Comparison between DL models' performance on the regression task based on the UCI Heart Disease Dataset

Master's degree in Artificial Intelligence
A.Y. 2024/2025

Machine Learning course
Author: Averoldi Lorenzo, VR521277

1. Intro and motivation

Statistically, parkinson, with Alzheimer's, is one of the most frequent degenerative diseases in the world. In Italy, there are more or less 300.000 persons with parkinson, and this number is growing, together with the aging of the population.
At this day, there is no cure for parkinson; the only thing medicine can do for patients is try to treat symptoms and improve their quality of life.

The task in this case is regression, trying to predict parkinson's progression in time, in particular the 2 progression indexes: motor UPDRS and total UPDRS (Unified Parkinson's Disease Rating Scale).

The motivation of this work is to provide help and support for diagnosis and clinical monitoring in the long term.

2. SOTA

With this kind of dataset and task, SOTA isn't defined universally like more famous and widely used benchmarks (like ImageNet).

Using MSE as a metric for comparison, the baseline, used also in this work, is represented by a simple MLP, which doesn't consider the sequentiality of the input data and achieves an MSE between 5 and 15.
Then, in the scoreboard, we see the models we use also in this work, Vanilla RNN, LSTM, and BiLSTM, which will be seen in detail in this project.
To consider, there are also Pure Transformer (0.1-0.4 MSE) and LSTM + Attention + Ensemble, which represent the current SOTA with an MSE lower than 0.1
In this work, with a BiLSTM architecture, we will reach SOTA results with relatively low training time.

3. Objectives

The main goal of the project is to compare the performance of different Deep Learning architectures for the regression task.
I implemented 5 different architectures from scratch, 1 used as a baseline (MLP), and the other 4 to see which model would perform better on the UCI Parkinson dataset.
The models will be trained to predict the 2 targets, in this case separately, which represent the Parkinson's progress in the patient.
The input is a window of 5 telematic recordings of the patient, and the model has to predict the fifth label.

4. Methodology

The work is divided into 2 main phases:

1. Data Preprocessing:

First, download the dataset and import the libraries we need to run the project.

The dataset is composed of 19 features, **age** is the only integer one, while the others are continuous.

The entries are biomedical voice measurements from 42 patients with parkinson in the early stage, who joined a recruitment for a trial of a telemonitoring device for remote symptom progression monitoring.

Each row corresponds to one of 5,875 voice recordings from these individuals. The main aim of the data is to predict the motor and total UPDRS scores ('motor_UPDRS' and 'total_UPDRS') from the 16 voice measures.

Then we a preprocessing for the MLP baseline and a different preprocessing for the other models:

MLP Preprocessing: First, we divide the dataset into features and targets, and then, using 2 standard scalers, one for features and one for labels, we scale the training data.

Then, we create a dataloader with the training dataset, and we are ready to train the MLP.

RNN and LSTM Preprocessing: Similar to MLP, but the difference is in how we organize the data.

First, we group each row for the subject, then define the parameters of the sliding window.

We pass the grouped dataset through a cycle that, for each subject in the dataset, we append to the lists that will contain X and Y sequentialized.
Now, each element of X_seq contains a sequence of 5 observations in order of time, and y_seq contains the 5th label; in this way, our model will consider time correlation between data.
Then we flatten X_seq and scale it, and scale also y_seq, using again a standard Scaler.

2. Definition of the models:
Now we define the 5 models we will use in our work. I created each model from scratch, in detail:

MLP (baseline): A simple Multilayer Perceptron to use as a baseline, it's composed of 3 linear layers fully connected with 2 ReLu between the 3 linear layers.
It uses the MSE as a loss function and the Adam optimizer with a learning rate of 0.001.

The training phase it's composed of 2000 epochs for every model, and the MLP after this training phase reached, as expected, mediocre performance, but that's ok because we need it just as a baseline.

Vanilla RNN: A simple recursive neural network, we expect to see a slight improvement compared with MLP, because in this case we work with windows of 5 observations instead of a single entry.
As weights, we use 2 linear fully connected layers, **Wxh** for the input sequence and **Whh** for the memory in time.

Whx applies a linear transformation to the input sequence Xt at time t, while Whh applies a linear transformation to the hidden state at time t-1.
We use a hidden state, constantly rewritten during the computation of the output.
The fact that H is constantly rewritten is a limitation for the model, because it can use only the previous hidden state to compute the output, not considering all the hidden states that could be meaningful also for next predictions.

