSML code library (repo linked in [1]). Additional lag features (lag of t = 1) were added for DHI, DNI, GHI, Dew Point, Solar Zenith Angle, Wind Speed, Relative Humidity and Temperature. LSTM was selected as the cell type and a mutivariate normal distribution was selected as the loss function, per Pytorch Forecasting recommendations.

The optimal learning rate was selected via Pytorch Forecating's native learning rate tunter. Native Pytorch Forecasting hyperparameter tuning was not available for DeepAR so an interactive grid search was performed. Their final values were as follows: batch size (128), learning rate (0.0079), encoder size (24), prediction window (24), hidden state size (60), number of recurrent layers (2), and dropout (0.1).

3.3. Temporal Fusion Transformer

The Temporal Fusion Transformer is an attention-based model designed for multi-horizon forecasting[7]. Previous deep learning approaches to this forecasting challenge relied mostly on black box methods with limited interpretability. TFTs incorporate several unique architectural features that make them well-suited to predicting results at multiple different future time steps:

1. Encoding context vectors for use throughout the network
2. Gating mechanisms filter out irrelevant information from future predictions
3. Multi-head attention integrates information from any past time step to future predictions

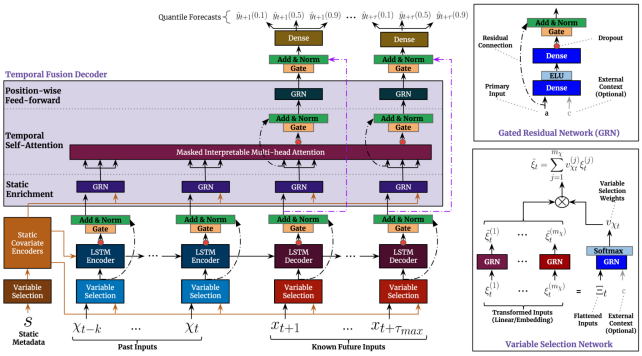


Figure 3.TFT architecture (from Figure 2 in [4])

By incorporating self-attention mechanisms to learn long-term dependencies, TFTs solve one of the problems with using deep learning-based time series forecasting expressed by Zheng et al in the original PSML paper. As such, we felt that it was a promising model architecture to test against the PSML dataset.

We first preprocessed the dataset using Zheng et al's provided PSML code library (repo linked in [1]). Afterwards, we used the pytorch-forecasting to implement the TFT[2], sorting our features based on their intrinsic characteristics so that they play the correct role within the TFT model. As in our other experimental architectures, we used hourly datapoints, forecasting load 1 hr ahead. This helped us speed up model training and parameter tuning, without a noticeable sacrifice in performance. In total, we wound up with around 15,700 trainable parameters.

For hyperparameter tuning, we used Optuna[8] to more efficiently search the space of possible hyperparameters. Their final values were as follows: batch size (128), learning rate (0.0275), hidden size (11), hidden continuous size (9), dropout (0.2), and the number of self-attention heads (1).

3.4. Results

Overall, we wanted to test whether our approaches improved upon the models comprising the PSML benchmarks. Given our limited time and scope, we didn't base our success on achieving the lowest error across all models. Instead, we sought to improve on the benchmarks for models comparable to the ones we tested.

The experiment result for Spacetimeformer is shown in Table 1. We can see that even with so few trainable parameters, the result based on RMSE and MAE is comparable to the other models experimented in the PSML paper. Although still with a lot of room to improve, The model is definitely promising for timeseries forecasting if its full computing potential is unlocked. Figures in Appendex shows the training and validation learning curve. From the plot we see that the loss converges within the first couple epochs. No obvious overfitting problem is observed.

While the DeepAR did not surpass the benchmark ETS model, it offered improvements compared the PSML DeerpAR implementation, with an average decrease in RMSE and MAE of 17.78% and 19.18%, respectively. This is likely due to both the encoder length and prediction horizon being tuned as hyperparameters, whereas the PSML implemented these as static values and tuned hyper parameters such as batch size and learning rate. Overall, the tuned DeepAR seemed to be approply fit, as evidenced by the learning curves in the appendix. Though it should be noted that there appears to be slightly instability throughout the training process. This is likely due to limiting the number of training and validation batches (equal to the batch size) required for an epoch to be complete which was done to expedite training time due to computational limitations. .

