Calibration using AI

JUSTINE Jean & Omid Saberi
Högskolan Väst
EIGSI
Trollhättan Sweden
jean.justine@aol.com

***Abstract*—The work we had to do was to find the inverse function to calibrate sensors. To do so, we will train neural network, to find the best models that can predict efficiently the true value using the sensors output as training set, and the true value of the environment as target. Then when the models are precise enough we will have our calibration algorithm. We used Deep Neural Network and Recurrent Neural Network. Our results were good, the algorithm manage to get good score and grasp the pattern inside the data especially DNN. All of the code is available on GitHub here: https://github.com/Justine-IA/Deep_learning-/tree/main/Project**

I. Data Analysis

Firstly, the project will be a Deep Learning project that consist of finding an algorithm that predict true value in function of the value that a sensor finds, for that we will use a dataset that measure pollution in a city the dataset is available here: https://archive.ics.uci.edu/dataset/360/air+quality.

A. Requirement

The requirement will be to find the inverse function to find the real value, through the value detected by the sensors. The sensor data (y) is a function of true measurement (x).
We have to find the inverse function, which is: *f^{-1}(y) = x.* This is an important task because having the real measurement can be crucial to take actions later on.

B. Data

The dataset is composed of time feature, value that the sensor finds, like hourly average sensor response of CO, and also the real value (Ground Truth) like CO, not the one the sensor detected. There is no missing value according to the website but there is some not correct value like negative number of pollution or -200 for sensors temperature so we removed them as well as for the GT data, there is a lot of missing value so we just replace them with the one closer in time to them, the time and date will be used for a Recurrent Neural Network later on because the sensor is detecting pollution in a city so it will be useful to train a model with the times feature, cars or anything else influence pollution don't pass at every hour. We also have humidity and temperature in addition to all the other feature of pollution and else.

C. Visualisation

Here we have some visualization for the data such as:

