DEEPressi - Report

Gianluca Ruberto, Riccardo Pazzi, Tommaso Brumani

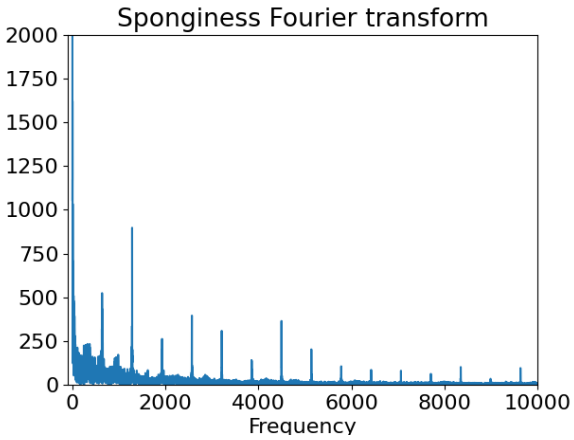
PREPROCESSING

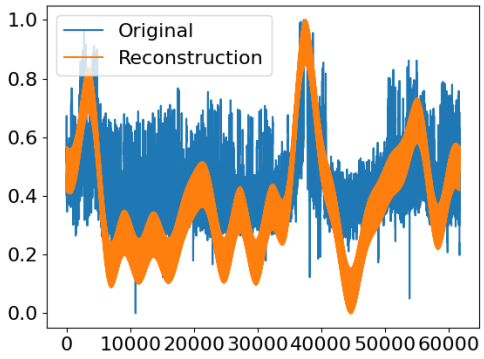
Normalization was applied to each signal to better learn relationships between them, independently from the relative scale of their values.
In the final versions of the model, once it had been thoroughly tested, we decided to continue using just the training and validation sets in order to benefit from a larger training dataset, which slightly improved the performance at test time (0.03).

DATA AUGMENTATION

Frequency:

We attempted to extract information in the frequency domain from the signal in order to identify the dominant frequencies: this was done using Fast Fourier Transform.





The idea was to reconstruct the signal using only the 10 most relevant frequencies (obtained result can be seen in orange), then use the reconstructed signals as side information for the network to better learn the main frequencies, this seemed promising since many frequencies where dominant in multiple signals (~650Hz and ~1200Hz).
However using the reconstructed signals to train the network didn't provide the expected results and ultimately we abandoned this approach.

Gaussian Noise:

Another form of data augmentation we tried was random gaussian noise, this was especially tedious to implement since the Keras gaussian layer was not compatible with the Keras version on CodaLab. In the end we found a workaround, and the results marginally improved with noise factors around 1-10%.

NETWORK STRUCTURES

Convolutional LSTMs

We first tinkered around the network structure seen during exercise sessions, both with bidirectional and unidirectional LSTMs, but we kept encountering the same problem whenever we plotted the results: the network seemed to learn the average signal without following the curves.The solution we found was moving the convolutional layer before the LSTM to extract features directly from the signal and then interpret the features using the LSTM.
Our best performing model adopted this approach and was composed by the following layers:

Conv1d → max pooling → biLSTM → dropout → dense → dropout

Single LSTM

We then tried a network made by a single, non-bidirectional LSTM with 864 neurons, followed by a convolutional layer with no activation function and a dense layer, which scored 5.69 on the hidden set. It was a direct forecast with an input window of 864. The score achieved was not so bad, but it wasn't good enough to convince us to proceed with this strategy.

Attention/Transformer

We tried to use an architecture similar to the Transformer. The encoder section was made of 2 blocks of multi-head attention followed by a convolutional section. The decoder section was symmetric but it also had a multi-head attention based on the encoder part. For multi-head attention we employed 4 heads with a size of 256. It was a direct forecasting model and It achieved 6.02 on the hidden set.

Because of the lack of masking we attempted to use a sort of "sequential autoencoder" with one block of 3 encoders and 6 blocks of decoders. Furthermore, we substituted the convolutional section with a dense one. Because of the exploding number of parameters (13M), we employed 7 heads with size 7 for multi head attention. It was an hybrid autoregressive model that predicted 108 time steps. It achieved 5.06 in the hidden set, which was very close to our best score at that time. We however decided to abandon this approach, as the model's number of parameters was already extremely high and further modifications were not showing discernible results in the local tests.

