Università degli studi di
Milano-Bicocca

Advanced Machine Learning

Final Project

# Electical Motor Temperature

*Authors:*
Federico Moiraghi -799735- f.moiraghimotta@campus.unimib.it
Pranav Kasela - 846965 - p.kasela@campus.unimib.it
Roberto Berlucchi - 847939 - r.berlucchi@campus.unimib.it

2019-2020

**Abstract**

The present Report is a summary of methodologies used to predict the temperature of various part of a prototype electrical motor given some tests on a bench. The resulting model have to yield acceptable predictions and to be light enough to be used by the car itself during its daily use: autos can start cooling components before the temperature grows critically (first task) or can estimate temperature without a specific sensor (second task), due to its cost and weakness, knowing only basic environmental information.

# 1 Introduction

The data set comprises several sensor data collected from a permanent magnet synchronous motor (PMSM) deployed on a test bench. The PMSM represents a German OEM's prototype model. Test bench measurements were collected by the LEA department at Paderborn University.

The recordings are sampled at a frequency of $2Hz$ and is divided in various profiles and has a total of 998070 observations. Each profile indicates a different session and each session can have different length varying from one to six hours.
The input variables are:

- **Ambient temperature** as measured by a thermal sensor located close to the stator;
- **Coolant temperature**, the motor is water cooled and the measurement is taken at outflow;
- The current and voltage are transformed through a $dq0$ transformation in a d-q coordinate system, it basically converts a three phase balanced reference system (in an AC system) into 2 coordinates, denoted by d and q, via a rotating reference frame with angle $\theta$. The currents are denoted by **i_d** and **i_q** and the voltages are denoted by **u_d** and **u_q**;
- **Motor speed**.

The target variables are:

- **pm**: Permanent Magnet surface temperature representing the rotor temperature, measured with an infrared thermography unit.
- **stator_yoke**, **stator_tooth**, **stator_winding** temperature measured with a thermal sensor of the corresponding components.

In some of the variables, gaussian noise is introduced to simulate real world driving cycles. Being sensors data, missing values are replaced by the provider with the previous one, causing some flat areas when sensors fall offline for a long period.
The main objective is to create a lightweight model to predict the `pm` and `stator` variables minimizing the MSE (because the model needs to be deployed with best cost-precision ratio); a secondary objective is to predict more accurately higher temperature than the lower temperature using a modified loss.

# 2 Datasets

The data set can be found on Kaggle[1]. From the data set the `torque` feature is immediately excluded, as it is deemed unreliable from the data set provider itself.
The data set is divided into train, validation and test set: validation data consists of `profile_id`s $20, 31, 46, 54, 62, 70, 79, 72$; the test set profiles are 35 and 42 and the training set consists of all the other profiles. Their relative distributions are plotted in the Figure 1.
In the Figure 2 the correlation between the variables is shown. The target variables are highly correlated among themselves, in particular the `stator` variables.
Data was already standardized by the provider (for some anonymization issues), but variables do not have a normal distribution thus a further normalization between 0 and 1 (calculated only on the training set) is applied.
For each profile the data set is modified: to each row the lagged features are concatenated, the number of lags to keep can be changed, after this procedure the rows of all the profiles are united and shuffled through a custom data loader.

# 3 The Methodological Approach

Different *Deep Learning* architectures are tried in order to compare them and choose the most suitable model to the problem: each model is built