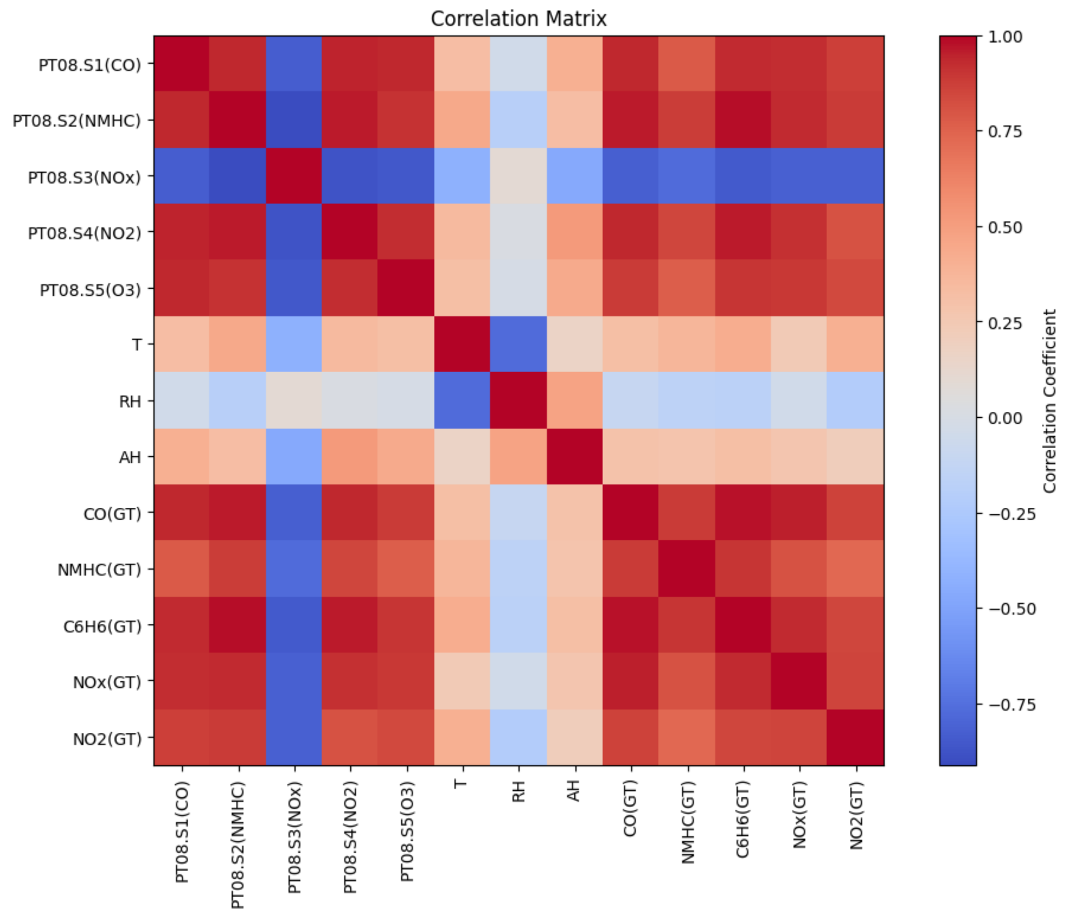

Fig. 1. Correlation Matrix

As shown in fig1, we can see the correlation between the polluting elements measured and its ground truth are very related to each other. We can also see that NOX and RH are not very related to anything but themselves.

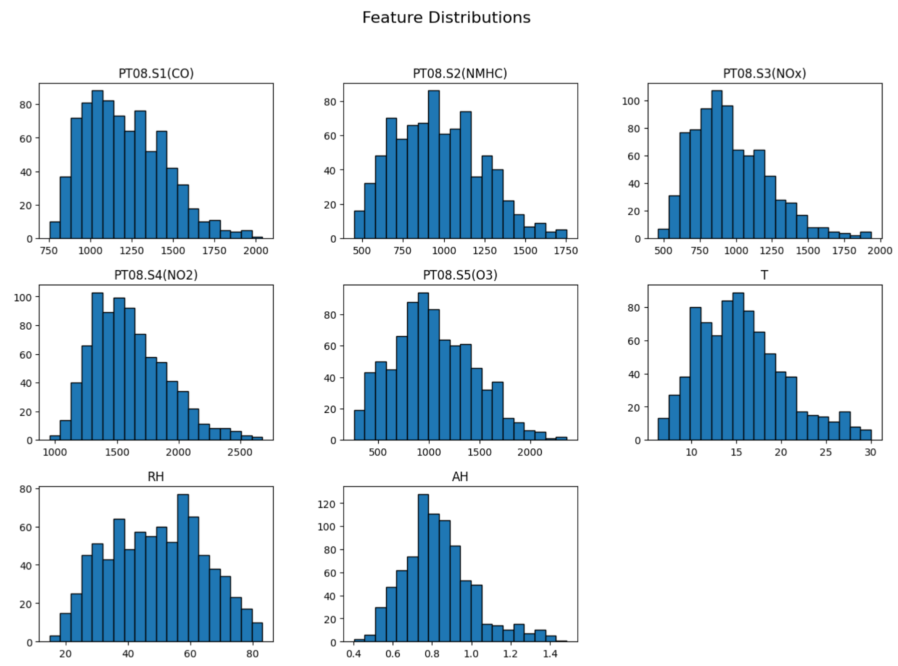

Fig. 2. Feature Distribution

We can see that in Fig 2, most of the data is correct and doesn't need further data cleaning except for a scaling later on.

D. *Requirement*

Some requirement that are in existing research. For temperature and light sensors some Neural Network (NN) were develop for [1].
Also ML based calibration can handle over time drift, and environment variability, improving long time accuracy [2].

List of requirement:

- Acquire Data
- Choosing Model
- Model Training and validation
- Figure of merits
- Documentation

II. Algorithm Design

A. *Data Modeling*

Firstly, we fetch the data, then, as said in the first part, we separate the feature form the target, y, the target, will take all of the GT value, x will take the rest, we will build a Recurrent NN so we keep time in a variable call X_RNN, whereas X will not have time or date.

After that we need to scale the data, firstly we will need to transform the data into numerical value, like 2004 for year, 10 for month, 23 for day, 18 for hour instead of the usual 23/10/2004 and 18:00, we also have day of the week, after that we will add cyclical encoding for the time and day of the week, using cos and sin to have a cycle.

After all of that we need to do the real scaling we just import standards scaler and scale X and X_RNN as well as y, we need to scale the data for having a better consistency, better metrics calculation, and it prevents certain data to be too dominant compare to others.

After we split the data between X train X test y train y test as usual but we also have X train RNN and y test RNN which contains the date and time.

Also, we had to reshape X train, X test RNN, into a 3D tensor shape with samples, time step and features, because the original data was a 2D tensor with only samples and features, and for RNN it only takes 3D tensor because it needs the time so we added time step.

