Artificial Neural Networks and Deep Learning Homework 2

Team: **TheLearningLegion4**
Camille Buonomo (10978972), Simone Gabrielli (10715031),
Andrea Ricciardi (10931392), Andrea Zanin (10833512)

Dataset

The dataset provided for the challenge contained 48000 monovariate time series of different length, the maximum one being 2776 time steps long. The dataset was divided in three parts : the padded data which contained the 48000 time series 0-padded to length 2776, the range to consider for each time series in the padded data and the categories of the time series ranging from A to F. Each time series was normalized between 0 and 1. The task was to forecast time series in a range of 18 time steps in the future. For this purpose, we used the range file to extract from the padded data the original time series.

Data preprocessing

Since each time series had a different size, we splitted each time series in sequences of the same length using a sliding window approach; we chose a stride of 10 steps to limit redundancy in the dataset while still training the model to predict all the possible steps. We also tested various window lengths and we determined that 100 steps was long enough to provide all the relevant information to the model, longer windows just lead to model overfitting.

We encoded the categories with one-hot encoding and concatenated this encoding to each time series window.

We used 9% of the time series as validation set and 10% as test set; we performed this split before the windowing procedure, in this way we ensured that no part of the validation and test set was also present in the training set.

TimeGAN

We used a GAN to generate new time series to augment our dataset. We just created samples of categories A and F, because they were the least represented. We used a dataset, for training the GAN, the padded data because the input and the output had to have a fixed size. For the F-category time series we also removed the outliers, which were the time series that started before the 2200 time step. We used as a generator a network mainly based on Conv1D layers and as a discriminator, one that is mainly based on LSTM layers. This has been done because the discriminator needs to use the correlation between different time steps to better understand if a sample is fake or not. During the training of the GAN, for each batch, we trained the generator 5 times and the discriminator only once to guarantee the convergence of the losses. We monitored the training by plotting, for each epoch, 5 random samples generated by the GAN. After the training, we generated 2000 samples for category A and 200 for category F, which were processed to make them more similar to the real samples. The processing consisted in : choosing a random number “N” between 2400 and 2500 and setting the values of the generated time series to 0 in the range 0:N. We also applied a gaussian smoothing filter to the data to denoise it. The provided dataset was then extended with our generated samples.

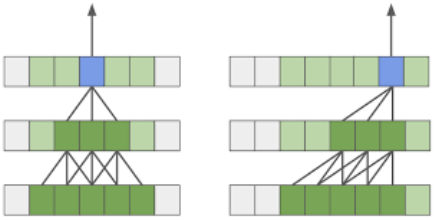
Model

We decided to use an auto-regressive architecture with a ResNet50-inspired base model. We then improved the performance of this model by introducing an attention mechanism.

Base model

We splitted the input into sequence and encoded categories using Lambda layers. We connected the sequence input layer (without the category) to an augmentation layer, which added gaussian noise to the sequence. After this layer, we implemented a ResNet50 like structure with 3 residual blocks composed as follows : 2 Conv1D layers, 1 batch normalization layer, 1 ReLU activation and 1 add block which sums up the output of the ReLU activation and the input of the first Conv1D layer.

For the Conv1D layers we have used causal padding, which is more suited for time series forecasting; indeed it ensures that the filter only considers past and current values of the time series, preventing information leakage from future values into predictions. It is essential for maintaining the cause-and-effect relationship in time series forecasting.



To increase the receptive field of the CNN and to prevent overfitting we added a Max Pooling 1D and Spatial Dropout 1D layer after the first residual block, we also added a spatial dropout layer after the second residual block.

After the CNN we added a multihead attention layer using as a query the last item in the CNN output (which is the representation of the last part of the time series) and as key and value the whole CNN output. This mechanism allows the network to pay attention to the parts of the time series that are similar to the end of it, this is useful because those are the ones with relevant information for forecasting. We used specifically a multihead attention layer to allow the network to focus on different patterns at the same time, since each head captures different temporal patterns.

The last part of the network is a Multilayer Perceptron with 2 layers, its output is a prediction of the next 6 steps in the time series. This MLP receives as input the attention layer output, the category and the last 100 steps of the unprocessed time series (this works similarly to the skip connections in the ResNet architecture).

Full model

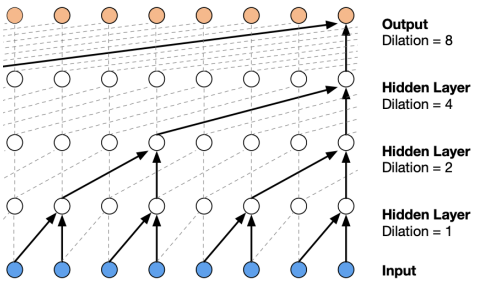
To produce the full 18-steps forecast we use an auto-regressive approach: the base model 6-steps output is appended to the time series and this extended time series is used as input to predict the following 6 steps; with the same approach the final 6 steps are predicted.
The base model has a CNN followed by an attention layer with fixed query size, so it can receive an input series of different lengths; this allows it to receive 100 steps at the first iteration, then 106 and then 112.

