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 - xxxxxx - r.xx@@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 θ. 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_ids` $20, 31, 46, 54, 62, 70, 79, 72$, 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 is 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 built using `pytorch` and optimized with an Auto-ML algorithm using `sherpa` opti-

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

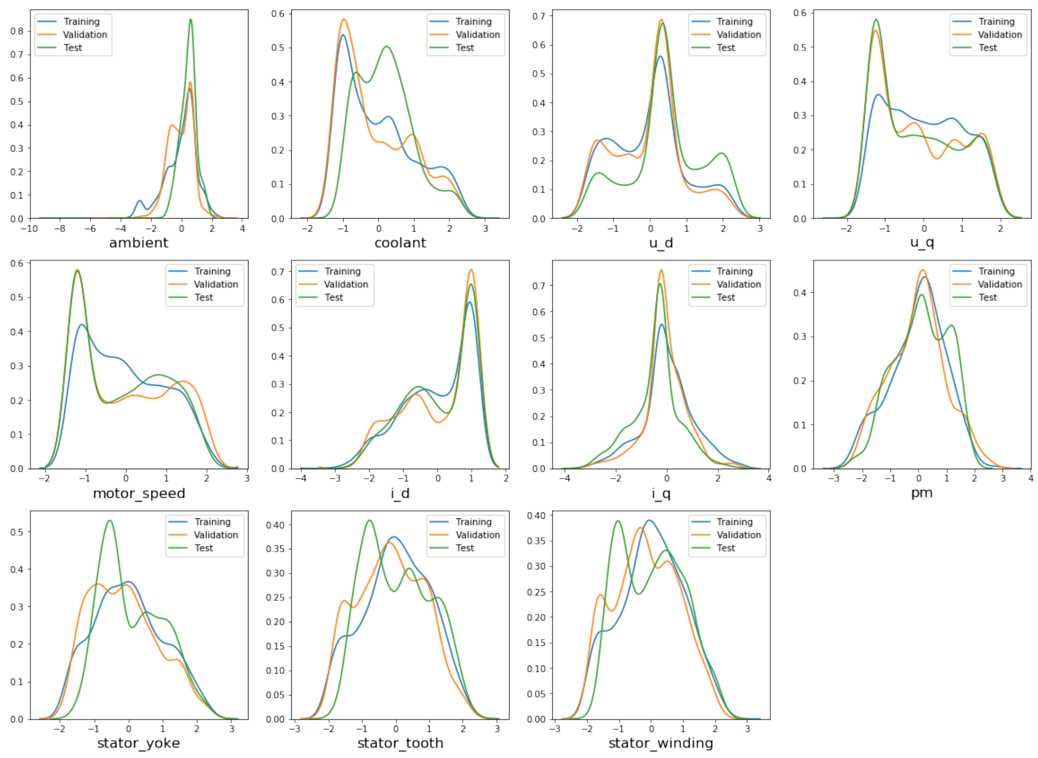

Figure 1: Distribution of the variables grouped by the division

mization library, in particular the GP surrogate model is used with the EI acquisition function to have some exploration during the optimization.

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, providing real-time information to the driver.

3.1 First Task

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

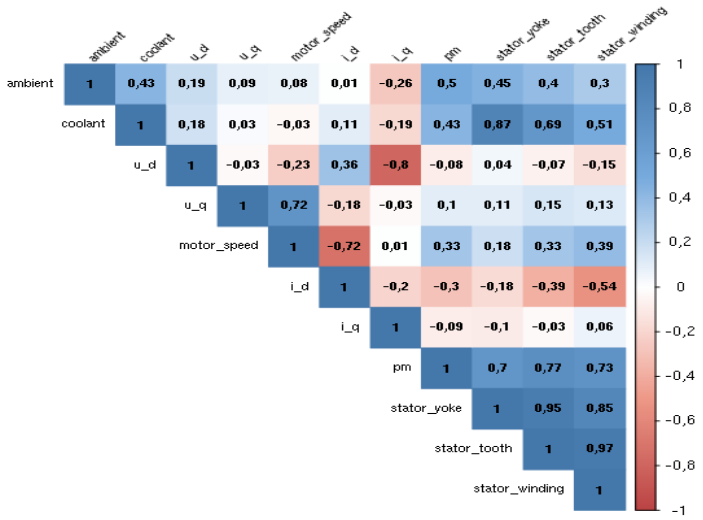

Figure 2: Correlation Plot of the considered variables

the hidden layer, data flows through two independent feed-forward neural networks (with two layers both) to estimate values for `pm` and `stators` 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. In this case two independent models are created: one to predict only the `pm` variable and one to predict the 3 `stators` variables.

In this case the window is fixed to 60 lags and 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`s.

The hyperparameter optimizer need 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 MSE and it is calculated, after one epoch of the training set, on the validation set.

CNN The convolutions happens a temporal moment, i.e. it convolves on all the variables of a certain time t.

4 Results and Evaluation

4.1 First Task

As shown in figure 3, after the optimization process LSTM and GRU yield similar results (having a similar structure), while CNN provides results comparable with the previous architecture. RNN however yields poor previsions, due to its simplicity, compared to other architectures.