B. *Algorithm Design*

I choose as algorithm**: Recurrent Neural Network** and Deep Neural Network. I choose RNN because it is design and efficient to handle sequential data, and help capturing temporal dependencies in our dataset. In our case the feature is a shape of 5 time steps. I started the model with a layer of Simple RNN using relu as activation, then 0.2 dropout, then Dense layer of 32 neuron with Relu asd activation, another dropout of 0.3 and finally the output layer with the shape of our target [3].
The other algorithm is **Deep Neural Network** (DNN), I choose this one because it is efficient at finding complex pattern in high dimensional data, and nonlinear relationship. The model has an input layer of 128 neurons, using Relu as activation, a dropout layer of 0.3, then another dense layer of 64 neurons still using Relu, once again a dropout layer of 0.2 the last hidden layer is Dense with 32 neurons still using Relu and finally the output layer is like the RNN but with linear activation [4].

C. Figure of merits

For figure of merits I choose, **Mean Square Error** (MSE) for the loss, and **Mean Absolute Error** (MAE) for the metrics.
We also had **R², **and **RMSE**.

D. First Results

I used optimizer Adam for both algorithm and I got those results:

1) RNN Performance

- Test RMSE: **0.421**
- Test R2**: -0.834**
- Test Loss (MSE): **0.1773**
- Test MAE**: 0.2411**

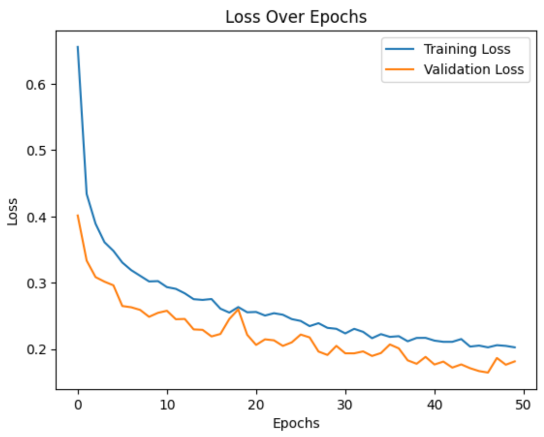

Fig. 3. Loss over epochs for RNN

2) DNN Performance

- Test RMSE: **0.452**
- Test R2: **0.798**
- Test Loss (MSE): **0.2387**
- Test MAE**: 0.2565**

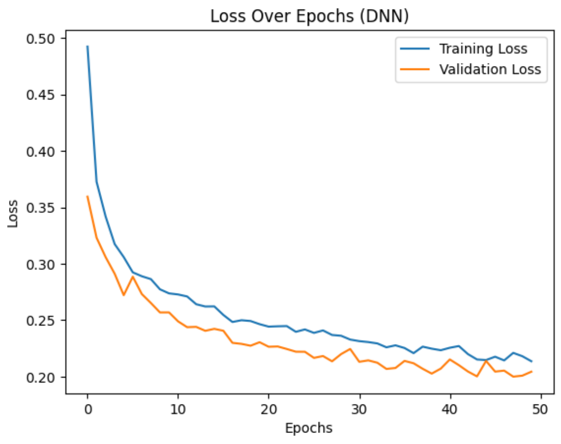

Fig. 4. Loss over epochs for DNN

We can see that the RNN outperform the DNN both in MAE and MSE, the DNN is slightly below but is still performing well even on sequential data. Also as seen in Fig 1. And Fig 2 the graph shows us that both model learn pretty well, they decrease the loss well over the epochs, the validation loss show that the model generalizes well on unseen data. But there is still some improvement that can be done.

The two RMSE show that the models manage to capture the data, the R² of DNN is also good and attest of the DNN understanding, R² of RNN show that there are still room for improvement and didn't get some data.

E. Improvement

To improve the two models, I choose multiple way:

- **Increasing complexity:**
  For the DNN I increased complexity by adding more neurons in the dense layers, to 256, 128, 64.
  For the RNN I did the same thing but with less neurons, I increased to 128 64 and 32.
  The reasons for that is: adding more neurons, improve the models ability to understand more complex and nonlinear pattern in the data, also it's better to understand high dimensional data [5].
- **Change Simple RNN to LSTM in the RNN model:**
  I changed it and increased the number of neurons to 128, the reason for that is that LSTM is better at understanding temporal patterns, as well as long term dependencies [6].
- **Increase Dropout Rate to 0.5 in total instead of 0.4:**
  Dropout help to prevent over fitting, we did not have it with the first models but with more complexity and more epochs it could have happen so we decided to increase the dropout rate from 0.4 to 0.5 [7].
- **Increasing the number of Epochs and batch size:**
  We increased the number of epochs from 50 to 100, and batch size from 32 to 64 in the goal of allowing the model to fully converge because the loss did not attain a plateau of no amelioration. We increased Batch size to stabilize more efficiently the gradient.

