Artificial Neural Networks and Deep Learning Course First Challenge
Report

Time Series Forecasting

Submitted by

Mohammad Amin Abbaszadeh (10831781)
Codalab ID: maaz

Riccardo Tripodi (10620589)
Codalab ID: trepodi

Maksim Beliakov (10913275)
Codalab ID: mksmblkv

Alessandro Perricone (10661417)
Codalab ID: AlePerry

Date: Autumn 2023

Polytechnic University of Milan

Introduction

In this challenge, it was required to train a deep learning model able to forecast future time steps given an input sequence. In particular the test sequences were made up of 200 time steps and the model had to forecast 9 and 18 time steps respectively in Phase 1 and Phase 2 of the competition. The training data was made up of 48000 padded sequences with 2776 time steps. Additional information was given in order to retrieve the orginal un-padded sequences. Moreover, every sequence was assigned to a categorical class. In total there were 6 classes whose meaning was undisclosed.

Training Dataset

Once understood the problem we were facing, we had to build the training dataset reformulating the problem as a supervised learning problem with an input sequence and a sequence of targets to predict. It was possible to do it sliding through the un-padded data and selecting a choosen amount of consecutives time steps as the input sequence and the target sequence. Consequently, this process depends on three hyperparameters: *window*, *stride*, and *telescope*. The *window* is the length of the input sequence, *stride* is the number of steps separating the beginning of two consecutive sequences and *telescope* is the length of the target sequence. The *telescope* was set to 9 since it is a submultiple of the final prediction sequence. In order to set the two remaining parameters we considered the autocorrelation of the sequences. In particular, for each category we computed after how many time lags the autocorrelation value decreased below a threshold. Since we noticed that, considering one standard deviation, 80 percent of the correlation was within almost 200 steps we set *window* to 200 and *stride* to 50. Tests with other values were performed.

Preprocessing

Each sequence in the input data was already normalized between 0 and 1. Two additionally preprocessing were considered: robust scaling and a Fourier filter. *RobustScaler* is a scaler in *scikit-learn* library that robustly scales features by using the median and interquartile range (IQR), making it less sensitive to outliers compared to standard scaler. It subtracts the median and divides by the IQR for each feature. This is useful when outliers can affect traditional scaling methods. The Fourier filter removes from the spectrum of each sequence the frequencies associated with the lowest squared amplitudes keeping constant the percentage of remaining cumulative sum of the squared amplitudes. None of the two preprocessings led to significant improvements in the forecasting task so the final models did not include them in the training pipeline.

Models

Baseline

Before proceeding further it is important to have a baseline. We made 2 baseline models, a constant and a linear model. The constant model assigns to the whole output sequence the last value of the input sequence. The linear model is made up of two non-linear dense layers to directly map the input sequence to *telescope* output values. They achieve a mean squared error on the validation dataset of 0.012 and 0.005.

Meanwhile, it was tried to take advantage of the auxiliary categories. This idea was practiced by using a learnable embedding layer. The integer encodings of the category labels are fed as input to the model alongside the input time series. After extracting the embeddings, they contribute to calculate the final output of the model by concatenating them at some points in the model. Similarly, in order to improve prediction performances on minority categories, we trained some models assigning different weights in the loss function for samples belonging to different categories. Finally, a satisfying improvement in the results was not observed which leads us to use more novel approaches that are elaborated in the rest of the document to deal with the problem.

Inception-ResNet

Given the proven success of CNNs as feature extractors, we tried to train a 1D-CNN and predict the output sequence on the latent representation of the convolutional network. Since in a time series the relevant features can have long range dependencies we thought of using Inception-like convolutional modules with filters of various sizes (1x1, 3x3, 5x5, 7x7) in parallel. This allows the network to capture features at multiple scales in parallel, enabling better representation of both fine and coarse details in the input data. The use of 1x1 convolutional bottlenecks in the architecture helps reduce the number of parameters. Additionally, in each parallel path there are skip connections to allow a better gradient flow. The model reached a mean squared error of 0.006 on the validation dataset. This result underlines both the surprising expressivity power of the linear model and the necessity to adopt a model intrinsically able to handle sequential data.

Encoder-Decoder LSTM

LSTM (Long Short-Term Memory) networks are well-suited for sequential data due to their ability to capture long-term dependencies, facilitated by a sophisticated memory cell structure. Unlike traditional RNNs, LSTMs incorporate gating mechanisms that regulate information flow, allowing selective updating and retention of crucial data. This mitigates the vanishing gradient problem and facilitates effective learning of patterns in diverse sequential data types, such as time series.

An encoder-decoder architecture with LSTM is a neural network design commonly used for sequence-to-sequence tasks. In this architecture, the encoder processes the input sequence into a fixed-size context vector, capturing the input's semantic information. LSTMs in the encoder help capture long-range dependencies. The context vector serves as a condensed representation of the input sequence. The decoder, also equipped with LSTM cells, then generates the output sequence step by step, using the context vector to guide the generation process.

In order to fully exploit the architecture, two additional enhancements have been adopted: the attention mechanism and teacher forcing. The attention mechanism allows the decoder to focus on different parts of the input sequence dynamically during the decoding process. Instead of relying solely on a fixed-size context vector from the encoder, attention assigns different weights to different parts of the input sequence at each decoding step. This enables the model to selectively attend to relevant information, improving its ability to handle long sequences and capture intricate relationships. Teacher forcing is a training technique that involves using the true target values (ground truth) from the training data as inputs during the training process, instead of using the predicted values from the previous time step. In the context of sequence-to-sequence tasks teacher forcing helps stabilize and accelerate training by providing more accurate and reliable feedback to the model during the early stages of learning. It aids in preventing error accumulation during training and contributes to faster convergence. During inference, the model generates output by recursively using its own predictions from previous time steps as inputs for future time steps. This is in contrast to training, where teacher forcing is employed to facilitate the learning process by providing correct target sequences at each decoding step.

