Report on second ANN challenge

Alessio Braccini, Giovanni Luca d'Acierno, Davide Li Calsi
January 2022

Introduction

Recurrent Neural Networks (RNN) are used in tasks like forecasting of time series and sequence to sequence learning.
The purpose of this challenge is to create a NN capable of predicting a multivariate time series.

Data Description

Dataset is composed of a multivariate time series with a uniform sampling rate with the following characteristics:

- Length: 68528
- Number of features: 7

We perform some checks in the data in order to exploit all their characteristics, for lack of space we include them in the attached notebook.

Pre Processing

As the first thing, we split the dataset into a training and a validation set. After many tries, we found a split with validation of 3800 samples to be the best choice.
The pool of data was firstly normalized and then, in order to produce a compatible format to feed the model, we used a processing method that transformed raw data in a sequence and turned the time series dataset into a supervised learning problem.
The most relevant parameters of the method were:

- Window: the dimension of the time step;
- Stride: the dimension of the overlap between a window and the following one;
- Telescope: the number of future samples to predict.

We also tried to implement data augmentation by following the methods and techniques proposed in this paper. While the paper suggests several data augmentation techniques, such as magnitude domain transformation, time and frequency transformation, we only tried a small subset of magnitude domain transformation techniques, mostly because of our unfamiliarity with the other types of transformation. However they did not seem to yield significant benefits, probably because we should have adopted more sophisticated techniques, so we eventually gave up on augmentation.

Model selection

The first approaches were to try the dataset on very simple networks composed only of one recurrent layer. One with RNN which gave a result of 4.8 and one with LSTM, which gave many promising results, around 4.5.
In order to predict the values for more than one channel, in each NN, we used a hidden dense layer with telescope times the number of channels (7) neurons followed by a reshape layer to obtain a tensor of dimension: (none, telescope, num_channels).

After many trials with the different networks, we found out that the best values for the window and stride size turned out to be 100 and 10.

Then we added to the previous models supplementary convolutional layers, after some tries we found the models with fewer layers to be more powerful on the given problem, in particular:

- Bidirectional LSTM: it used two convolutional layers with batch normalization and a maxpooling with batch normalization too, followed by a global average pooling, then a Bidirectional LSTM layer with 1024 units divided the fully connected part. it used Adam optimizer (learning rate: 1e-3).
    - Validation mae: 0.0777
    - Validation loss: 0.0115
    - Test accuracy: 3.83

- Bidirectional RNN: it used two convolutional layers coupled with maxpooling with dropout set to 0.5. Then we added a Bidirectional Simple RNN layer with 256 units, another convolutional layer, and a global average pooling followed by a flatten layer, it used Adam optimizer (learning rate: 0.00075)
    - Validation mae: 0.0817
    - Validation loss: 0.0126
    - Test accuracy: 3.67

At the same time, we also tried to make some autoregressive type models. We tried with several configurations but no one brought us any kind of good result.
We tried with some different sizes of telescope, not only 1, but also 4, 8, 16, 32, and 64. The best model that we managed to make was this one:

- Autoregressive: it used four convolutional layers coupled with maxpooling with dropout set to 0.5 and 2 simpleRNN. Then we added a global average pooling layer followed by the fully connected part, it used Adam optimizer (learning rate: 0.00075) and a telescope of 32:
    - Validation mae: 0.0765
    - Validation loss: 0.0124
    - Test accuracy: 5.26

Afterward, we entered the most serious part of the challenge, we focused on the construction of models with the attention mechanism, a fundamental key in the realization of NN that manages time series.

The first tries were not very successful but after we understood how to work with the mechanism and how to exploit it we managed to build up some great models.
We implemented the attention by creating a class containing all the logic of the mechanism (you can see the code for more information), this class will be used in the code of the model, just by declaring it, like a normal layer and it will implement the attention.
Here are some models that we create until the best one.

- First attention model: it used two convolutional layers with batch normalization and maxpooling followed by a global average pooling. Then a Bidirectional LSTM layer

with 512 units is followed by the attention and in the end the fully connected part. It uses Adam optimizer (learning rate: 1e-3).
    - Validation mae: 0.0972
    - Validation loss: 0.0172
    - Test accuracy: 3.89

- Best attention model: it used three convolutional layers coupled with maxpooling. Then there's a Bidirectional GRU (Gated recurrent unit) layer with 512 units followed by the fully connected part, it uses Adam optimizer (learning rate: 0.00075)
    - Validation mae: 0.0789
    - Validation loss: 0.0121
    - Test accuracy: 3.56

However, the final and most effective approach was **divide et impera.** We built a model that was an ensemble of more models: each model predicts one or more quarters of the test set. At first, we tried to construct the model from scratch. We modified the *build_sequences* function so that it could return 3 sequences, one for each quarter in the Development phase. We then designed 3 models (based on previous experience) and trained them on the sequences. This approach was not that effective, but it yielded a model(***1**) that performed extremely well in the 1st quarter. We realized we could recycle it, so we changed our approach. We built a model using the same approach, but with our old models. We examined the output scores of every past model, and selected the three models that performed best in the 1st, 2nd and 3rd quarter. By combining them (as explained before) we were able to build an extremely performing model, leading to a test accuracy of **3.40** in the Development phase. This model was a combination of the model at (***1**) and a Bidirectional RNN model (also mentioned in this paper, the one that returned 3.56 rmse), whose source code is in the python notebook. We also include the model.py code to better illustrate how we recombined the models' outputs.

For the entire Development Phase, we trained our networks with adam optimizer that proved to be the most efficient and effective.

Conclusions

In conclusion, after all the testing, we chose to submit the model that reached the best result: the divide et impera model. For further details, you can have a look at the code in the notebook file.
Surprisingly we found out that the RNN models performed better than the LSTM ones, this could be due to the periodicity of the given data and RNN could predict in a correct way without going back for more than 8/10 timesteps.
After all, attention gave us the best result in a single model approach, this is as we saw in class and as we expected.
Finally, we moved to a divide et impera approach. If a single model might have difficulties while predicting that many values, two models can share the burden and specialize on a specific part of the sequence. In our approach, we saw that if one model specializes on the first quarter, and another model takes care of the remaining ones, performance drastically increases.