Artificial Neural Networks and Deep Learning 2023 - Politecnico di Milano
3 Pentium 4 in a Trenchcoat
10667728 Lorenzo Morelli, 10676573 Davide Edoardo Pellegrino, 10703747 Pier Luigi Porri

Time Series Forecasting with LSTM Networks

The goal of the second challenge is to design and implement a forecasting model to learn how to exploit past observations in the input sequences and correctly predict the future by predicting several uncorrelated time series. The prerequisite is that the model exhibits generalization capabilities in the forecasting domain, allowing it to transcend the constraints of specific time domains.

1. How we approached the task

As an initial exploration of the challenge, our team conducted a thorough examination of the provided data, this time with a focus on the time series' lengths and overall distribution. It was immediately apparent that most series' *valid_periods* occupied the last half of the timeline, with many having valid points in the last 25%, and few rows with significant values in the first 50 or so timestamps. This padding problem will reveal itself as a major challenge for the team later.

The already applied normalization saved us a step in the preprocessing, but it was immediately obvious that a lot of work was needed to effectively integrate the information about the categories in our input to the model, as well as selecting the right amount of data points for each series. With these aspects in mind, we began looking for an effective solution for the sequence building process.

During the dataset splitting phase, we initially agreed on splitting the dataset on the series' lengths, devoting the last 20% for testing and validation. However, since the rows are all independent from each other, there was really no need to split the dataset that way, rendering more difficult the process of building the actual sequences for the model. Thus, we later switched to splitting vertically the dataset, meaning that the last 20% of the series' themselves were reserved for testing and validation. After some speculation, we found this new solution more appropriate.

2. The Categories Conundrum

2.1 First Contact

The *Categories* array started out as a bit of a mystery for us. Initially, we didn't quite understand the use of this information, nor how to properly integrate it in the time series dataset. Thus, we decided to just ignore it, for the time being, and train the model on just the data points in the *training_data* array.

We quickly turned back, however.

2.2 First Attempt at Integrating

It just didn't make sense to us that such useful information could be left aside, so we started working on a way to feed it into our model alongside the rest of the dataset.

Our first attempt was a simple and straightforward one: add a third dimension to our two-dimensional array, now (48000, 2776) where the first dimension is the number of rows and the second the length of the series. A third one would represent the number of features: considering that the sequences are univariate, the total would come up to 2, including the *Categories* feature. After applying one-hot encoding to the 6 possible categories, though, the total number goes up to 7, resulting in a shape of (48000, 2776, 7).

The dataset in this format couldn't be fed to the model, though. In order to properly train the model on forecasting, we had to split the series in sequences, shift them, and only then start the training process. This need is what made us realize that the way that we currently handled the categories was not sustainable: every time we went through the *build_sequences* function that we created, we would run out of RAM on Kaggle.

The problem lay in the size of our dataset: by integrating the categories the way that we did, we effectively multiplied it by 6 times, and that meant 6 times the sequences. That was obviously inefficient.

2.3 The Problem that Solved Itself

Until we found a solution, we convened on shelving the categories once more. Although, we later realized that the problem became nonexistent once we switched to a different sequence building method that used way less of the original dataset, saving on size and allowing the categories back in by appending them to the sequences instead of appending them to the series. More on this later.

3. The Padding Problem

3.1 First Speculation

Initially, the presence of the padding didn't bother us much. We speculated that our model would be intelligent enough to ignore the long sequence of zero values that preceded the actual ones. Testing our theory in practice, though, revealed that too many of our sequences were just zeroes back to back.

3.2 The Padding Removal

Removing the padding from the series was overdue, but doing so presented another problem: the latter were now of different lengths. That meant that we couldn't feed them in the model as is, our sequences had to be padded somehow to be generated with the same length from a set window. Thus, we implemented a check in our *build_sequences* function, to assert that all sets of data points that were shorter than the window would be padded accordingly.

3.3 Masking the Padding

Meanwhile, we agreed on giving a shot to masking, by inserting the corresponding layer after the input one in the model. This meant changing all the values of the padding (wherever padding exists) to a specific value that our network would ignore. We chose -1 and tested the model. The results seemed promising, so we settled on this method, both on the dataset and in our *build_sequences* function.

4. The Sequences Speculation

4.1 Big Sequences

Building the sequences correctly was our priority since the beginning. Initially, as mentioned in **2.1**, we just went on with the standard window-stride-telescope method seen at lecture, ignoring the *categories* array completely, resulting in 3-dimensional arrays for both training and testing, where the first dimension was the number of sequences, the second was the window size, and the third was the features (1 for training, 9 for testing, since we had to predict 9 data points). As we decided to integrate the categories, this method proved unsustainable and caused us to fill up the RAM too quickly. We had to adapt.

4.2 Not-So-Big Sequences

To mitigate this problem, we speculated that the length dimension was completely unnecessary: just like the dataset as explained in **2.2**, the sequences themselves could be seen as a feature, thus transferring that information in the (formerly) third dimension of the dataset and dropping the second. With this implementation and a series of well-timed *del* commands, the memory problem was solved and we could integrate the categories once again and ended up with 3-dimension arrays for training and 2-dimension arrays for testing, where the categories didn't matter.

5. The Model

Unlike the last one, this challenge had us try and test many pre-processing strategies instead of models.
We started from a simple LSTM network, we worked our way up to some more complex ones with concatenated LSTM layers, we tried adding additional dense and Convolutional layers, we even had a layer of Self Attention at one point. In the end, the best performing model was the one to the right. The GAP and Dense layers towards the end were particularly useful to ensure our model would apply the right shape to the outputs.

```
Model: "LSTM"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_layer (InputLayer)    [(None, 100, 7)]          0

 masking_3 (Masking)         (None, 100, 7)            0

 lstm_5 (LSTM)               (None, 100, 160)          107520

 dropout_6 (Dropout)         (None, 100, 160)          0

 lstm_6 (LSTM)               (None, 100, 80)           77120

 dropout_7 (Dropout)         (None, 100, 80)           0

 global_average_pooling1d_3  (None, 80)                0
  (GlobalAveragePooling1D)

 batch_normalization_3 (Bat  (None, 80)                320
 chNormalization)

 dense_3 (Dense)             (None, 9)                 729

=================================================================
Total params: 185689 (725.35 KB)
Trainable params: 185529 (724.72 KB)
Non-trainable params: 160 (640.00 Byte)
```

References

- Keras, LSTM layer: https://keras.io/api/layers/recurrent_layers/lstm/
- Keras, GRU layer: https://keras.io/api/layers/recurrent_layers/gru/
- Keras, GAP1D layer: https://keras.io/api/layers/pooling_layers/global_average_pooling1d/

Contributions

- Davide Pellegrino: categories integration in the dataset, model selection, even more traffic management in Cities Skylines II during training wait times, and report work
- Lorenzo Morelli: most of the coding, model selection, data preprocessing and notebook embellishment
- Pier Luigi Porri: autoregression work, model selection and strategy research

3 Pentium 4 in a Trenchcoat are Lorenzo Morelli, Davide Edoardo Pellegrino and Pier Luigi Porri.
See attached notebook for more information.