AN2DL Homework 2 - Report

Gabriele Bozzetto - Lorenzo Castiglia - Davide Mantegazza

Leaderboard: bozzetto-castiglia-mantegazza

Introduction

We approached the task by building different types of neural networks models following the outlines provided during lectures. In this short document we are going to highlight and present the most notable models and experiments we made: starting from our first and raw model, describing our thought process to justify the changes and tweaks done in order to enhance it.

Models

First experiments

Similarly to the first ANNDL competition, we decided to start with a simple model without any particular optimization, in order to assess the situation. Initially, we prepared the data by dividing them in sequences of 200 samples (also referred in this document as windows), each of them taken with a stride of 10, and by defining as a target the subsequent 864 samples (we did not employ autoregression). Then the model was created with a single LSTM layer with a memory of 128 neurons, followed by a single fully connected layer made by 6048 neurons, followed by a reshape layer used to have a 864*7 tensor as output, who's finally fed into a Conv1D layer. The model was then trained without any counter-overfitting measure and it yielded a RMSE of 4,6. After this, we tried to improve the result by adding other features to the model, and we came to the conclusion that the addition of "complex" layers like Bidirectional LSTM, Dropout layers, convolutional layers, and dense layers with an increased number of neurons, only seemed to increase the error on the test set. Therefore, we decided to focus on the original model structure, and try to improve its performance by varying the number of LSTM, dense layers and neurons in, them, by employing overfitting measures like Early Stopping and learning rate reduction on plateau and by using LeakyReLU as the activation function for the dense layers, in order to tackle an eventual vanishing gradient issue. After some tests we found out that the best results were obtained when the LSTM memory size was similar to the one of the sample window, and in particular we achieved the best outcome with 2 LSTM layers of size 256 and 3 dense layers with 6048 neurons each, that resulted in a RMSE of 3,895. Together with the very first one, this model had also been trained at a later time without the final Conv1D layer; this modification led to an improvement of the root mean square error on these two models (respectively 4,42 for the first model and 3,72 for the other one).

Convolutional model

While experimenting with different types of models, we observed worse performance when using more complex models. For instance swapping the LSTM layers with bidirectional LSTM or GRU led to higher RMSE. Adding dense layers had the same effect, probably due to overfitting. Therefore, we explored the opposite direction, reducing the model complexity and parameters by using convolutional layers instead of LSTM. We tested a variety of configurations, the best of which is composed of 4 Conv1D layers with an increasing number of filters alternating with MaxPooling and 2 Dense layers following. The result obtained was just under 4 RMSE, a decent score at the time but not close to our best overall score, so we decided to continue optimizing the LSTM model using KerasTuner.

KerasTuner

During our experiments, we decided to set up a grid search to find which hyperparameters were best for our model, rather than proceeding by trial and error. In particular, we extensively employed a library called KerasTuner. We attach among the uploaded notebooks an example of how we used this tool. Our search for hyperparameters focused on: the number of Dense layers, num LSTM and the alpha value for the "leaky_relu" activation functions at the end of each dense layer. Starting therefore from our best model, and using the tuner several times and in several iterations we obtained these hyperparameters:

- numLSTM: 3 layers (plus one without return of sequence)
- numDenseLayers: 3 layers
- alpha0: 0.09340838701412477
- alpha1: 0.17930289675746583
- alpha2: 0.2532112296895768

After creating the new model, having fixed the values of the hyperparameters, we also slightly changed the training strategy. Up to this point, in fact, to avoid overfitting the dataset, we had always performed a train-validation split in order to effectively set up an early stopping. Since the split used for validation was 10% of the dataset itself, after finishing the first training we decided to bring together all the data and we retrained the model for a single epoch, this time on the entire dataset. Overall this strategy brought our score to 3.42 (our best up to date).

After these attempts we made other tests, always using the KerasTuner, changing the percentages of the split training/validation and wrote a notebook for a custom k-fold cross-validation, but without obtaining significant improvements.

Attention mechanism

Since we exhaustively explored the possibilities of improvement of our best model, which occupied the top of the leaderboard for several days, we decided to move on and try to implement the attention mechanism.

In order to do so, we replaced the recurrent layers with encoder blocks of a Transformer model, similar to those described by the research paper "Attention is all you need". To apply

these concepts to time series classification, we followed the guidelines contained in the official keras documentation. [1]
We performed a considerable amount of hyperparameter tuning using KerasTuner, including for instance the number of encoder blocks, number of heads and size for the MultiHeadAttention layers, number of dense layers, activation functions and alpha parameter for the leakyrelu and dropout rates after almost every layer. A notebook named *'transformer-kerastuner.ipynb'* is present in the folder, showing the search of some of the first hyperparameters we considered, but as mentioned previously we experimented with many others. The best RMSE obtained by our Transformer model is around 3.6, unfortunately not outperforming our best LSTM model which scored 3.4. This result was obtained with 2 encoder blocks, 4 attention heads of size 128, 3 mlp units with leakyrelu and some dropout layers.

[1] Theodoros Ntakouris - "Timeseries classification with a Transformer model", https://keras.io/examples/timeseries/timeseries_transformer_classification/

Notebooks

Together with this report, we are also including the most significant notebooks that we used for the duration of the competition:

- SimpleForecasting: this was the first tested model
- Convolutional: convolutional layers in place of LSTM
- 2LSTM-3DL-Mem256: this notebook returned the best model before starting to experiment with KerasTuner
- LevelAlphaSearch1024: this model returned the best result by using only KerasTuner
- 4-LSTM-128-fixed-alpha-Retraining-on-all-data: best model overall
- transformer-kerastuner: attention mechanism implementation

Inside the notebooks there are plots of the models' structures and the validation loss on the training set and the validation set over the epochs.

Notes

- Notebooks have been executed via Kaggle, therefore it might be necessary to modify the cells in order to correctly load the dataset
- It is important to note that we set the seed to make the experiments reproducible but, since the evaluation with a GPU introduces a stochastic element, the experiments may not be repeatable with perfect accuracy
- During the first phase, we performed a custom cross validation process locally, creating several train-test splits of the dataset, which allowed us to identify the best fold to submit