[1] https://www.kaggle.com/wkirgsn/electric-motor-temperature

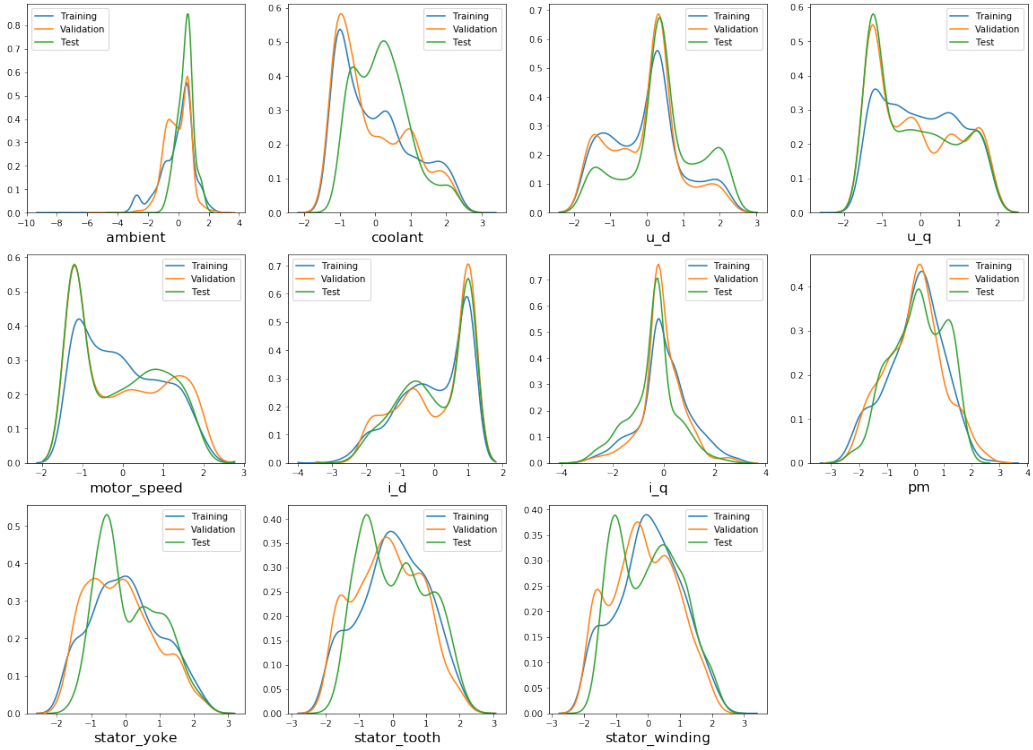


Figure 1: Distribution of the variables grouped by the division

using `pytorch` and optimized with an Auto-ML algorithm using `sherpa` optimization library. In particular the *Gaussian Process* (GP) surrogate model is used with the *Expected Improvement* (EI) acquisition function to have a better exploration during the optimization.

Each model is optimized using Adam optimizer and a standard *Mean Squared Error* (MSE) as the loss function in a first trial and then re-optimized using a custom weighted-version that assigns a triple importance to temperatures that surpass the value of the median (computed on the train set).

## 3.1 Dataloaders

To avoid giving all input and output matrix to the model (which has a side-effect: it fills both RAM and GPU memory and crashes everything), two ad-hoc dataloaders are programmed to optimize the learning process.

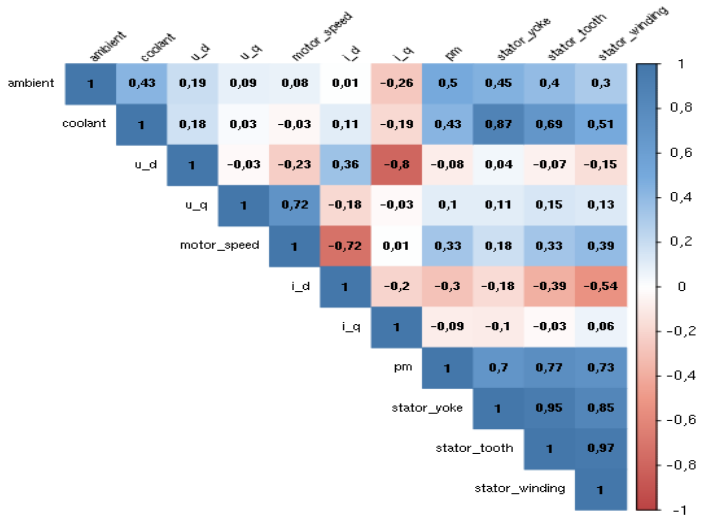


Figure 2: Correlation Plot of the considered variables

Belonging to various independent series, data is represented internally as a sequence of tuples $< \mathrm{id}_{serie}, \mathrm{id}_{observation} >$ so that can be easily shuffled and yielded to the model (without side-effects) using an arbitrary large batch size.

**First Task** To accomplish the first task, data is given to the algorithm so that input matrix contains all values of all features and target variables throw the temporal window (starting from an arbitrary moment in the past $t_{-max}$ (the maximum lag) to the previous observation $t_{-1}$), while target vector contains following values of only target variables at time $t_0$.

**Second Task** For the second task instead, dataloaders yield a input matrix containing only features values for a window time starting in a moment $t_{max}$ to $t_0$; in addition a vector of values of target variables at time $t_0$ is given as ground truth, so it's a real time prediction of the target and in this case there is a unique model which predicts all four targets.