As predicted, after 2000 epochs of training, the model slightly increased performance compared to MLP, but it's again mediocre.
It uses the MSE as a loss function and the Adam optimizer with a learning rate of 0.001.

Vanilla LSTM: Similar but very different from RNN: it includes two states: a hidden state and a cell state, both vectors of length n, the cell contains long-term information, and LSTM can read, write, or erase from the cell. The selection of which information is erased/written/read is controlled by three corresponding gates, vectors again of length n.
In this case we have 4 Fully connected linear layers: **Wf** that make possible to LSTM to forget information not considered meaningful, **Wi** that decides what new information (from previous hidden state and new input data) store in the cell states, **Wc** that updates the value of the cell and **Wo** that computes the value to send through the output layer to get a final prediction (or the new input).
It uses the MSE as a loss function and the Adam optimizer with a learning rate of 0.001.

After 2000 epochs of training, performance improved a lot, reaching a near-optimal MSE, but we can do better.

Attention LSTM: The mechanism it's identical to Vanilla LSTM, but we also have a list that during the forward pass accumulates and memorizes all the hidden states.
Then, in the main LSTM, we compute the attention weights from that output sequence and the context vector from the attention weights multiplied by the output.
Then we use a last linear layer for the final prediction.
It uses the MSE as a loss function and the Adam optimizer with a learning rate of 0.001.

After 2000 epochs of training, performance slightly improved, but it's very similar to the vanilla LSTM, and again, we can do better.

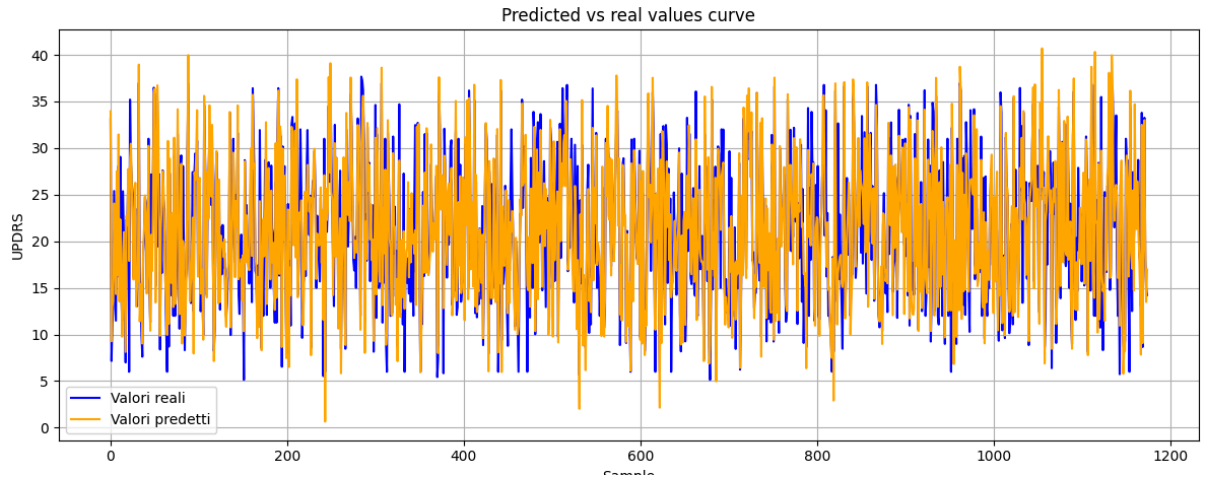
Bi-LSTM: In this case, the approach is again similar, but yet different from the previous LSTM.
It uses a vanilla LSTM and an LSTM that works backwards, meaning it takes in input the reversed sequence, then we get the final hidden states from the 2 LSTMs and concatenate them together, to finally pass the concatenated state through a last linear layer.
It's powerful, but slow, it takes twice the time of training respect to a normal LSTM, because it has to train 2 LSTMs instead of just one.
It uses the MSE as a loss function and the Adam optimizer with a learning rate of 0.001.

After 2000 epochs of training, the model reaches SOTA results in terms of MSE and R2 score, and a relevant improvement with respect to previous models.