Our TFT implementation, while it didn't quite reach the heights of traditional ETS models, actually improved upon the benchmarks of the PSML-provided transformer-based architectures based on RMSE and MAE. However, the tuned models didn't always outperform the RMSE and MAE of the baseline models, which simply repeats the last observed value for each time series in the validation set. This shows that there is definitely room for improvement in our model implementation. Our TFT model did not suffer from major overfitting problems, as the validation set appears to perform similarly to the training set. Overall, the results suggest that transformer-based models adapted for the correct domain may be powerful tools for forecasting, just as they've become indispensable tools for natural language.

4. Challenges

The biggest challenge we faced was with version alignment across the required packages. While the PSML repo is only two years old, deep learning frameworks, and their necessary supporting packages, are constantly being updated. Initial compatibility issues were encountered and exacerbated by cunning the code on different hardware than the PSML package was originally developed for.

Lastly, as we ran two models (DeepAR, TFT) with Pytorch Forcasting utilities vs the PSML code, due diligence was necessary to ensure consistent processes.

5. Future Considerations

For applications where there are several time series cross-sectional units, DeepAR can benefit from being trained jointly over all of the time series[3]. Given that the PSML contains high quality data from 66 locations, investigating the change in performance by including additional locations is of interest[3].

6. Code Repository

The PSML GitHub repository was forked and our contributions can be found at https://github.com/Matt-Carney/Renewable-Energy-Forecasting, most notably in the configs, models, and results

subdirectories located in Code /BenchmarkModel/ LoadForecasting.

Acknowledgments

We would like to acknowledge the researchers at Texas A&M University, University of Southern California, Massachusetts Institute of Technology, and Purdue who have undertaken the noble task of developing the load forecasting dataset and benchmark models for the advancement of machine learning within the renewable energy space.

References

[1] X. Zheng, N. Xu, L. Trinh, D. Wu, T. Huang, S. Sivaranjani, Y. Lui, L. Xei "A multi-scale Time-series Dataset with Benchmark for Machine Learning in Decarbonized Energy Grids", arXiv:2110.06324 [cs.LG], May 2022

[2] TFT implementation from pytorch-forecasting. https://pytorch-forecasting.readthedocs.io/en/stable/api/pytorch_forecasting.models.temporal_fusion_transformer.TemporalFusionTransformer.html

[3] DeepAR implementation from pytorch-forecasting. https://pytorch-forecasting.readthedocs.io/en/stable/api/pytorch_forecasting.models.deepar.DeepAR.html

[4] J. Grigsby, Z. Wang, N. Nguyen, Y. Qi, "Long-Range Transformers for Dynamic Spatiotemporal Forecasting", arXiv:2109.12218 [cs.LG], Mar. 2023

[5] Spacetimeformer implementation Github repo https://github.com/QData/spacetimeformer

[6] D. Salinas, V. Flunkert, J. Gasthaus, "DeepAR: Probabilistic Forecasting with Autoregressive Recurrent Networks", arXiv:1704.04110 [cs.AI], Feb. 2019

[7] B. Lim, S.O. Arik, N. Loeff, T. Pfister, "Temporal Fusion Transformers for Interpretable Multi-horizon Time Series Forecasting", arXiv:1912.09363 [stat.ML], Dec. 2019

[8] T. Akiba, S. Sano, T. Yanase, T. Ohta, M. Koyama, "Optuna: A Next-generation Hyperparameter Optimization Framework",arXiv:1907.10902 [cs.LG], Jul 2019

Table 1. Performance on 1-Hour Ahead Load Point Forecast

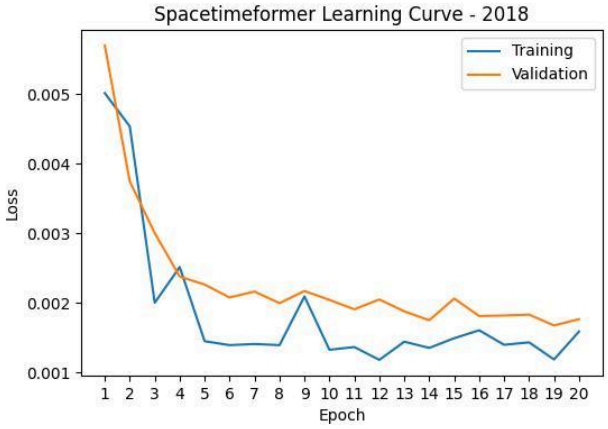
	RMSE (Benchmark Set)			MAE (Benchmark Set)		
Model	2018	2019	2020	2018	2019	2020
Exponential Smoothing	0.029	0.026	0.022	0.021	0.019	0.017
DeepAR	0.068	0.110	0.093	0.055	0.086	0.078
Transformer	0.108	0.121	0.132	0.086	0.095	0.101
Informer	0.129	0.110	0.073	0.102	0.086	0.059
	RMSE (Test Set)			MAE (Test Set)		
Modified DeepAR	0.089	0.072	0.061	0.068	0.059	0.051
Spacetimeformer	0.181	0.329	0.182	0.177	0.320	0.169
Temporal Fusion Transformer	0.087	0.067	0.096	0.077	0.054	0.077