F. Results of Improvement

We will discuss the results of the improvement after incorporated all of the above

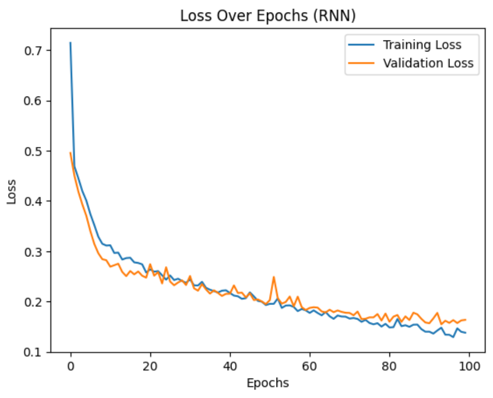

Fig. 5. Loss over epochs for more complex RNN

As seen in Fig 5. The validation and training loss stabilize around 70 epochs, the two lines are very close indicating that there is no overfitting and we had those results:

- Training MSE : 0.1379
- Validation MSE : 0.1635
- New Test MSE : 0.1583
- New Test MAE : 0.225
- RMSE: 0.36
- R2: -0.763

Compare to what we had first there is a slight improvement, our results were already good but become even better. Showing that the LSTM layer is more efficient than the previous Simple RNN layer, as well as more epochs and more neurons provided better results, also there is no overfitting, thanks to the increase in the dropout rate. But the R^2 score is still negative. The RMSE show also a good understanding [8].

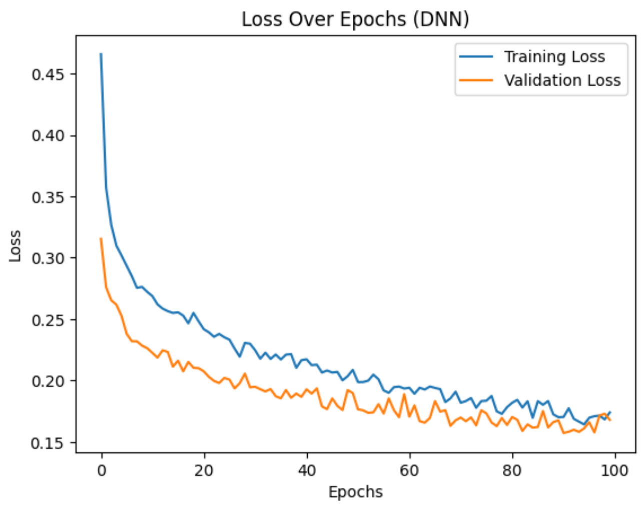

Fig. 6. Loss over epochs for more complex DNN

In Fig 6. We witness the growth of the new and more complex DNN, over 100 epochs. In this graph we can see the model is generalizing well, around 80 epochs the validation stabilizes showing good performance on unseen data. As for the training, it continues to decrease but is lower than the validation set showing a slight overfitting.

We had those results:

- Training MSE : 0.174
- Validation MSE : 0.168
- Test MSE: 0.162
- Test MAE: 0.227
- RMSE: 0.415
- R2: 0.829

We had slightly worse results than the RNN but still very efficient. The improvement we did had an impact and improved the model, more neurons helped as well as more epochs, we could increase even more the dropout rate to prevent overfitting but as it is minimal there is no need here. The R^2 score is better than the RNN showing more understanding of patterns [8].

III. Consolidation of results and Future Plan

A. *Discuss of the results*

The results I obtained both from the RNN and DNN can be labelled as good.
The reasons are as follows:
The RNN model had 0.1584 as test MSE and 0.225 as test MAE, showing it captured the temporal dependencies in the data. The use of LSTM layer significantly improved the model and its ability to detect pattern.
The Dropout rate also played a big role to not overfit even with that many epochs.
It can be considered as good because all the change makes it nearly optimal, and other addition might decrease the results and the learning of the model.
As for the DNN, it has results of Test MSE:0.162 Test MAE:0.227, showing us it grasped the pattern inside the data and had correct results even though it doesn't have any sequential modeling (although we removed time and date for the DNN) the R^2 score is even better than before adding to the proof that this DNN is more efficient.
Same as for RNN, the model is nearly optimal, adding more complexity or anything else is taking a risk that it will decrease the model results, there is still some improvement we can suggest but it might still lower the results.