In our architecture the encoder and decoder are made up of a single LSTM. The attention mechanism is a Luong style attention between the decoder output state and the sequence produced by the encoder LSTM and the resulting context is concatenated to the output of the decoder and followed by a dropout layer. The starting element of the decoder input sequence is the last element in the encoder input time series. The model was trained with the Adam optimizer and a learning rate kept constant for 10 epochs and after decreased of 1 tenth at each epoch.

This architecture reached a validation loss of 0.0026. In order to make its predictions more robust during phase 2 where the output window was increased from 9 to 18, it has been made an ensamble of models based on this architecture. In particular, once trained the network on all the training data (using K-fold validation to determine the number of epochs), the final 5 model's checkpoints were saved and used to predict the output of the hidden test set.

Moreover, several models comprising stacks of "LSTM", "Bidirectional LSTM", "GRU", "Conv1D" were built in different topologies and configurations of hyperparameters. However, a substantial im-

provement in the result was not achieved compared to the single LSTM in both the encoder and the decoder.

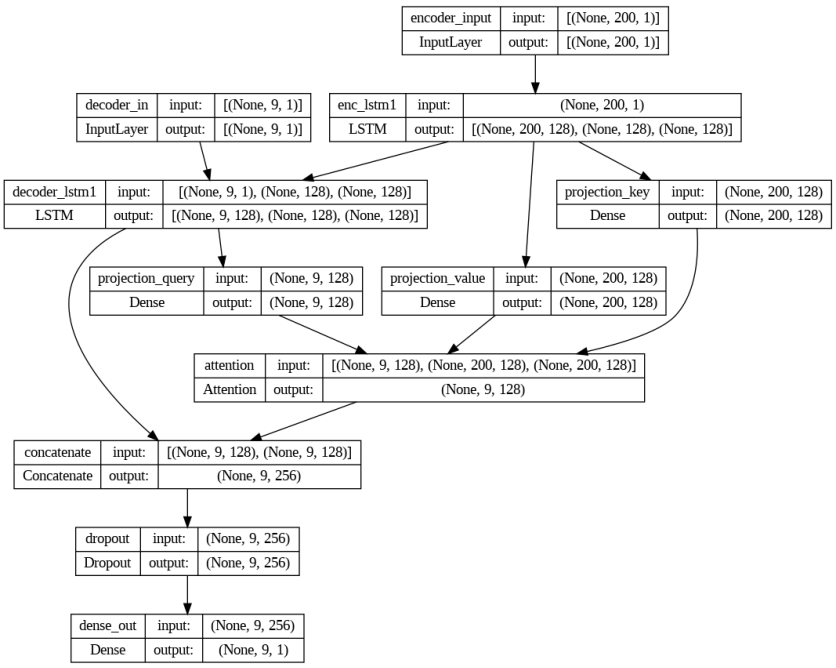

Figure 1: Encoder-decoder LSTM model

Transformer Model

One of the potential solutions which were expected to lead to a remarkable result on the test data is using a model architecture similar to what is proposed in the popular publication named "Attention Is All You Need". Although the model of the paper was defined properly in the context of machine translation, it was a challenging practice to explore it in a time series forcasting task. To this end, an initial model was built conforming to the structure of the reference model. In the beginning, The model was trained by the teacher forcing method, giving the ground truth of labels shifted to the right and concatenating with the last element of the input time series to the encoder. Unfortunately, a perfect result was not achieved during the inference phase on the local test data. This behavior could be rooted in the fact that the model was overfitting. To elucidate more, given the number of 256 as the hidden dimension of the model, a total number of trainable parameters was in the order of 1M which is much higher than the size of the training dataset. To address this issue, the training dataset was augmented using an external library named "tsaug" to augment the data so that the size of the dataset exceeds the total number of parameters. However, as the number of data and parameters grows, the training process slows down remarkably which is not practical given the resources on the Google Colab service. At the next step, the hidden dimension of the model is decreased to have a simpler model to be able to proceed with the training set. Given the dimension of 64, the model was trained properly using the teacher forcing method. It was observed that we could get a valloss of order 1e-5 at the latest epochs of training. Due to the great performance of on validation data, it was expected to get a near result on the local test data, however, the result on the local test data was in the order of 0.01 which was not acceptable. Several approaches were tried in order to address this issue such as removing the decoder part. In the end, none of them yielded satisfying results. Besides, a new solution that comes into mind is that, instead of using ground truth of labels during the training phase, feed the decoder a random noise and let the network learn to generate the output time series in one shot from a random noise. So that, in the inference time the output is not generated iteratively one token each time, in contrast, we generate the whole output in one shot. Generating the dataset with a stride equal to 5 and replicating the time series instead of zero padding a mean square error of order 0.011 is achieved during the second phase of the challenge on the coda lab test set.

Contributions

- **Riccardo Tripodi**: Model architectures and training, model ensemble, iterative decoding, Fourier filter. I state that i disagree using random noise in reference to the last paragraph.
- **Maksim Beliakov**: Brainstorming and training.
- **Mohammad Amin Abbaszadeh**: Transformer models, Augmenting data,Preprocessing time series for insight, some Baseline Models and Resnet.
- **Alessandro Perricone**: Helping with the training, testing, correction and submission of several models that led to the development of the final submission.