UNIVERSITAT ROVIRA I VIRGILI

ACTIVITY 1

NEURAL AND EVOLUTIONARY COMPUTATION

Prediction with Supervised Learning Models

BIEL FORADADA JIMÉNEZ

January 28, 2026

UNIVERSITAT
ROVIRA i VIRGILI

Contents

1 Hyperparameter comparison and selection

1.1 Results

The table with the results given multiple paramters is the following one:

Layers	Layer Structure	Epochs	LR	Momentum	Activation	MAPE	MAE	MSE
3	[194, 32, 1]	100	0.0100	0.90	relu	1.6004	0.8887	68.8298
3	[194, 64, 1]	100	0.0010	0.90	sigmoid	0.5330	0.1514	0.0553
3	[194, 128, 1]	150	0.0050	0.80	relu	0.6399	0.1505	0.0509
4	[194, 64, 32, 1]	200	0.0010	0.90	tanh	0.4856	0.1242	0.0401
3	[194, 32, 1]	100	0.0001	0.90	linear	0.4907	0.1342	0.0441
3	[194, 64, 1]	200	0.0001	0.99	relu	0.4870	0.1287	0.0422
3	[194, 32, 1]	150	0.0500	0.10	relu	0.6315	0.1706	0.0631
4	[194, 32, 16, 1]	250	0.0010	0.90	sigmoid	0.5303	0.1508	0.0549
4	[194, 16, 8, 1]	100	0.0100	0.50	tanh	0.4886	0.1303	0.0431
3	[194, 64, 1]	500	0.0005	0.90	relu	0.4352	0.1017	0.0305

Table 1: Hyperparameter comparison results.

1.2 Loss evolution and scatter plots

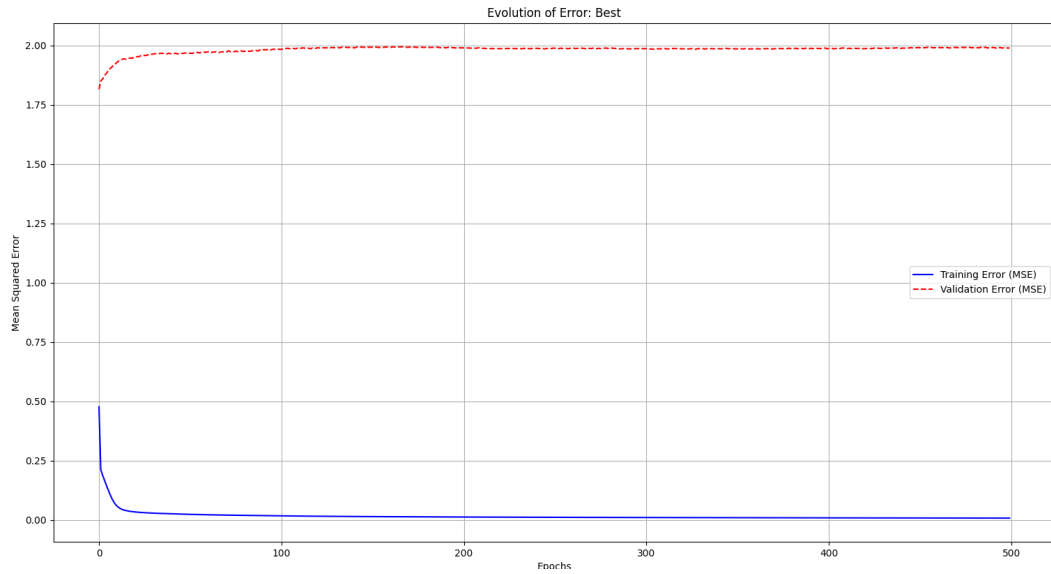The plots with the results are the following ones:

