University of Bern

CAS Applied Data Science 2018-2019, Module 5

Peer consulting report

Andrey Martynov, *andrey.martynov@giub.unibe.ch*

Bern, the 20th of June, 2019

The peer consulting has taken place in July 2019 in Bern, mutually with the CAS participant Tomasz Luniak. The consulting has been focused on a neural network application, being developed by Tomasz and intended for predicting the air pollution level based on preceding measurements.
The model has a Long Short-Term Memory (LSTM) architecture and uses the Keras Python package. It is build on the basis of the https://github.com/jaungiers/LSTM-Neural-Network-for-Time-Series-Prediction project, which is described here in details: https://www.altumintelligence.com/articles/a/Time-Series-Prediction-Using-LSTM-Deep-Neural-Networks

In its current form the model uses for training the air pollutant concentration measurements, taken from publicly available data sources in a form of a one-dimensional sequence of measured values and produces a single forecast value for the next step of the mearuring sequence (the Point-by-Point prediction mode). The model comprises several Jupyter notebooks, Python scripts, a configuration file, and is accompanied by a training dataset. A copy of the model code with a test dataset has been given to me by Tomasz Luniak and has been successfully installed and executed on the climcal4.giub.unibe.ch server (Debian GNU/Linux 9.9).

Among the good things about the model I can mention the following:

1. The model architecture and underlying basis project are in general adequate to the task. The LSTM neural architecture can be efficiently used for forecasting the short-term behavior of not fully stochastic time series by reproducing the patterns learned from training datasets. The pollutant concentrations in the atmospheric air are partially stochastic, but to a considerable extent are determined by natural climate cycles – in particular, diurnal and annual - which can be trained and reproduced by a neural network. In addition, non-periodic short-term patterns in the latest available observations can also be caught up by the model.

2. The model uses a regularization method for avoiding overfitting. This approach is realized in the model by implementing the Dropout technique (https://machinelearningmastery.com/use-dropout-lstm-networks-time-series-forecasting/). That is, neurons (nodes) are randomly dropped out from

the neural structure during training, which prevents the model from over-adapting to the training data by fixing up the mistakes of the previous layers – which might lead to overfitting.
By implementing the Dropout technique, the possibility of overfitting the model is largely reduced.

3. The model configuration is determined by a single configuration file, which allows to easily modify most of parameters of the model and its implementation (the configuration of the neural model itself, Dropout settings, data and training parameters).

However, to my understanding the model in its current state requires some additional work. Here are my main suggestions.

1. As stated before, the current model is only able to produce Point-by-Point predictions. The current code is not performing the verification of the predicted values. Thus, the efficiency of using the model can not at the moment be estimated. It also makes impossible optimization of the model settings and searching for optimal training dataset properties (frequency, length, etc).
My main proposal would be to add to the code a model validation mechanism, using the so-called hind-casting approach, that is, using past observation series for “predicting” next observed values. It can be done by selecting a training frame of fixed length in the end of the original training dataset, using only this frame for “forecasting” the next value, and then shifting the training frame to the next timestep. By recording the predicted values the model will produce a series of forecasts (of the length of the entire dataset - the length of the training frame), which can then be compared to matching observations. A qualitative measure of matching between these timeseries can be used as a qualitative measure of the model efficiency.

2. I would also suggest to the author to introduce additional mechanisms, preventing overfitting in training with multiple epochs. The current code saves the model state only once a preset number of epochs has been performed. One possible option would be saving the model state after each epoch and compare the model performance with that of the preceding epoch. If it starts to deteriorate, the previous epoch should be kept as the optimal model state. Another option might be to save the model state at every epoch, until the training is finished, and then select the best one.

3. Additionally, I would suggest to the author of the model to add more comments to the parts of the model code, modified and/or written by him and to use the version control for better following the modifications of the system.