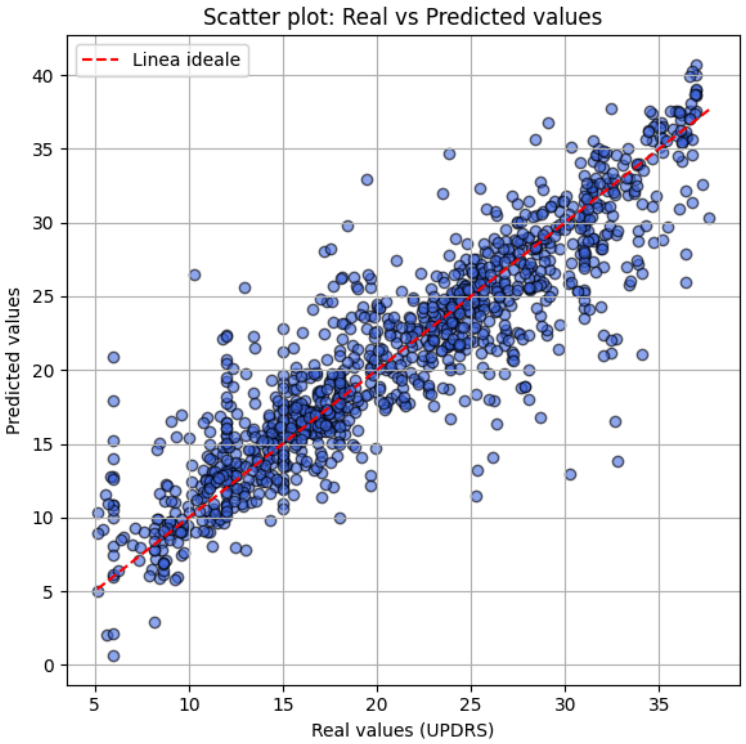
5. Experiments & Results

The evaluation for each model it's simple: first run a training cycle composed by 2000 epochs of training and using MSE as a loss function and adam optimizer, then proceed to evaluate the model using 3 quantitative results (MSE, MAE, R2 score) and 2 qualitative results (A simple plot and a scatter plot)

MLP

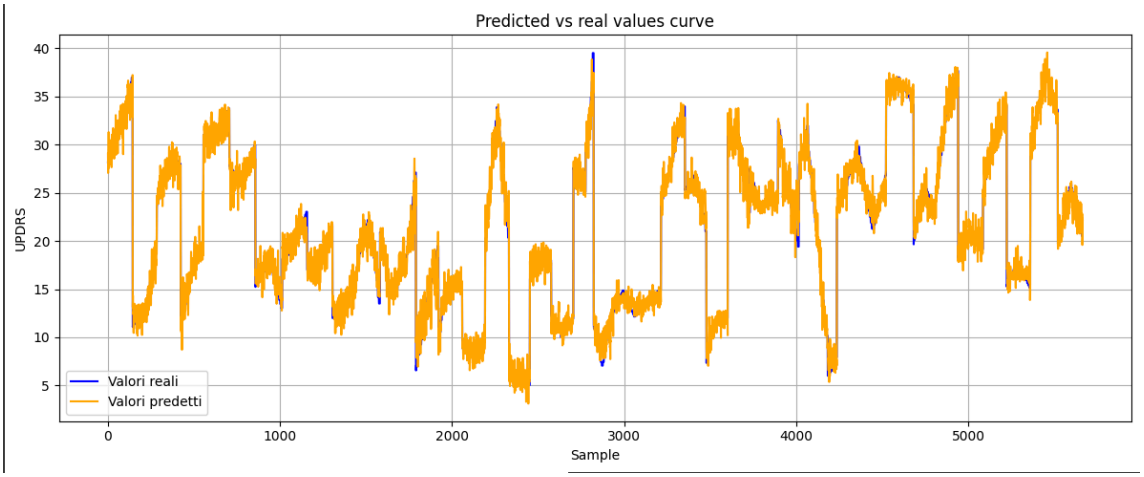



```
Test MSE: 12.00
Test MAE: 2.41
R² Score: 0.8121
```