## 3.2 Models

All models are implemented as Sequence to Value (Seq2Val) so that the training process can be paralleled and shuffled, with improvements in both speed and quality. In addition, a Sequence to Value model can be easily rewritten as Sequence to Sequence (Seq2Seq), providing real-time information to the driver.

**RNN** In order to accomplish both tasks with a lightweight model easily usable by a car, a simple Recurrent Neural Network is implemented: this model is provided with one hidden layer that depends on both current data and previous observations, and uses it to estimate the target variables. After the hidden layer, data flows through two independent feed-forward neural networks (with two layers both) to estimate values for `pm` and `stator` variables: this approach is justified by the higher correlation between `stator` temperatures and lower correlation with `pm`.

Which layer (the previous input, the estimated output or the embedding) the RNN yields to the next step and how to use this information (where it is linked) are parameters settable in the model construction and optimized through Auto-ML; in addition, *learning rate*, number of neurons per hidden layer, and length of the input sequence are optimized in the same way: to do so, the `Custom_RNN.forward()` method is implemented recursively to make it as adaptable as possible.

**LSTM and GRU** Two more type of recursive neural architecture are tested, the first one uses a LSTM layer while the other uses a GRU layer instead. For the first task two independent models are created: one to predict only the `pm` variable and one to predict the 3 `stator` variables and the window is fixed to 60 lags, it does not need to be optimized since the LSTM and GRU should automatically understand how many lags are important.

The model is pretty simple (as required by the task): it has one recursive layer (either LSTM or GRU) followed by two fully connected layer, the second fully connected layer can be omitted if the number of neurons are 0, lastly there is the output layer which has one neuron while predicting `pm` and three neurons while predicting `stator`. During the second task all the four targets are predicted together.

The hyperparameter optimizer needs to optimize the number of hidden units, the number of neurons for each fully connected layer, the learning rate

and the batch size of the data. The objective score is the MSE and it is calculated, after one epoch of the training set, on the validation set.

**CNN** A CNN model is also tested, even though the number of parameters increases compared to the other approach described before.

For the first task, the model has 2 Convolutional layer each one followed by a Max Pooling layer, the last convolution-pooling block is flattenized and followed by 2 Fully Connected layer and the output layer.
The second Convolutional and Fully Connected layers can be omitted if the number of filters or neurons are 0.
For the second task the architecture is simplified: the poolings are removed.

The convolution happens on the vectors that contain all the variables for each specific time in $[t_{current-lag}, t_{current-1}]$, where $t_{current}$ is the time at which the target needs to be predicted and lag is the temporal window that the model can see. Basically this model is a lighter version of a FC model (it has less number of weights) and it extracts the most relevant features to pass to the FC layers. The temporal window is the same used by the LSTM model.

In this architecture the hyperparameters to optimize are the stride and kernel of the convolutions, the number of neurons for each FC layer, the learning rate and the batch size.

# 4 Results and Evaluation

## 4.1 First Task

In the Figure 3 the result of hyper-parameter optimization are shown, for both models one predicting the `pm` variable and the other predicting the three `stator` variables, the RNN is excluded because of it's poor performance when compared to the other three models.

The results of the model on Training, Validation and Test are reported on the Table 1, in this case all the output variables are concatenated together and a general loss is calculated.

In addition, to improve predictions of critical temperatures a custom weighted version of MSE is used as described in the previous section. Results of hyperparameter optimization with the new loss are shown in the Figure 4.

| | MSE | | | Weighted MSE | | |
|---|---|---|---|---|---|---|
| Models | Train | Validation | Test | Train | Validation | Test |
| RNN | | | | | | |
| CNN | 2.32e-5 | 2.28e-5 | 2.39e-5 | | | |
| GRU | 1.29e-5 | 1.24e-5 | 1.28e-5 | | | |
| LSTM | 0.98e-5 | 0.96e-5 | 1.0e-5 | | | |

Table 1: Results on Training, Validation and Test Set for the first task.