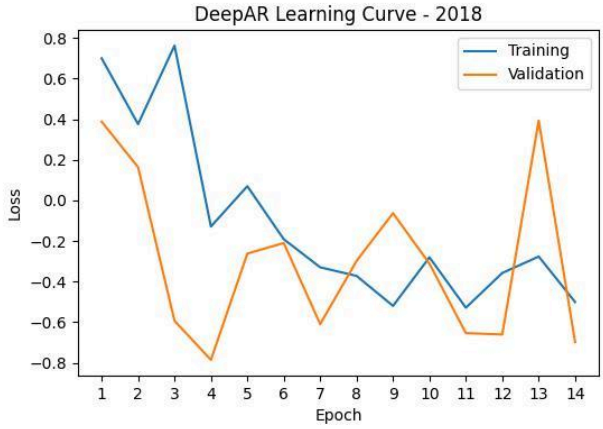
Table 2. Contributions of Team Members

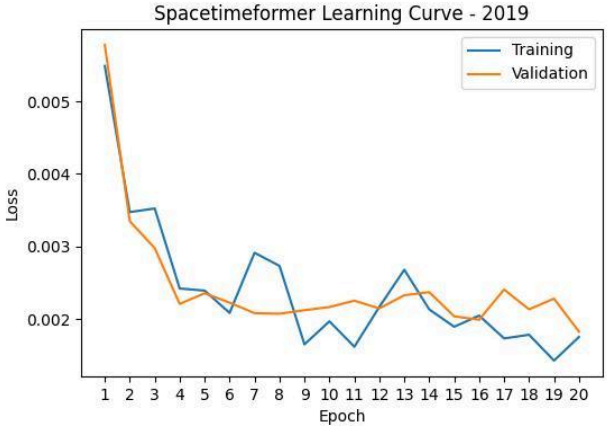
Student	Contributed Aspects	Details
Chang Sun	TFT implementation, experimentation, analysis	Experimented with different methods to implement a temporal fusion transformer with the PSML dataset. Ran parameter tuning and logged experimental results. Wrote up analysis and takeaways.
Pei-Chuan Chao	Spacetimeformer implementation, experimentation, analysis	Experimented with different methods to implement Spacetimeformer model with the PSML dataset. Ran parameter tuning and logged experimental results. Wrote up analysis and takeaways.
Matt Carney	DeepAR implementation, experimentation, analysis	Experimented with different methods to implement Deep AR model with the PSML dataset. Ran parameter tuning and logged experimental results. Wrote up analysis and takeaways.