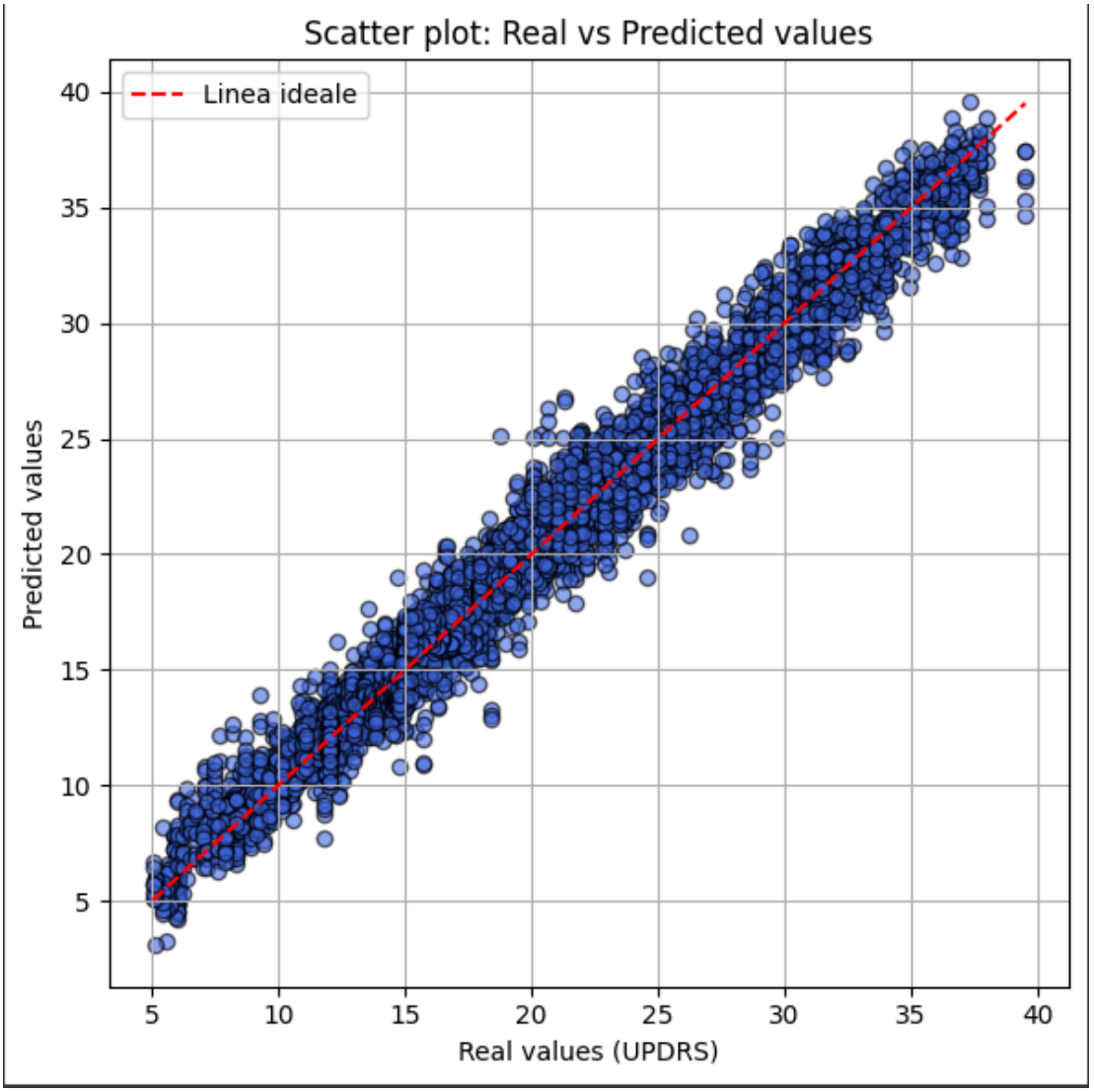


The plots and the table give us the baseline to make a comparison between all models: the MSE and R2 score are high, and the plots show the inaccuracy of the model on the data, caused by considering every sample singularly and not as a sequence.