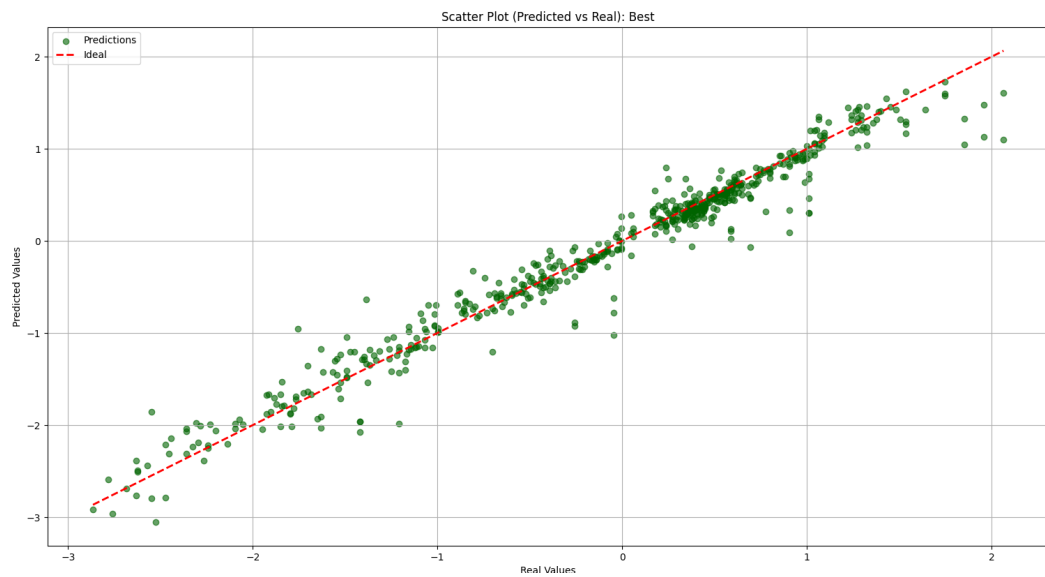Figure 1: Evolution of the loss in the best configuration.

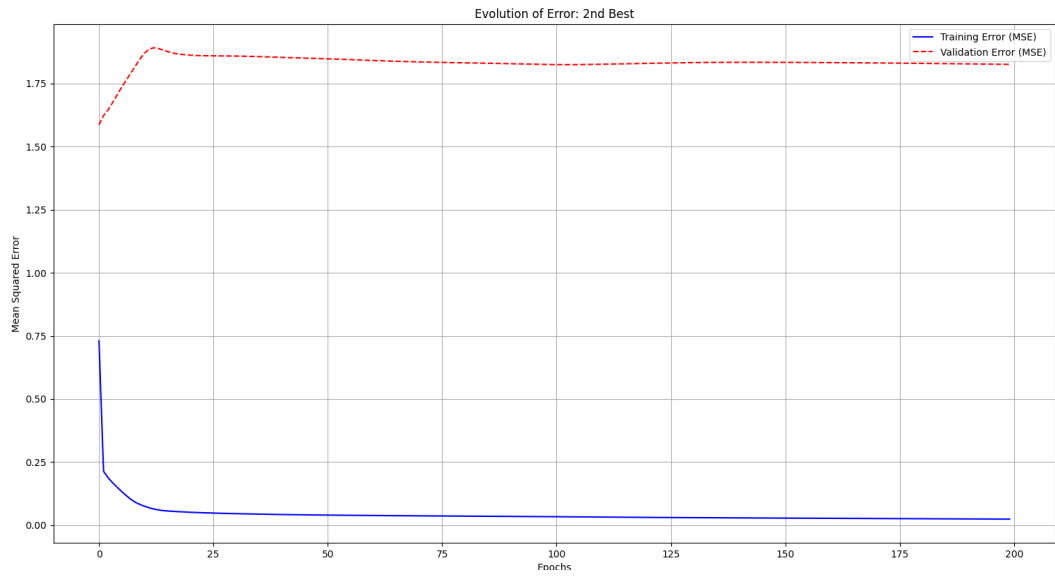Figure 2: Scatter plot of some values with the best configuration.

