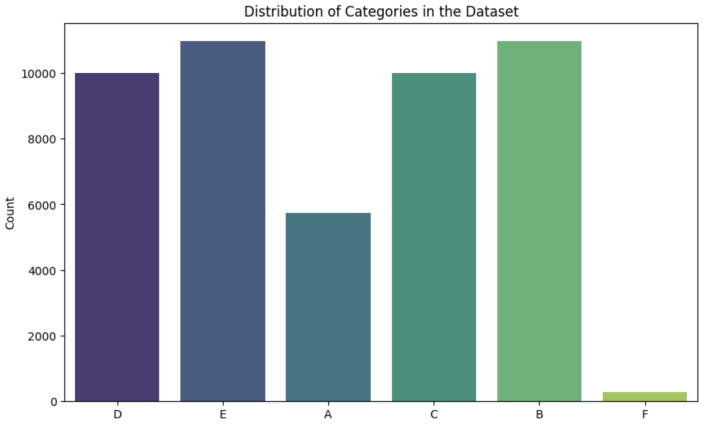
Artificial Neural Network and Deep Learning Time Series Forecasting Homework

Fatih Temiz (10901682), Ecem İzde Kaya (10902759), Hessam Hashemizadeh (10900041)
Team Name on Codalab: "LeCunTanh"

Exploratory Data Analysis

The dataset comprised three numpy arrays: one containing the actual time series data, another indicating valid time slots for each series, and a third specifying the category corresponding to each series. Upon plotting the distribution of time series data based on each category, a notable observation surfaced—imbalance. Class F exhibited significant underrepresentation, while class A was only half as prevalent as the other classes. Initially, we considered employing class weights during training, but we ultimately opted not to pursue with this approach.



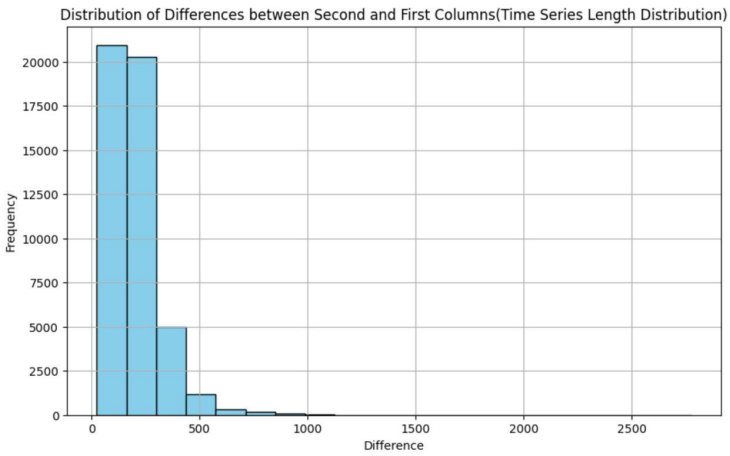
Splitting Training, Test and Validation data sets

We partitioned the data into distinct sets, allocating 15% for testing, 15% for validation, and 70% for training. It's noteworthy that this split was done based on the categories. To ensure category-wise consistency, we made a dictionary for each category, satisfying the respective proportions within the specified test, validation, and training sets. This approach guaranteed a balanced representation of split across the different category sets.

Data Preprocessing

We observed variations in the lengths of time series, with some sequences being notably short. To address this, we implemented a data replication strategy to ensure a more balanced and representative dataset.

We replicate the data for series shorter than 200 units to exceed this window size by at least 20 units. This process involves calculating a replication factor and adjusting the series' start indices accordingly. Additionally, the build_sequences function transforms these time series into structured sequences. It checks the 'row_size % window' then zero pads sequences. We also expanded the dimension since we had a single feature (univariate time series).



Initially, we experimented with models incorporating autoregression, specifically with parameters such as autoregressive_telescope set to 1 and autoregressive_telescope set to 3. However, we found that this approach resulted in a suboptimal solution. The model struggled to learn for longer sequences, leading to an increase in the overall Mean Squared Error (MSE). As a result, we opted to explore alternative strategies for better performance.

With our data preprocessing strategy we increased data size dramatically which reached 360K time sequences each having 200 length. This gave us better results in the final.

