AN2DL 2023 Homework 2

Team_FEMS

Fabio Cabeccia - 10936531, Simone Bortolotti - 10935857,
Elahe Houshmand - 10701966, Mohammadreza Javadi Namin - 10893373

December 2023

1 Dataset Specification

For this second assignment we were given a dataset of 48000 time series, each with a different length, but zero padded to reach a common length of 2776 time steps. In addition to the data we were also given two other arrays:

- **Valid Periods** - an array that contains, for each of the time series, their start and end index, which basically the original data without padding.
- **Categories** - an array that contains, for each of the time series, the code of its category, which can be one between A, B, C, D, E and F.

1.1 Dataset Pre-Processing

Our first approach to the presented problem was very trivial and mainly involved a very swift exploration of the given dataset. To achieve that, we basically printed and plotted random elements of the dataset without padding, hoping to understand if any kind of pattern could be found. While doing so, we discovered that not only all the time series were already normalized in the [0, 1] range, but also that there were many outliers in the values of all time series. Thus, we decided to try to tackle the latter using Robust Scaling, with the goal of comparing the performance of models trained using either raw or scaled data. In the end, robust scaling did not improved models' performance and was discarded.

1.2 The Padding Problem

Given that the time series were zero padded on the left, at first we found a baseline result by simply use the data as they were. Then we moved on to other approaches, mainly no padding and interpolation padding.
The no padding route, i.e. removing the zero padding and using the series as they were, was not very successful. Thus, we decided to augment the data and lengthen the time series to 2776 once again, but replacing the zeros with a linear interpolation of the missing values for each time series. The padding technique of choice was pre-padding, since it should guarantee better performance when used along with LSTM models [2].

1.3 Training and validation sets

Knowing that the model would have been tested on the server, we decided to split our dataset into only training and validation sets. In practice, we tried validation splits ranging from 0.1 to 0.05.

2 Framework and Sequences Generation

The framework selected to tackle the forecasting task was autoregression. Basically, at inference-time the model was fed with data obtained by progressively sliding a window of fixed size over the input data, each with its own set of targets. For the strategy, we went with Multi-Input / Multi-Output, where the window $x_1, ..., x_t$ was fed as input, and the values $x_{t+1,...,x_{t+N}}$ were provided as output.

In order to provide data structured in the way defined above, we developed a function able to extract subsequences of fixed size from the dataset, given their length, the future window size and the stride. Some experiment were also made using the keras function Time Series Generator[1], but since it does not allow for multi-step prediction it was quickly discarded in favour of a more flexible handmade function.

Regarding the actual values of the parameters, after some testing we got that the following are optimal for our problem:

- Sequence length: 200
- Stride: 50
- Future window size: 18

In theory, we could have gone for a bigger future window size, but given that we already knew that the test set would have required *at most* 18 samples, we decided to fix it to that value. In addition, since training each model with one time series at a time was not doable due to extremely high training time, we decided to generate batches of sequences made of 60 time series. Basically, this allowed us to have an input shape of (60, 200) instead of (1, 200), and vastly reduced the training time.

3 Development

Here we briefly list our attempts, to highlight the strategy that brought us to the final model. All presented models are stored in the file models.ipynb.
Evaluation. The following models have been evaluated by considering Mean Squared Error (MSE) and Mean Absolute Error (MAE). Each model was tested using a batch size of 256 samples, an upper limit of 300 epochs, and two overfitting reduction techniques: (1) early-stopping with a patience of 18, and (2) learning rate reduction on plateu with a patience of 8 and a reduction factor of 0.2. Note that the epoch limit was never reached.

3.1 Base Models

Given the temporal dependency of the dataset provided and the nature of the problem we had to tackle, we have decided to firstly compare some basic models based on different recurrent units. Starting up by defining simple architectures allowed us to have a lower bound in terms of model complexity that could act as a reference point for next comparisons. The recurrent architectures selected were the following: RNN, LSTM, and stacked-LSTM. In order to make a fair comparison the different models shared the same amount of recurrent-unit (512).

3.2 Convolutions and LSTMs

In order to reach better results, testing a combination of Convolutional Neural Network (CNN) with Long Short-Term Memory (LSTM) architecture proved to be a meaningful endeavor. On one hand, CNNs excel at capturing local patterns and spatial dependencies within temporal data, providing an effective means of extracting relevant features; on the other, LSTMs are adept at modeling the temporal dependencies and capturing long-range dependencies across sequential data points.

To be more specific, we settled with three architectures: a simple convolutional network with a bidirectional LSTM, a base ResNet model, and a more complex model inspired by DeepMind's WaveNet architecture [4], an autoregressive deep generative model used for audio waveforms generation. The latter works around *dilated casual convolutions*, which not only ensure that the ordering in the input sequence is preserved while processing it, but also allow for an increase of the receptive field when they are stacked (with little to no impact in the computational cost). The main block that gets repeated through the network is made of a stack of dilated 1-dimensional convolutional layers (obtained by progressively doubling the number of filters and dilatation-rate), followed by batch normalization and average pooling layers. Stacking a various amount of these blocks in a ResNet fashion we obtained a new network that was tested not only against the previous base models, but also against the much simpler bidirectional convLSTM, made by a Bidirectional LSTM layer followed by two 1-D Convolution layers and a Cropping to crop the output to the desired length. Surprisingly though, the wavenet model was not only outperformed by the bidirectional one, but also performed worse than the base models. In this context, the ResNet model proved to perform better than WaveNet, but worse than the Bidirectional ConvLSTM.