Diluted Convolution

We attempted a very simple model based on Diluted Convolutions of the WaveNet architecture. Despite having just 980 parameters, it was a model with 7 convolutional layers with increasing dilution rate that took an input of 864 timesteps and produced an output of the same size; it achieved good results in local testing. We could not submit it because of problems with CodaLab. We kept modifying the network, but the result on the local test did not improve over our best network, so we didn't proceed in this direction.

Bidirectional LSTM and Residuals: model H_MK_XXXIIId

We tried to use a network made of 7 blocks. Each block had a dropout layer, a bidirectional LSTM of 128, a layerNormalization that took as input the sum of the output of the bidirectional LSTM and the output of the dropout (residual connection), a convolutional layer of 256 and a maxpooling layer. It took as input 2048 timesteps and produced as output 16 timesteps. It achieved a score of 4.42 on the hidden set and it was our best score at that time. It was a complex model with 4M parameters and despite our efforts to improve it, we had to change direction.

Single Dense layer: model D_MK_X

During the final days of the challenge we tried a direct forecasting model with a single Dense layer followed by a PRelu activation function. It took as input 864 timesteps and produced as output the same number. It has a very large number parameters (36M), although the size of the model depends purely on the size of the output sequence: reducing the output sequence to 16 resulted in a far more contained size while worsening performance by very little. Our theory on why it works so well is that the provided time series are relatively simple and without exceedingly long time dependencies, therefore they can be learned successfully with a simple dense network.

HYPERPARAMETER TUNING

Parameter Tuning

First we tried a manual approach, then moving to Keras Tuner's bayesian search, hoping for better results. In the end however, the approach which worked best was using random values taken from a specified interval.

Clipnorm -> NaN

We noticed that when training a network with multiple LSTMs and stride = 1 we seemed to have the issue of exploding gradient. In fact, sometimes the loss function even assumed NaN value. To solve this

problem we applied clipnorm to the optimizer (Adam). For our implementation, the clipnorm value that appeared to solve the problem was between 0.01 and 0.001.

Window

One parameter which was really significant in score results was the window size: we wanted to keep a window size big enough to include signal oscillations and accurately predict at test time, so we started with a size 2000, but later on moved to 800-900 due to better results.

REGULARIZATION

Learning Rate decrease on plateau was used with and without Adam, in order to obtain more refined solutions and find more precise local minima.

Early Stopping was used throughout the whole process as the main protection from overfitting, and we also used it to understand which models would immediately stop (and therefore immediately overfit) to gauge which degree of structural complexity worked best with the task.

Despite what we have learnt in class, we noticed that sometimes adding a **dropout** layer resulted in a flat line as prediction, so we employed them only when they provided discernible improvements.

OUTPUT

Autoregression / Direct Forecasting

At first we tried an **autoregressive** model, but it resulted in a very long processing time in codalab.

At the same time however, models with **direct forecasting** often used too much memory and resulted in crashes on Colab and Kaggle (we also used this approach for our final model). We found our sweet spot in an “hybrid autoregression” model: the majority of the models that we trained, produced as output 16 timesteps at a time.

SALIENT MOMENTS IN MODEL PROGRESSION

Model Name	Description	Score
D_MK_I	Direct forecasting as seen in class	11.68
A_MK_I	Autoregression forecasting as seen in class	10.02
H_MK_XVI	Model with convolutional lstm, hybrid autoregression, data normalization and stride = 10	5.68
H_MK_XIX	H_MK_XVI, with clipnorm and hyperparameter tuning	4.46
H_MK_XXXIIId	Model with residuals	4.42
H_MK_XXXVc	H_MK_XIX with window halved to 1000 timesteps	4.20
More hyperparameter tuning		4.11
Removed the test set		4.08
D_MK_X	Single dense layer	3.71
D_MK_Xb	Increased epochs and patience	3.64
D_MK_Xc	Decreased stride to 1	3.62