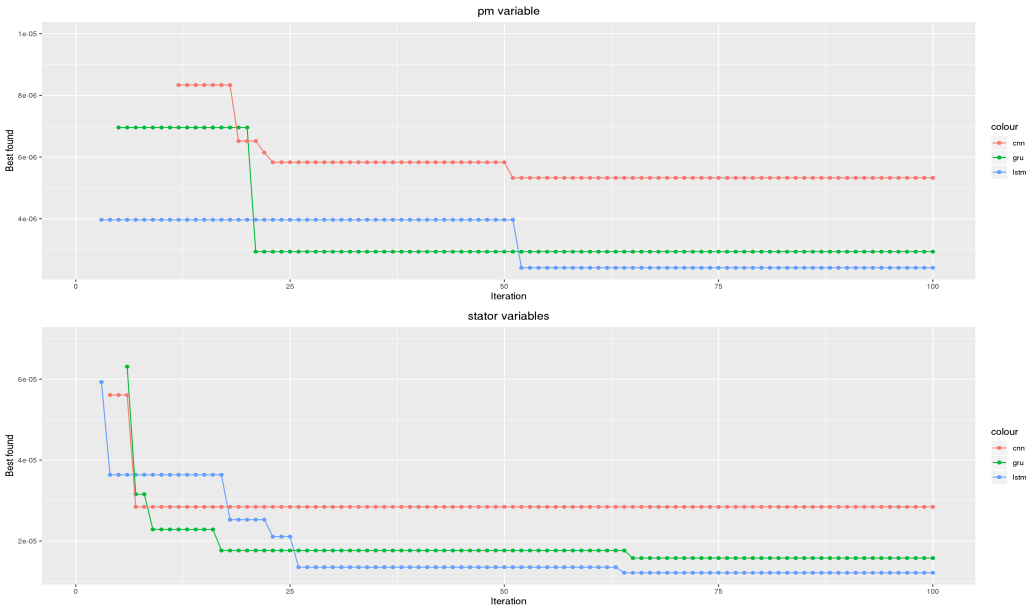


Figure 3: Results of the optimization process on the pm and the three stator variables respectively using the MSE for the first task (RNN is excluded due to its poor performance)

## 4.2 Second Task

For the secondary task (predicting motor temperature without knowing previous values) results of hyperparameters optimization are shown in the figure 5:

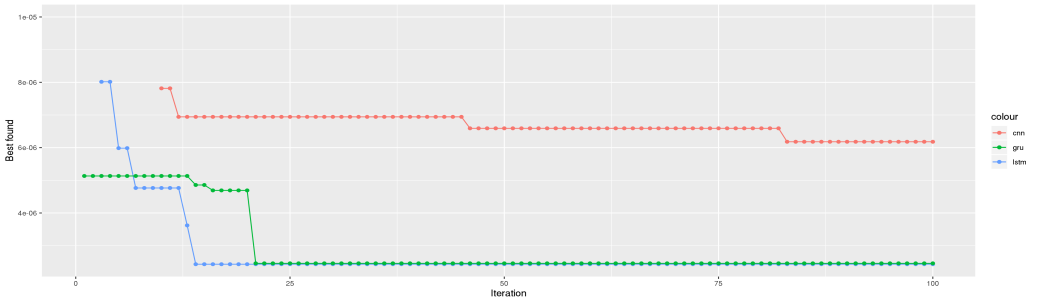


Figure 4: Results of the optimization process using a custom loss for the first task (RNN is excluded due to its poor performance)

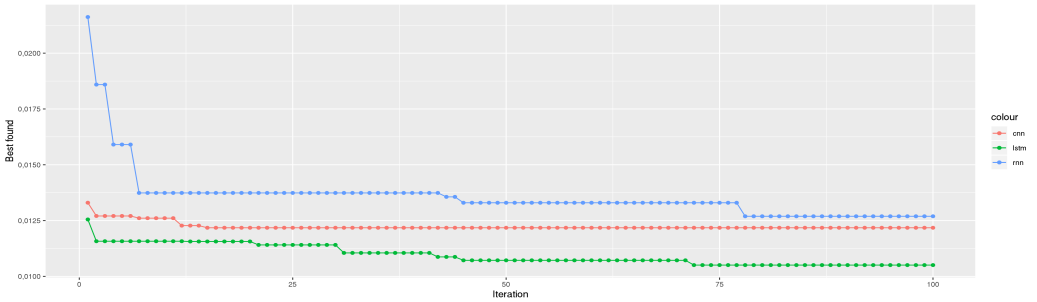


Figure 5: Result of the hyper-parameter optimization for the second task

# 5 Discussion

As shown in Figure 3 and Figure 4, after the optimization process, the LSTM and GRU yield the best results (LSTM is slightly better), and CNN provides results comparable with these architecture. RNN however yields poor previsions when compared to the other architectures.

However, comparing results using only the loss value is not satisfying for the following task: the model have to be light enough to be easily used by a car. So both number of parameters and performance are considered and plotted in the Figure **??**.