Vanilla RNN



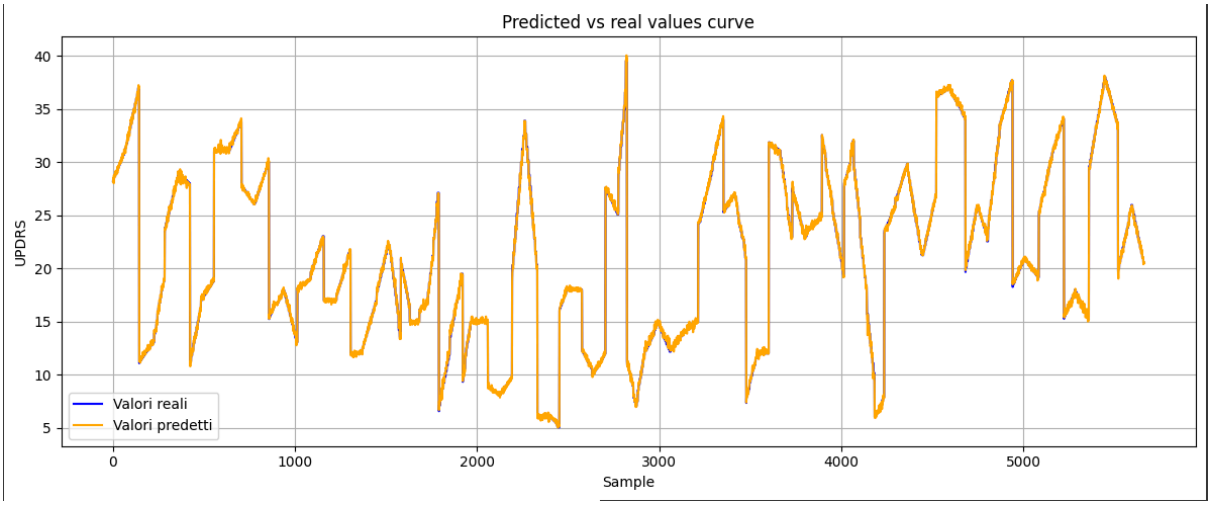

```
Test MSE: 1.54
Test MAE: 0.95
R² Score: 0.9767
```



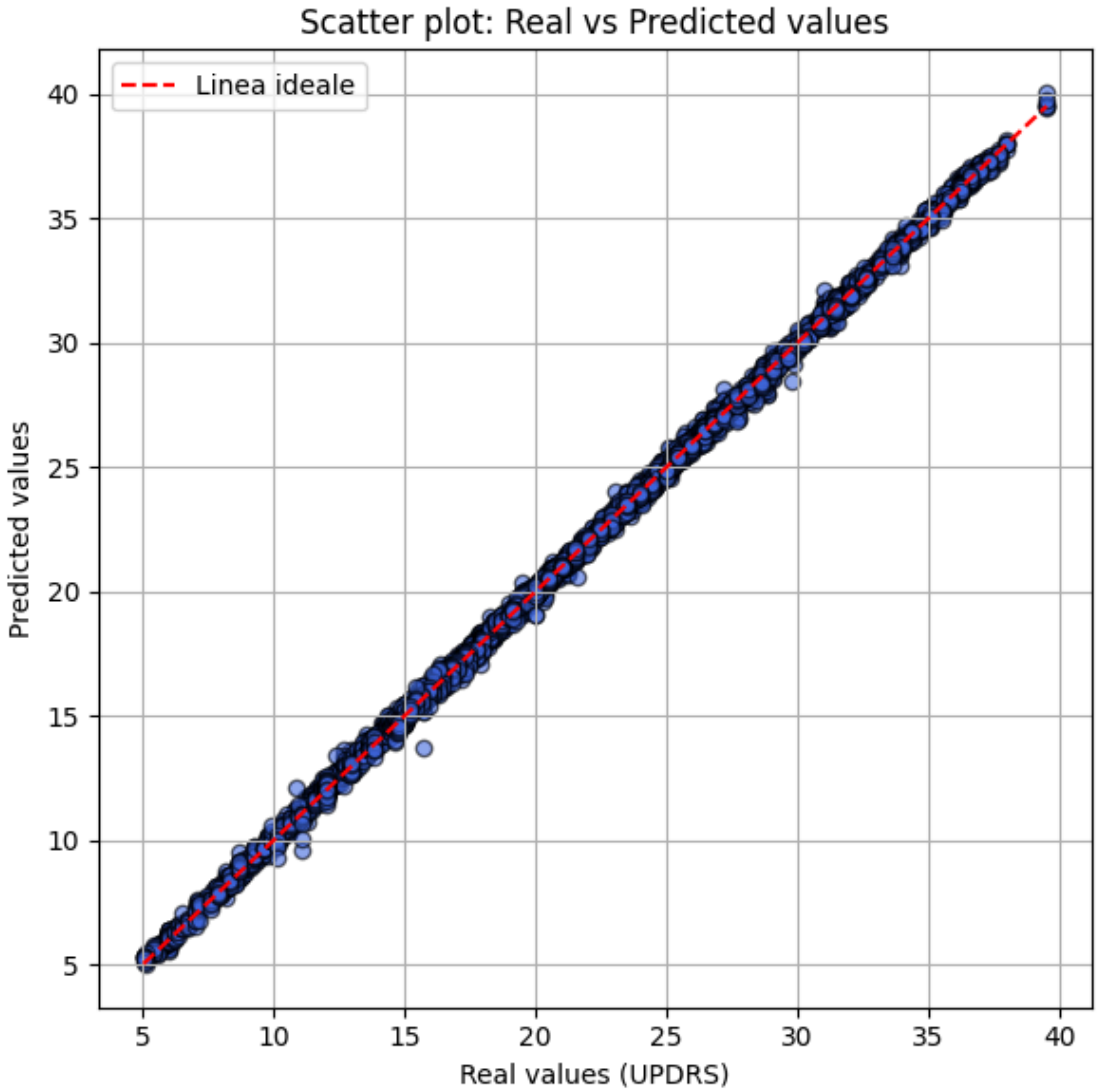
As we can see from the plots, performance improved significantly from the MLP performance, from 12.00 MSE and 0.8 R2 score it reached 1.54 MSE and 0.97 R2 score