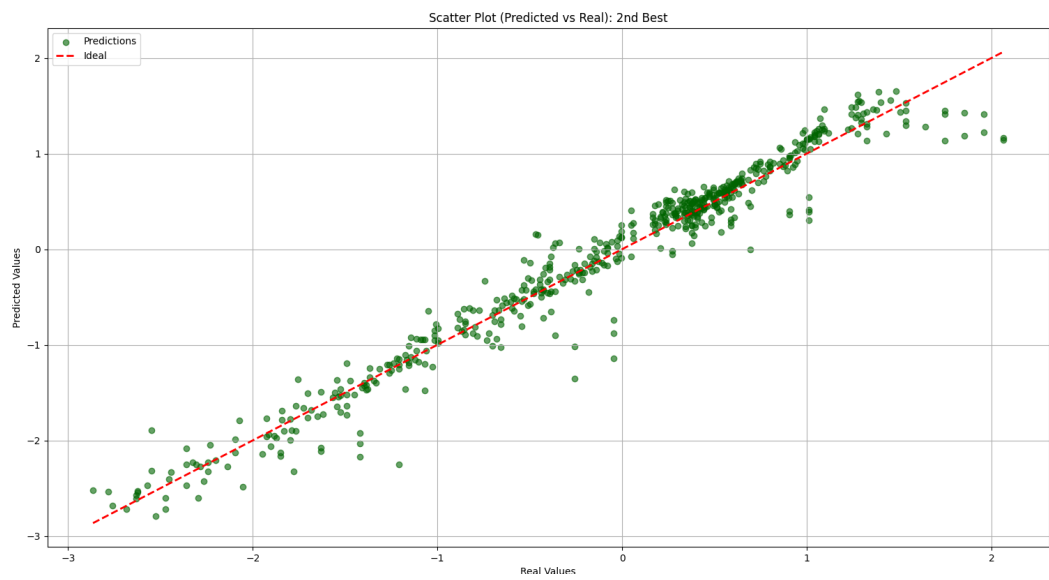Figure 3: Evolution of the loss in the second best configuration.

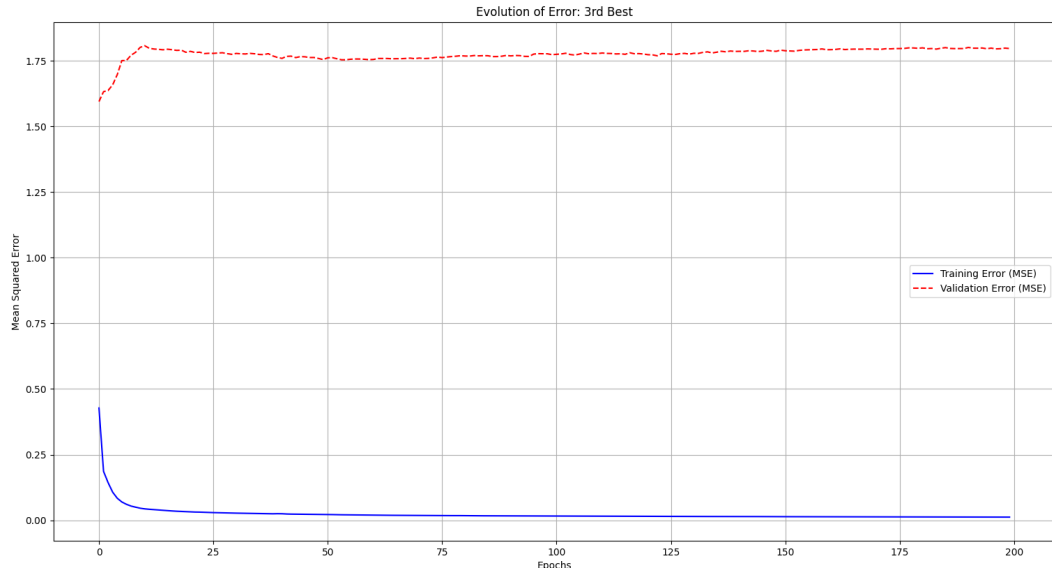Figure 4: Scatter plot of some values with the second best configuration.

