Age from brain Data

Project presentation

Marco Accerenzi

Dipartimento di Fisica

Esame di Computing Methods for Exerimental Particle Physics and Data Analysis, 12 Settembre 2024

1 Introduction
- Libraries
- What's in the code

2 Data extraction and manipulations
- Data manipulations
- The effect of normalisation

3 The regression model
- Callbacks
- Optimizing the hyperparameters

4 Results and conclusion
- Comparing the results

Introduction to the project

The objective of this project was to develop a deep learning regression model, trained to predict the brain age of patients from data extracted from MRI scan of their brains.
The model implemented in the project is capable of predicting the age of the patients in the data set with a RMS of about 6 years.
The best performing configuration for the model was a 2-hidden layer deep learning model, with 6 nodes per layer. The model trains and predicts at its best when the data is standardized and normalised using the methods provided in the project.

Libraries

- I used the *keras* library for the regression model to write a problem specific implementation.
- To read and manipulate the brain data, provided in .xlsx files, I used *pandas*.
- For plotting the results and visualizations I used matplotlib.
- I used the *os* and *sys* modules from the python standard library to get and manipulate paths and directories.

What's in the code

The source code for this program contains several classes and some scripts, as to provide a framework to use the input MRI data from the "raw" excel file to produce a trained model capable of making predictions, evaluating and visualizing those predictions.

- The ExcelData class is responsible for reading the excel file and providing suitable input for the model.
- The RegressionModel class contains the model itself. It implements methods for compiling, training, saving and loading regression models.
- The two "Console" scripts provide a basic UI to simplify the use of the project, allow the user to train a model and use it to make predictions without changing or writing any code.

What's in the code 2

- The Optimizator script allows to quickly choose values for the models hyper parameters.
- The Analysis script takes a .xlsx file containing the real labels and the predicted values to calculate the RMS and MSE of the predictions. It also allows to compare the data from the different groups in the original data set.

Data extraction

The MRI scan data for model training was provided for this project in the form of two .xlsx files, containing the features of 915 patients (or samples). For each sample the file also contains the real age at scan value (label) and the name of the group who provided the data for the sample (group name).

- Using the *read_exce l* function provided by the*Pandas* library the class creates a dataframe representing the data in the input file, including the column labels.
- The *to_numpy* method of the dataframe is then used to extract the numerical data to feed the model and is kept as a property of the ExcelData instace

All manipulations to the data is performed on copies of the *numpy* array. The samples can be shuffled before training to prevent overfitting, allowing to produce genuine predictions on the same test+validation data used in training.

Data manipulations

The "raw" data contains several blank cells and the various features have very different value ranges. To combat this issue the data is standardized and normalized.

- As the excel file is read each blank cell is filled with a -9999 value.
- All columns of the array are normalized: each cell is divided by the max value of each column. In this calculation the cell containing a value of -9999 are excluded.
- The model will train itself to ignore the -9999 values. This allows the use of a larger data set compared to the one obtained by deleting all samples with a missing feature.
- Normalisation and standardization greatly reduce the time needed for training. Instances of the model trained with normalized data make on average more accurate predictions on the data than instances trained without normalisation.

The effect of normalisation 1

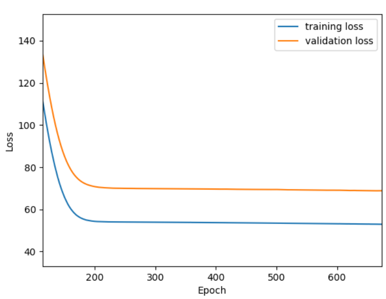


Figure: Plot of the first training epochs using normalised data.

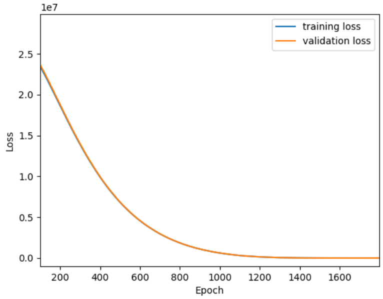


Figure: Plot of the first training epochs using raw data.

The effect of normalisation 2

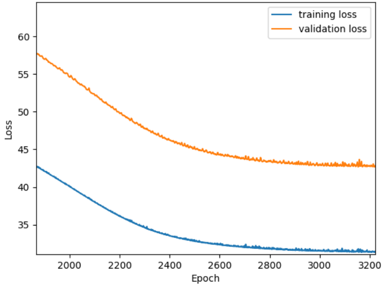


Figure: Plot of the last training epochs using normalised data.

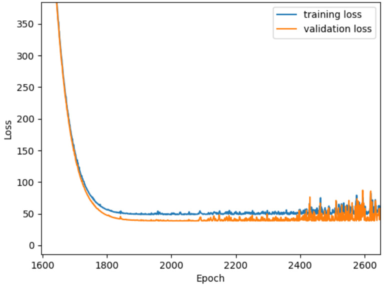


Figure: Plot of the last training epochs using raw data.