We refrained from applying additional scaling methods, such as Robust or MinMax, as we didn't observe any noticeable improvement in performance by doing so.

We experimented with training using Mean Absolute Error, but ultimately found that compiling with Mean Squared Error yielded better results in terms of model performance.

Models

We experimented with several models, starting from Convolution & BiLSTM, 1DCNN, 1DCNN Resnet, FCN, Seq2Seq Attention, and finally Time2Vec BiLSTM. Our primary emphasis was on CONV-LSTM and Time2Vec-BiLSTM. We found that BiLSTM was more suitable for our data, given the relatively large telescope size. Consequently, we focused on architectures that incorporated BiLSTM, choosing to discard 1DCNN, FCN, and Seq2Seq with attention models. We know summarize our experiments with these two in the following section

Conv-BiLSTM: We evaluated the initial basic model introduced during the lab sessions, which combines bidirectional LSTM, Conv1D, and Attention layers. The bidirectional LSTM processes data in both directions, capturing long-term dependencies. The Conv1D layer extracts local features, identifying key patterns within the data. The Attention layer allows the model to concentrate on the most crucial parts of the input sequence. This model demonstrated effective performance, especially as the dataset size increased.

Time2Vec+BiLSTM (Final Model): As documented in the logbook on the drive, our work involved implementing the Time2Vec layer in our model. This layer plays a crucial role in effectively encoding temporal patterns by capturing both linear and periodic dependencies inherent in time series data. It enhances the model's capacity to comprehend complex temporal dynamics. The Bidirectional LSTM layers further augment this capability by processing the data in both forward and backward directions, capturing long-term dependencies from both past and future contexts. Conv1D layers are employed to extract localized features within the time series, such as short-term spikes or drops.
The attention mechanism within the model is designed to focus on specific segments of the time series that are more relevant for the forecasting task. It can selectively concentrate on

certain parts of the data when making predictions. Finally, the model's output is fine-tuned to match the forecast horizon through additional Conv1D and Cropping1D layers, adjusting the output size to the telescope dimensions. This ensures that the predictions align with the desired temporal dimension. The model is compiled with the Mean Squared Error loss function and the Adam optimizer.

Category Based Ensembling

Initially, we attempted to include categories in the training data, but this approach didn't yield satisfactory results. Therefore, as mentioned in the training, validation and tests sets split section, we decided to divide the dataset based on categories during the training phase. This led to six distinct sets for training, validation, and testing. Our strategy involved training individual models on category-specific data and subsequently combining these models using attention layers to form an ensemble.

We have combined outputs from various models that were pre-trained on category-specific data. Each model in the ensemble processes input independently, and their results are joined together. An attention mechanism is applied to this combined output, where a Dense layer with softmax activation learns to assign importance scores (attention probabilities) to each model's output. These scores are then used to give more weight to the outputs of the more relevant models. The weighted outputs are combined into a final output through another Dense layer. We discovered that this is the most effective approach to incorporate category information into this problem. In the end, we created an ensemble of category-based CONV-BiLSTM and Time2Vec-BiLSTM models, but unfortunately, it didn't lead to a significant improvement in Mean Squared Error (MSE).

Hyperparameters

We used these parameters for training our best model.

Batch Size	Epochs	Min Learning Rate	Optimizer	Callbacks
64	100	10^-5	Adam	earlyStop and Learning Rate Scheduler

Results

In this table we compare the different MSE and MAE that we derived from our different models in the second phase of the challenge on Codalab.

Model	MSE	MAE
Conv_LSTM	0.013549977913	0.083983823657
Category Based Conv- BiLSTM	0.012339896522	0.080902226269
Category Based Time2Vec BiLSTM	0.010779700242	0.075902596116
Ensembled Conv-BiLSTM & Time2Vec BiLSTM	0.010715750046	0.075265273452

References

[1907.05321] Time2Vec: Learning a Vector Representation of Time (arxiv.org)

Hands-On Advanced Deep Learning Time Series Forecasting with Tensors | by Dave Cote, M.Sc. | Medium

Temporal Convolutional Networks and Forecasting - Unit8

Understanding masking & padding | TensorFlow Core

Contributions

All group members worked together beside each other during the working time, and each member had an equal contribution to the project.