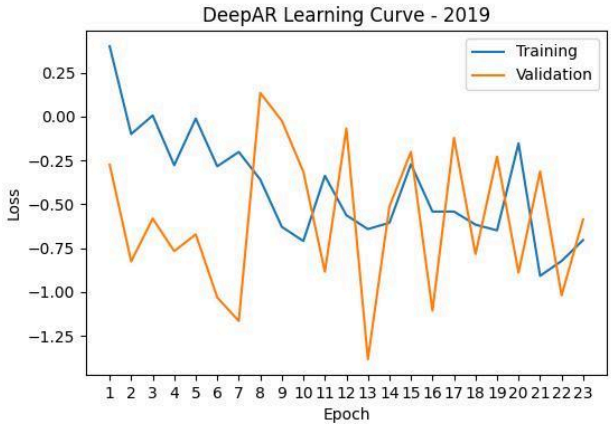
Appendix

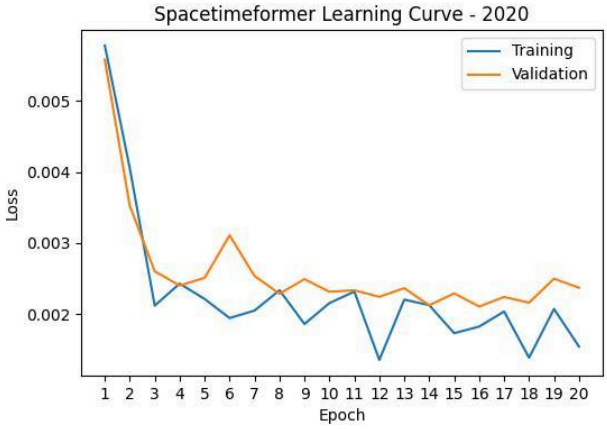
Learning Curves

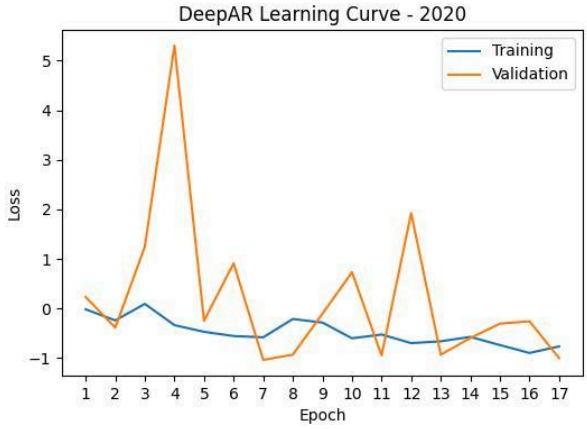


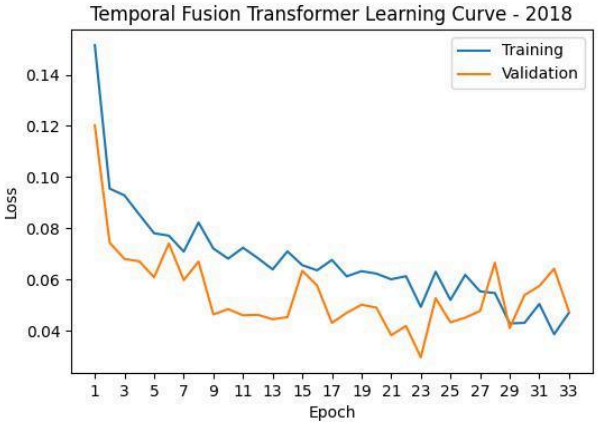


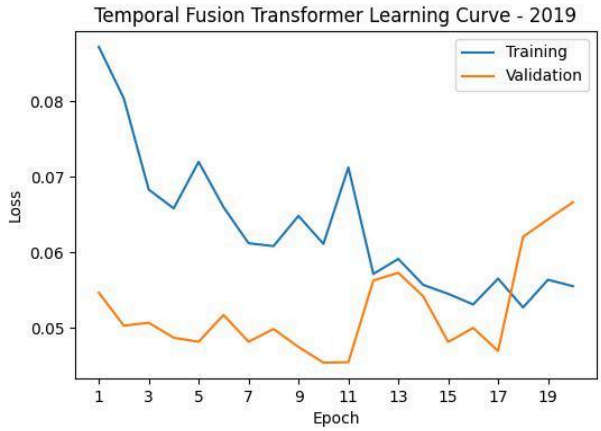


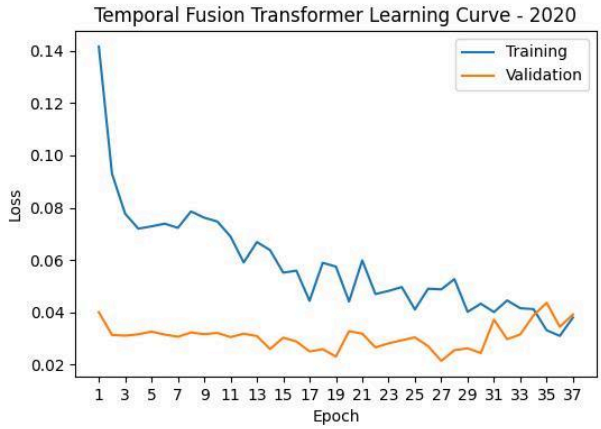







Temporal Fusion Transformer Learning Curve - 2018
Training
Validation
Loss
0.14
0.12
0.10
0.08
0.06
0.04
1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31 33
Epoch


Temporal Fusion Transformer Learning Curve - 2019
Training
Validation
Loss
0.08
0.07
0.06
0.05
1 3 5 7 9 11 13 15 17 19
Epoch


Temporal Fusion Transformer Learning Curve - 2020
Training
Validation
Loss
0.14
0.12
0.10
0.08
0.06
0.04
0.02
1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31 33 35 37
Epoch