The regression model

The regression model class uses the *models*, *layers* and *callbacks* module from the *keras* library. It implements several methods:

- The Compile_Model method defines the instance of the model, it sets up the layers and nodes of the model and fixes the shape of the input data. After the Compile_Model method is called the model is compiled and ready for training.
- The Start_Training method trains the model over some data, using the supervised training algorithm provided by the *keras* library. The number of epochs and other parameters of the training can be inputted by the user.
- The Plot_History and Save_Model are meant to be used after Start_Training, to save the training history and the model itself.
- When initializing an instance of the model it's possible to use the *load_model* function to load a trained model: this model can be used only for inference and not for further training.

Callbacks

To optimize the training phase several callbacks are used:

- Early stopping: This callback stops training if, after a user-set number of subsequent epochs, the loss functions hasn't diminished. It allows to stop training before overfitting. After stopping it restores the best performing weights.
- ModelCheckpoint: This callback saves various "checkpoints" during training. It allows to keep a backup of the training progress.
- ReduceLROnPlateau: This callback reduces the *learning rate* of the model if the training reaches a plateau (meaning if for a certain number of epochs the loss function stops diminishing). It reduces the amount by which the model's weights are changed during training, allowing for further training after reaching plateaus.
- PrintProgress: This is a modified version of Callback from the *keras* library. It prints the loss functions every n epochs. The n parameter is set by the user through a "verbose_training" parameter of the Start_Training method.

Optimizing the hyperparameters

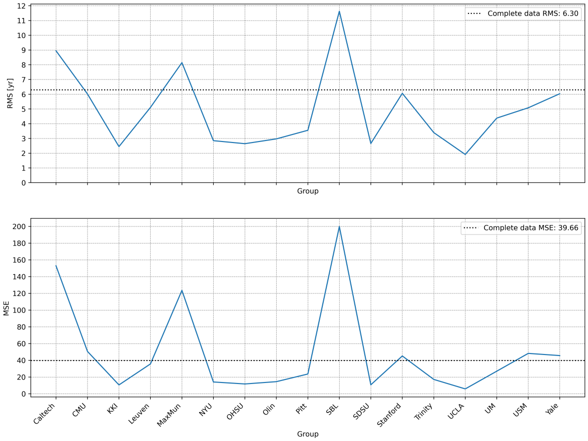
The Optimizator.py script was used to choose the hyperparameters.
It loops through several “reasonable” values of the hyperparameters and performs a shorter training run, the training history can then be evaluated by the user to choose the hyperparameters or to refine the Optimazator’s range of parameters.
The best performing model for this project was a 2-hidden layer model with 6 nodes per layer. This configuration had the fastest training time and overall the best average prediction performance.

Predictions

The trained model was used to evaluate the input data and predict the ages of the patients. Each training run produced slightly different models, the models trained with the optimal configuration have an average RMS of 7.1 years, the best performing model has a RMS of 6.30 years and a MSE of 39.66 over all the input data.
Repeated training could improve the average performance of the set of trained models but it is unlikely that these model could predict on the input data with a lower than 6 years RMS without gross overfitting.
Only the best performing model will be considered for the next analysis.

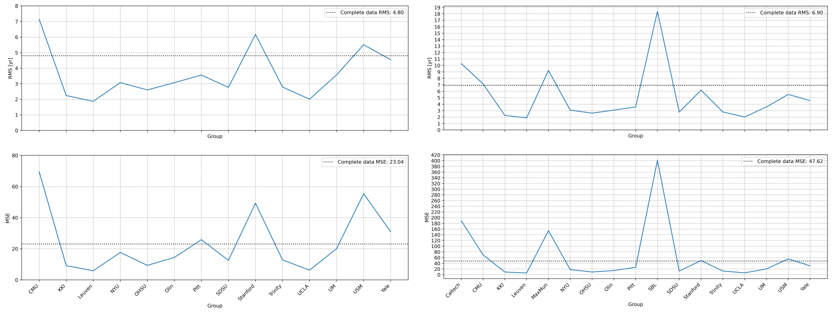
Comparing the results

The performance of the model on all the different groups is compared in the figure below: the prediction error for the data of the "Caltech", "MaxMun" and "SBL" groups is vastly above the average.



Comparing the results 2

Repeating the training on a reduced data set by removing the three outlier data groups doesn't significantly improve the performance of the model: the trained model performs similarly on the remaining data groups but significantly worse for the three outliers.



Git and the repository

The project uses a public github repository. It has a main branch and three active branches: develop, feature/docs and feature/presentation.

Documentation

The documentation for the project was generated via Sphinx, using its *autodoc* functionality to extract the docstrings from the source code. The public GitHub repository allows to distribute the projects documentation through ReadTheDocs and GitHub's own Pages.