Predicting fewer steps at each iteration can lead to better predictions because the attention output is computed more times thus allowing the network to shift attention more times, however it makes the

overall network deeper and thus harder to train. We found 6-steps increments to be a good balance of the two effects.

Trial and exploration

A trial that we have done was to implement a Temporal Convolutional Network (TCN), a neural network architecture designed for sequence modeling. It uses 1D causal convolutions along with an increasing dilation rate when going deeper in the network, in order to expand the receptive field efficiently. They can capture both local and global context, making them effective for various applications. However it did not improve our performances a lot and we opted for the attention mechanism only.



We tried to implement a separation of the times series into 2 different time series, one was the trend and the other the seasonality. The idea was to have 2 forecasting models, one for each time series and then add both outputs so that each model had a simpler task to learn. There are many different ways on how this can be done. We implemented several of them, but overall the results were always worse than using a single model so we didn't keep that idea.

Another tool we tried to use was autocorrelation. It's a statistical quantity that quantifies how much a signal is correlated to itself at different time steps. It's a very efficient tool to find the periodicity of a signal. Usually, it has to be computed on a 0 mean signal, but since there is no implemented TensorFlow function to compute it, we instead trained a model to return the autocorrelation of the input signal. Then, by viewing the autocorrelation as a sort of attention, we predicted the next value in an autoregressive fashion by taking the scalar product of the autocorrelation and the signal. We used another model to predict the amplitude of the predicted value since it is not information that correlation can provide. The results were promising when hand tuning the amplitude, but the generalization was not really efficient and thus we abandoned that idea.

We also tried to use the fourier transform (FFT), which is another way to introduce the periodicity of the signal inside the model. We used a CNN plus a LSTM layer to extract the interesting feature of the FFT and added it to the model by multiplying the extracted feature with the predicted values given by another model. This method was performing better than the one using autocorrelation, but not more than the one with ResNet architecture.

We also tried to implement the autoregression mechanism bypassing the CNN layers: we fitted a network to predict the following item in the CNN output and used this predicted CNN output as input for the attention layer and the final time series forecasting layer. This approach was faster, since it skipped several passes through the CNN, but it had bad performance so we discarded it.

Finally we also tried to implement six models in parallel that worked on the six different categorties, using as base model a deep one inspired by the ResNet50 structure. However the complexity of the model was too high and the ability to generalize was very bad, hence we tried to use all the data to train this ResNet50 like model, but still the performance was bad, so we discarded this model and we just kept the idea of implementing a ResNet50 like structure, but simplifying it.

Contribution

- Camille Buonomo (10978972) : He looked at how to directly introduce the periodicity of the time series inside the model rather than only learning it. For that, he looked at the autocorrelation and the Fourier transform. Both of the methods gave better results than the "classic" RNN models but they were worse than the ResNet like model. He also tried to separate the learning into 2 phases, one for trend and one for seasonality but the results were not promising and the training longer so the idea was not kept.
- Andrea Ricciardi (10931392) : He initially tried to train 6 different models (one for each category) by using a deep ResNet50 like structure, but the results were bad due to very high overfitting (too complex models with not enough data). For this purpose he implemented the TimeGAN, slightly improving the result, but the performance was still bad due to the too high complexity of the models. For this reason he used his model with all the data (of each category and also the generated ones). This improved the performance a lot, but still there was high overfitting. For this purpose he decided to use an autoregressive model, by also including the category encoding, which was inspired by the ResNet50 structure, but this time with a simpler structure, noticing that this was the best model so far. The model is the one mainly described in the report. He even tried to apply the Robust Scaler to the time series, not achieving good enough results.
- Simone Gabrielli (10715031) : He introduced the concept of causal padding in the Conv1D layers, better performing for time series forecasting. He then tried to apply the idea of temporal convolutional networks, using dilation in the layers. Even though it was performing better, the idea was not kept because we went to the introduction of attention mechanisms in the network, which was better than "tcn" . He also tried to preprocess the data with RobustScaler, which takes into account also outliers removal, but the training procedure was too long and the advantages not so remarkable, so we decided to not use it.
- Andrea Zanin (10833512): He implemented the data preprocessing, including the code for the windowing and the choice of representation of the categories. He implemented the structure for the autoregressive models, which is structured with a base model called several times within the full model; this implementation allowed us to train directly the full model, instead of training the base model on shorter sequences. He also introduced the attention mechanism in the model, first trying Luong Attention and then settling on the Multihead Attention. He implemented the autoregressive model which bypassed the CNN, but this had bad performance. He added gaussian noise to the model to limit overfitting.

References

-causal padding and dilation
https://unit8.com/resources/temporal-convolutional-networks-and-forecasting/

-GitHub - proceduralia/pytorch-GAN-timeseries: GANs for time series generation in pytorch

-GitHub - benearnthof/TimeGAN: A pytorch implementation of Time-series Generative Adversarial Networks (https://github.com/jsyoon0823/TimeGAN)

-GitHub - numancelik34/TimeSeries-GAN: Generation of Time Series data using generatuve adversarial networks (GANs) for biological purposes.

-[2202.02691] TTS-GAN: A Transformer-based Time-Series Generative Adversarial Network (arxiv.org)

-https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.RobustScaler.html

-https://www.tensorflow.org/guide/ragged_tensor?hl=it#keras