This resulted because we implemented the **memory** mechanism, and we use sequence data in the training phase of the model.

Vanilla LSTM

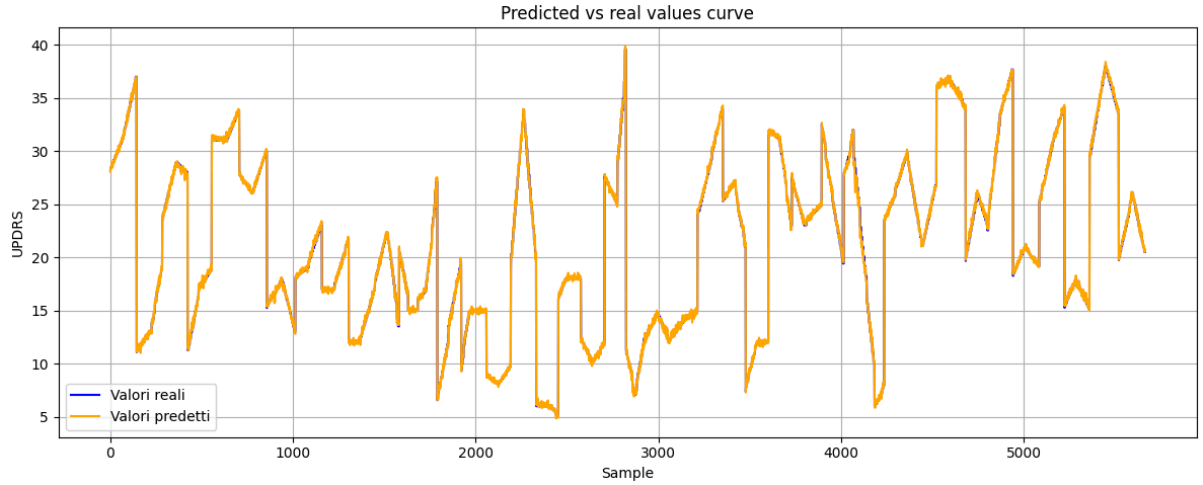


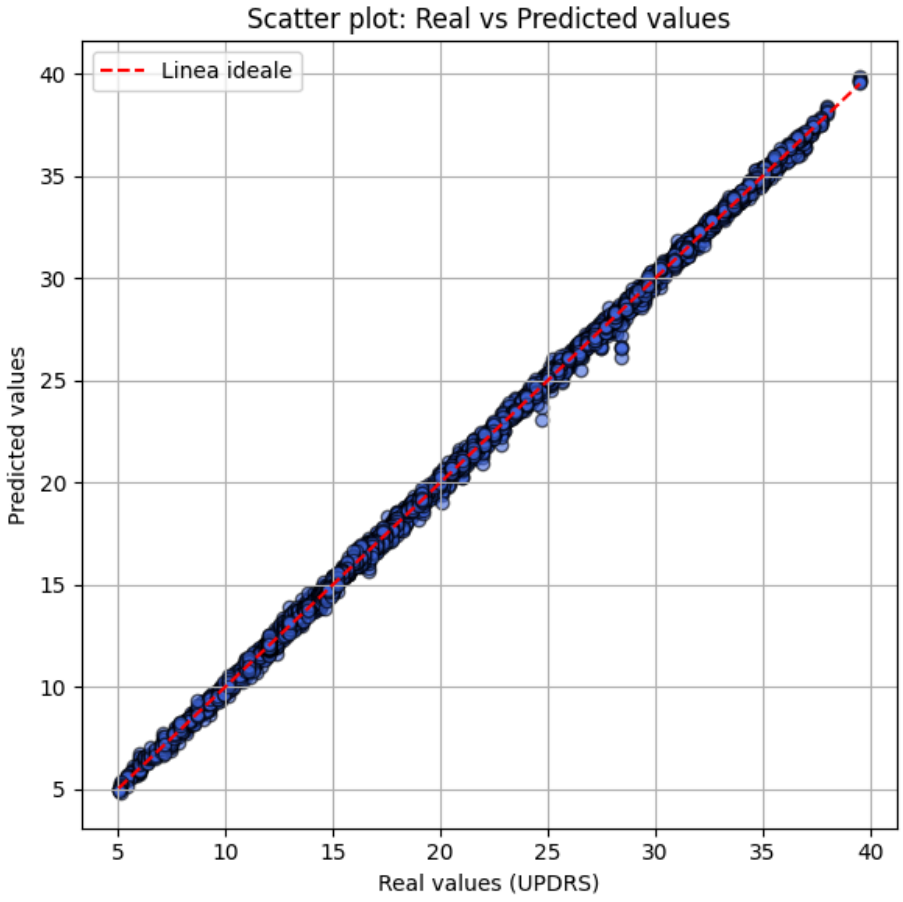

```
Test MSE: 0.04
Test MAE: 0.15
R² Score: 0.9994
```



The plots and the table show another strong improvement in the metrics, reaching an MSE of 0.04 and an R2 score of 0.994.
This is the result of implementing a more sophisticated memory mechanism that considers the sequence and not only the last hidden state.