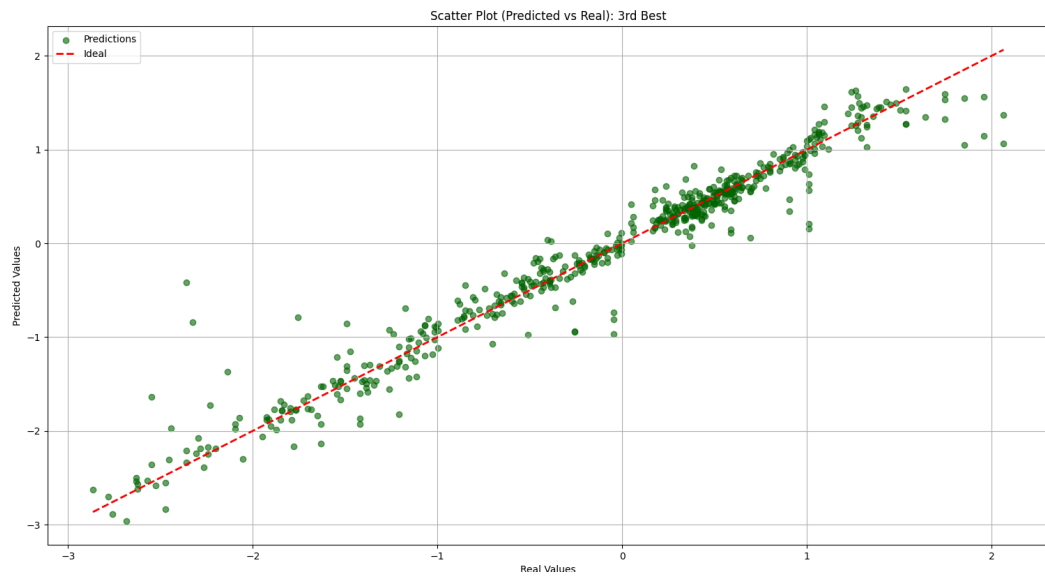Figure 5: Evolution of the loss in the third best configuration.

Figure 6: Scatter plot of some values with the third best configuration.

5

1.3   Explaining of the results

Based on the results obtained in the hyperparameter comparison, the most accurate configurations were those that balanced network shape and a high number of epochs. Configuration 10 (Architecture: [194, 64, 1], ReLU, 500 epochs) was the best model, having the lowest MSE (0.0305). This could be attributed to the higher number of epochs, which allowed the SGD to lower the loss of the life expectancy dataset. By using a smaller learning rate (0.0005), the model avoided the oscillations ensuring a steady decline in the training loss.

The evolution of the loss in the top three models (configurations 10, 4, and 6) reveals a consistent pattern of stabilizing the loss value after a 100 epochs. However, the validation rises a little bit from the initial epochs, this might be a consequence of the model overfitting on the dataset. I did not came to any solution to this problem, I already tried to shuffle the dataset before the training, with the function `numpy.random.permutation()` of the numpy library, but it did not resolve the issue.

2 Part 3.2: Model Result Comparison

2.1 Description of Parameters

- **BP (Scratch) - Config 10**: A 3-layer architecture $[194, 64, 1]$ using the ReLU activation function. It was trained for 500 epochs with a learning rate (α) of 0.0005 and momentum (μ) of 0.9.

- **MLR-F (Scikit-Learn)**: Implemented using the `LinearRegression` class[1], using the default parameters.

- **BP-F (PyTorch)**: A Neural Network based on the `nn.Module` of pytorch, featuring a hidden layer of 64 neurons and ReLU activation, the same as the scratch implementation. It was trained for 500 epochs using the SGD optimizer, with a learning rate 0.001.

2.2 Comparison of Predictions

The table below summarizes the error metrics (MSE, MAE, and MAPE) for the three models.