B. *Future work*

For the future work, to try to improve even more the model is:

Having more data to train on, we only had 9000 correct data after cleaning, if we had a bigger dataset it could help us improve it.

We could also replace the missing value with some technique like KNN but we did not have much of bad data so it won't help that much more.

We can also do more hyperparameters tuning like grid search or changing the learning rate even though the optimizer Adam is very efficient for that.

Exploring more advance models like GRU for the RNN or ensemble methods for the DNN can be a way to improve it.

Lastly, we could test the model on others data, in a similar environment that we had.

IV. Conclusion

In this project, we had to analyze a dataset, clean it and then train two different models on it. The goal of the project was to find the inverse function that allow us to find the true value of the sensors instead of the sensors value, for that we had as Features pollution data and temperature humidity and relative humidity and as target all the Ground Truth value of those pollution.

The data cleaning was pretty complex, there were technically no missing value but they were replaced by incorrect value so we had to create mask to filter them out or replace them like for the Ground Truth value. After we changed the date and time, we scaled it using a cycling scaling and then scale the whole data to train it. Also we had to reshape the training data for the RNN to fit in the model.

Then we created the two models a RNN and a DNN, which had correct results at first with Test Loss (MSE): 0.1773, Test MAE: 0.2411 for the RNN and Test Loss (MSE): 0.2387 Test MAE: 0.2565 for the DNN, our models were simple but worked. And were quick to train so modifying them was easy. The RNN had an input layer with Simple RNN and two dense layer and 2 dropout layer, the DNN was 4 dense layer and 2 dropout layer. We used the same metrics for both: Adam as optimizer, MSE for loss and MAE for metrics. We used 50 epochs to train them and 0.2 validation split as well as 32 for the batch size.

We needed to improve those results, so we increased the complexity for both models, for the RNN we replace it with LSTM, we increased the number of neurons and dropout.

For DNN we also increased complexity, we increased number of neurons and dropout rate.

In addition to that, I double the number of epochs to 100, and same for the batch size to 64.

Thanks to those change our models had as results:

For RNN:

- New Test MSE : 0.1583
- New Test MAE : 0.225

For DNN:

- New Test MSE : 0.162
- New Test MAE : 0.227

Which is better results and took the same time for computation.

We suggested some way to have better results but some we cannot do them right now like having more data.

To conclude, we had good results, we achieve the goal of the project of improving the models and having the inverse function which is just the models algorithm.

Also I had to ask AI for help on reshaping the data for the RNN models as well as for the cycling encoding of the time. I also used AI to plot histogram and helped me when I had an error and couldn't find the solution.

All the code is available on my GitHub here:
https://github.com/Justine-IA/Deep_learning-/tree/main/Project

References

[1] Y. Zhang, L. O. H. Wijeratne, S. Talebi, and D. J. Lary, "Machine Learning for Light Sensor Calibration," Sensors, vol. 21, Sep. 2021. Available: https://www.mdpi.com/1424-8220/21/18/6259.

[2] Z. Wang, "Evaluating the Efficacy of Machine Learning in Calibrating Low-Cost Sensors," Applied and Computational Engineering, vol. 71, 2024. Available: https://www.ewadirect.com/proceedings/ace/article/view/15136.

[3] S. Smyl, "A hybrid method of exponential smoothing and recurrent neural networks for time series forecasting," *International Journal of Forecasting*, vol. 36, no. 1, Jan. 2020.

[4] F. Kastner, B. Janßen, F. Kautz, and M. Hübner, "Exploring Deep Neural Networks for Regression Analysis," in *Proceedings of the International Conference on Performance, Safety and Robustness in Complex Systems and Applications (PESARO)*, 2018

[5] M. Wager, S. Wang, and D. Liang, "Investigating the Relationship Between Dropout Regularization and Model Complexity in Neural Networks," 2021. [Online]. Available: https://arxiv.org/abs/2108.06628

[6] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," Neural Computation, Nov. 1997. [Online]. Available: https://arxiv.org/pdf/2408.10006

[7] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A Simple Way to Prevent Neural Networks from Overfitting," Journal of Machine Learning Research, vol. 15, pp. 1929–1958, Jun. 2014. [Online]. Available: https://www.geeksforgeeks.org/training-neural-networks-with-dropout-for-effective-regularization

[8] Akshita Chugh MAE, MSE, RMSE, "Coefficient of Determination, Adjusted R Squared" 08/12/2020 [online] Available: https://medium.com/analytics-vidhya/mae-mse-rmse-coefficient-of-determination-adjusted-r-squared-which-metric-is-better-cd0326a5697e