In the second task the performance are pretty good for the `stator` variables but not so good for the `pm` variable. The best model in this task is again the LSTM. The RNN's behavior is improved too. In the prediction

of the LSTM model on the test set (Figure 6) a lot of noise can be noted in the prediction of the `pm` variable. The `stator` variable prediction seems to follow the local trend pretty well, especially, in a case where there is no direct formula connecting different physical quantities.

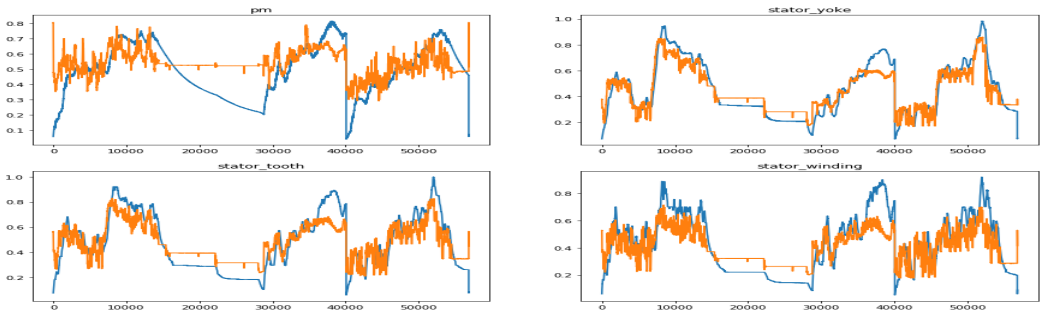


Figure 6: Prediction of the LSTM model in the second task

The weighted loss has been applied also in this case (with the CNN model), it improves, as expected, the prediction for higher temperature as shown in Figure 7, the prediction for lower and mid range temperature is more noisy, thus the overall MSE decreases.

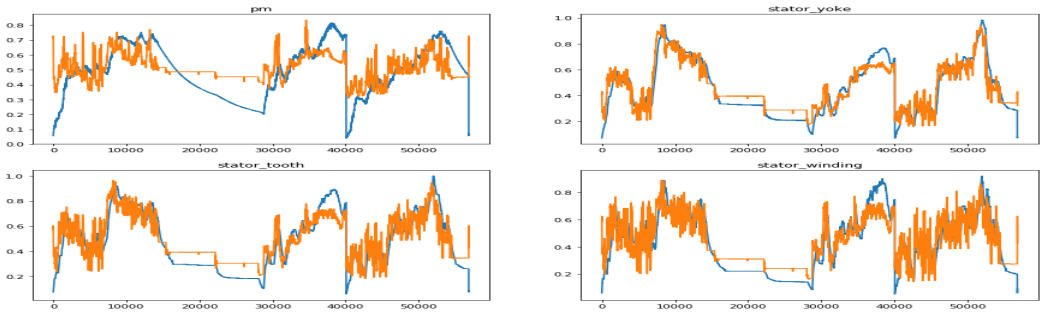


Figure 7: Prediction of the CNN model in the second task with the custom loss.

Even creating a separate independent model for the `pm` variable doesn't improve it's performance.

# 6 Conclusions

The recursive models, since they are used for this king of data/objective, are expected to work well, but reading enough literature [1] [2], it can be deduced that the CNN should also work well, this was a confirmation of this fact: there performance are at least comparable to the recursive models. This report cannot be a proof of them working better or worse because NAS was not done for either model, only the hyper parameter were optimized with a simple pseudo NAS, where the number of layers can vary in some cases.

The target variables have a normal distribution when collected from different independent profiles, but in each profile the distribution is not normal, thus the optimal predictor for a series might not be linear and the arima ACF/PACF plot has different ARIMA coefficients and components so for each series a new ARIMA needs to be trained making it unpractical to use, but the deep learning model generalizes well on the test set, if the model needs to be simplified even more, a great option could be an exponential smoothing model for the first task and a generalized linear regression for the second task given the high correlation among some of the variables or a combination of both.

To improve the second task models, recursive models can be used in a different way: after the prefixed lag number of event have taken place, the predicted values of past can be used with the model created in the first task to 'smoothen' the model's prediction.

# References


[1] Cristian Borges Gamboa J, "Deep learning for time-series analysis," *arxiv*, 2017.

[2] Ismail Fawaz, H., Forestier, G., Weber, J., "Deep learning for time series classification: a review," *arxiv*, 2018.