3.3 Seq2Seq and Attention

Following the design characteristics of autoencoders' architecture, we developed a model to address the task by leveraging an encoder-decoder structure. The encoding and decoding tasks were performed by two LSTM recurrent layers, one of which bidirectional. Then, this same architecture was enhanced using the *Luong Attention Mechanism* [3]: first compute the alignment scores a, between the source *s* and the target *t*, that can be referred as our encoded and decoded hidden states

$$a_t = \text{softmax}(\boldsymbol{h}_t^T \boldsymbol{h}_s) \tag{1}$$

and then compute the context vector c as the weighted average of the source states by using attention scores. This was then concatenated with the source state and fed into a fully-connected layer. This new architecture did not although improved the already good performance provided by the Bidirectional Convolutional LSTM, which, in the end, was chosen as our final model.

3.4 Multiple models

Another attempt involved creating a model for each class, aiming to produce not a general model but rather several, more tailored to each individual class and its specific characteristics.

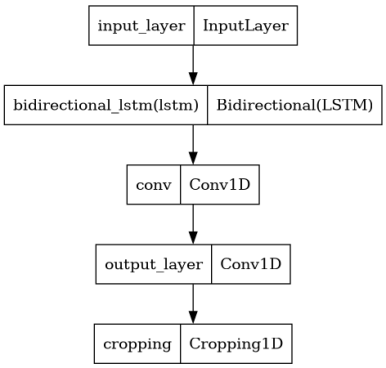
The time series within each class were separately evaluated using three approaches:

- The base model, a Bidirectional Convolutional LSTM.
- The model implementing Seq2Seq and Attention mechanisms.
- The base model trained with all the data, using only data belonging to the specific class for validation.

Although during training classes D and E, in particular, achieved very good results (class E reached validation mae: 0.007), during testing, each combination of sub-model tested proved to be worse than the base model.

4 Final Model, Parameters and Results

As already mentioned, the final model was a Bidirectional Convolutional LSTM, with structure:



Regarding the hyperparameters, multiple tests on the following have been manually performed: input-output sequence length, learning rate, and number of recurrent units. (1)After testing the base models with multiple input window sizes we concluded that being in the range 200÷400 allowed to capture the periodicity of the timeseries while mantaing a reasonable computational load. The output window size was instead set to 18, as it was both the time steps of the set and quite optimal for our multi-output framework. (2) The learning rate was set to 1e-3 for all models (with learning rate reduction up to 1e-7). (3) Multiple values for the amount of LSTM units were tested, but an upper bound of 512 allowed for good performance with negligible increment in computational costs.

In conclusion, this final model was also tested using the time2vec representation, but since the performance did not increase, we stuck with the original one.

Model	**MSE**	**MAE**
RNN	0.015346	0.084491
LSTM	0.003088	0.025929
Stacked LSTM	0.002967	0.024929
Bidirectional ConvLSTM	**0.002586**	**0.021042**
WaveNet	0.005125	0.048105
Sec2Sec + Attention	0.003140	0.027030
ResNet	0.004492	0.038672

Table 1: Models Comparison

5 Contributions

Following are the contribution of each member to the homework. Note that the majority of the testing process was done with constant communication inside the team.

- Fabio: Prepared the data; tested different combinations of input/output window sizes in order to find the most efficient one; applied robust scaling to the data; tested various interpolation techniques to rebuild the unpadded series; tested time2vec representation.
- Elahe: optimized gaussian augmentation, different interpolations and autocorrelation check with ResNet and LSTM.
- Mohammadreza: Various methods have been used for dealing with time series with different lengths, including interpolation, zero-padding, and resampling. LSTM and RNN models were tested.
- Simone: Tested several methods to speed up the training, choosing to group every 60 time series in the end; tested the Seq2Seq and Attention model and the possibility of swapping the time series in each group of 60; tried the approach of creating a model for each class; explored the transfer learning option.

References

[1] URL `https://www.tensorflow.org/api_docs/python/tf/keras/preprocessing/sequence/TimeseriesGenerator`.

[2] M. Dwarampudi and N. V. S. Reddy. Effects of padding on lstms and cnns. *CoRR*, abs/1903.07288, 2019. URL `http://arxiv.org/abs/1903.07288`.

[3] M. Luong, H. Pham, and C. D. Manning. Effective approaches to attention-based neural machine translation. *CoRR*, abs/1508.04025, 2015. URL `http://arxiv.org/abs/1508.04025`.

[4] A. van den Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. W. Senior, and K. Kavukcuoglu. Wavenet: A generative model for raw audio. *CoRR*, abs/1609.03499, 2016. URL `http://arxiv.org/abs/1609.03499`.