LSTM ATTENTION

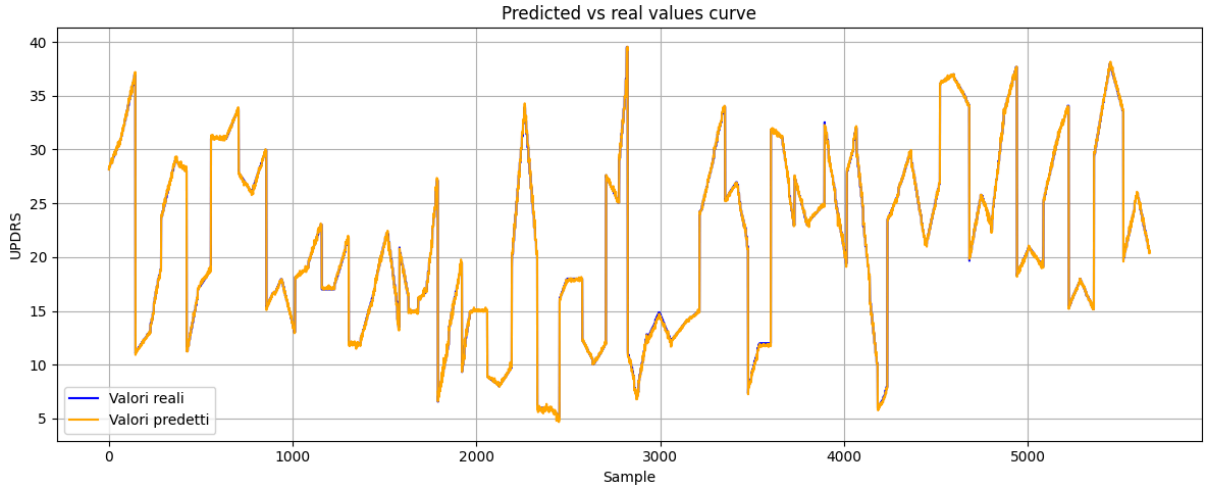


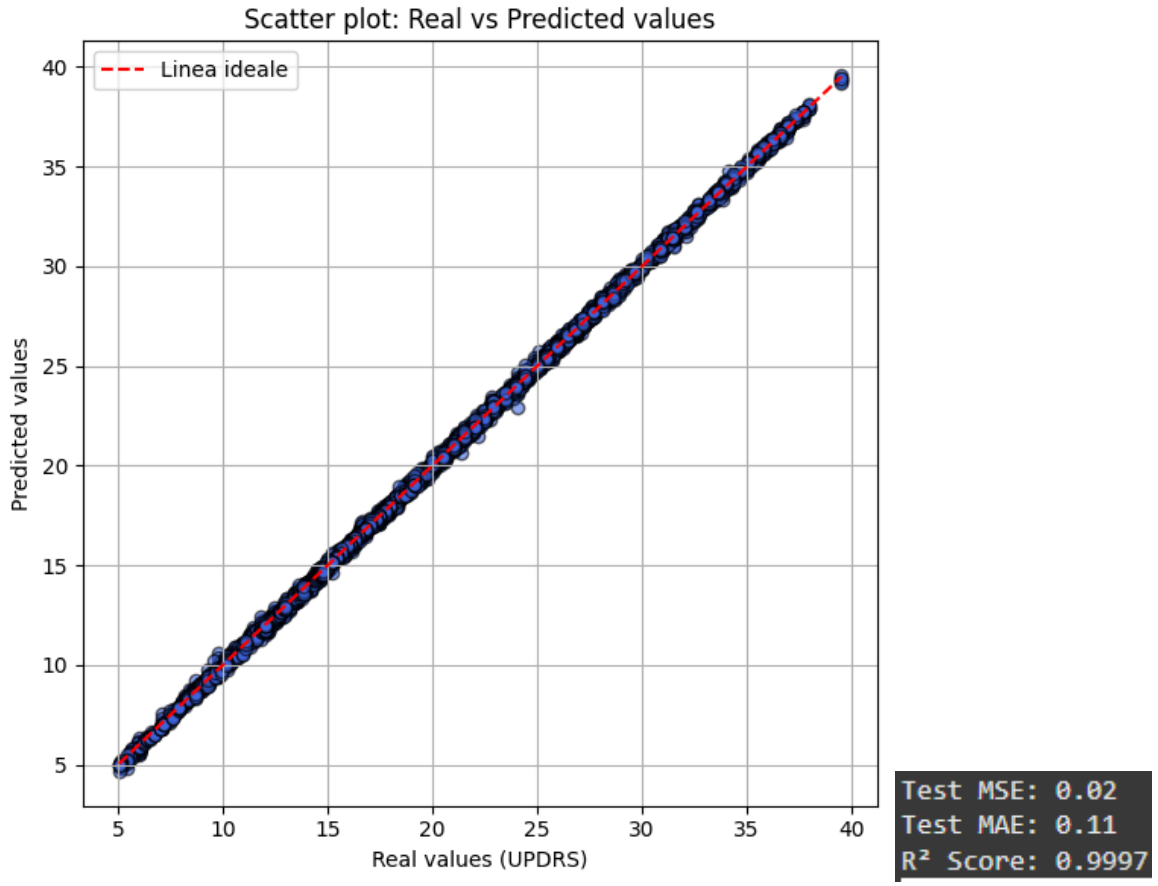


This is a strange result: implementing attention, the performance downgrades.

The downgrade is negligible, but this happened because of the randomness of the training; probably doing 100 times the training for both models, in 70% of the cases, the attention-LSTM would have at least the same performance.

LSTM-BI





The result shown by the plot it's the best so far, and it's perfectly in line with the SOTA.
The model reached an MSE of 0.02 and an R2 score of 0.9997, and these are optimal results.
These results are given thanks to the use of a backwards LSTM, giving the memory mechanism a strong evolution.

6. Conclusions

With this work, I tried to compare the performance of 5 regression models on the UCI Parkinson dataset, and see which one performs the best.
The models we used for a comparison are MLP (as a baseline), RNN, and 3 types of LSTM.
The models performed as expected, with performance improving proportionally with the complexity of the model's memory.
Attention-LSTM led to an unexpected result, but that was only due to the randomness of the training.
Possible future improvements should focus on trying also with the other target (total UPDRS) and trying a regression with both targets in multioutput.