However, comparing results using only the loss value is not satisfying for the task: the model have to be light enough to be easily used by a car: so architectures are plotted considering both number of parameters and performance.

For the secondary task to improve the prediction a weighted MSE where the weights depend on the target value, in particular unitary weight is used for values under 0.5 and a weight of 3 is used for values over 0.5, the results of hyperparameter optimization are shown in the figure 4.

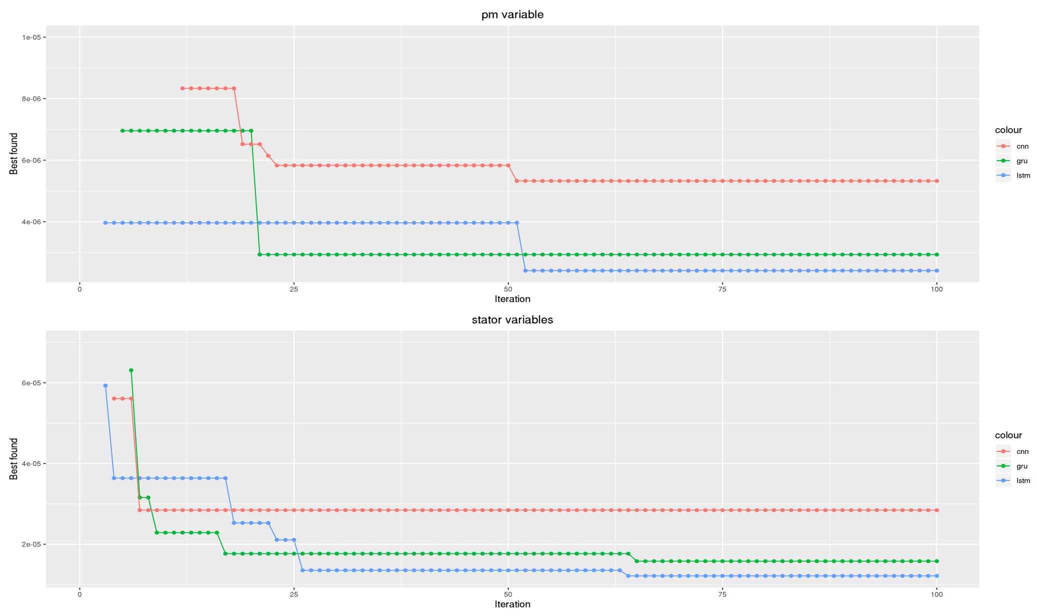

Figure 3: Results of the optimization process on the pm and the three stator variables respectively using the MSE (RNN is excluded due to its poor performance)

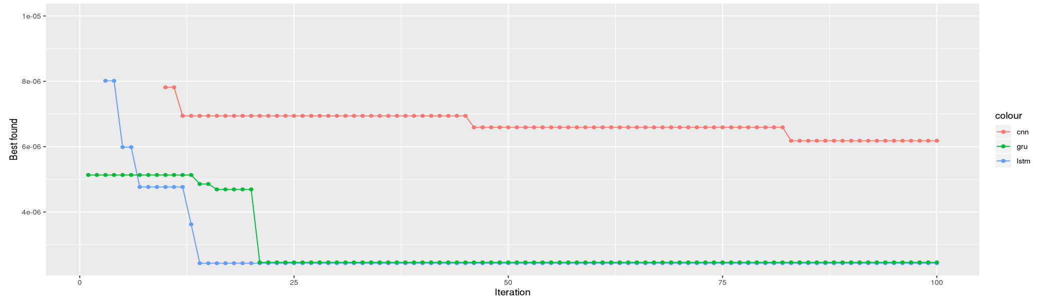

Figure 4: Results of the optimization process using a custom loss (RNN is excluded due to its poor performance)

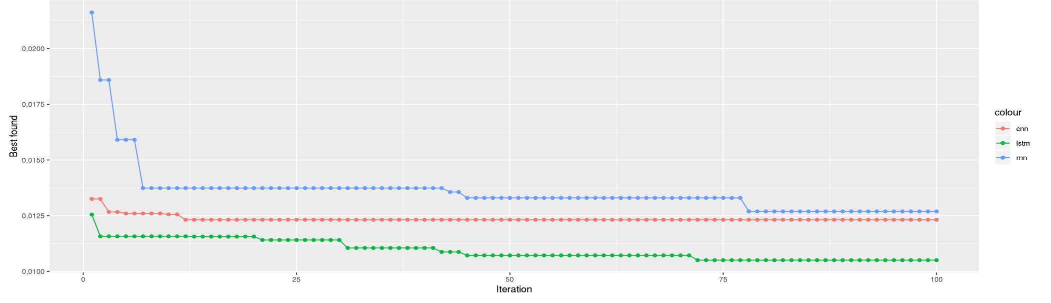

Figure 5: Result of the optimization for the second task

4.2 Second Task

5 Discussion

6 Conclusions

References