Model	MSE	MAE	MAPE (%)
BP (Scratch) - Best	0.0305	0.1017	0.4352
BP-F (PyTorch)	0.0401	0.1220	0.4815
MLR-F (Scikit-Learn)	0.0467	0.1337	0.6063

Table 2: Performance comparison between custom and library-based models.
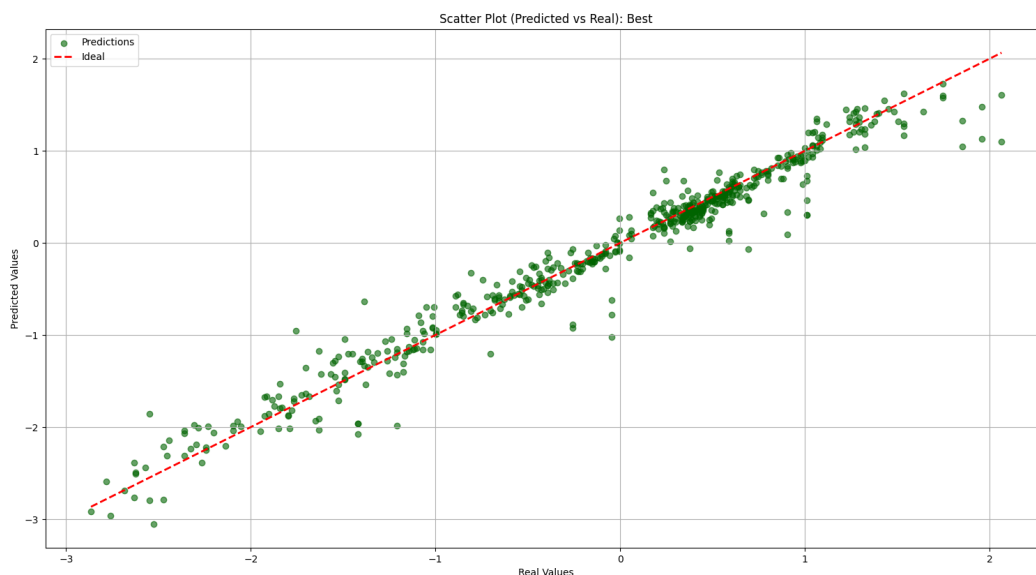
2.3 Scatter Plots: Predicted vs. Real

Figure 7: Scatter plot of Predicted vs. Real values (Scratch Neural Network).

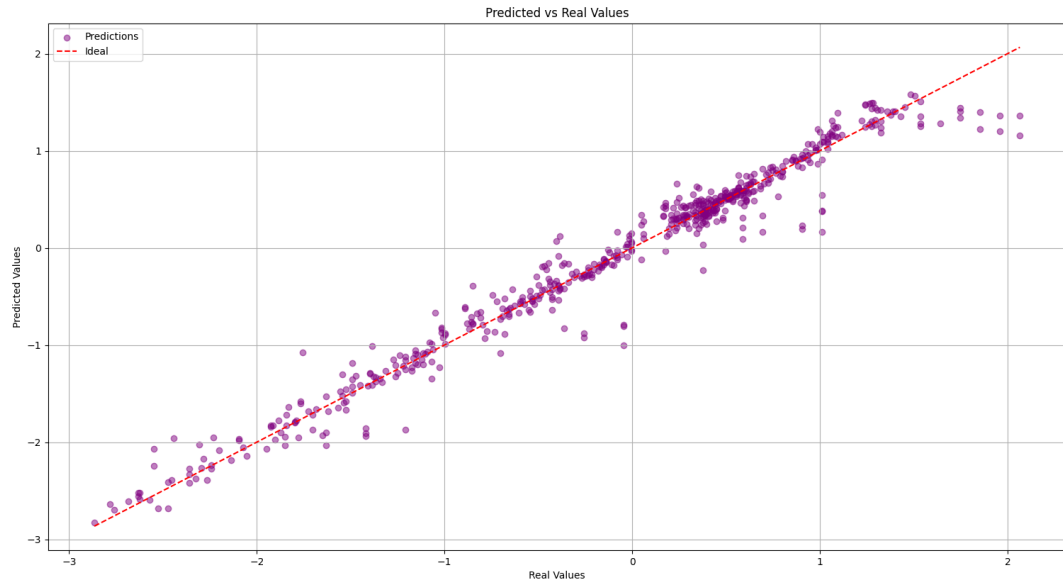[1]https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html

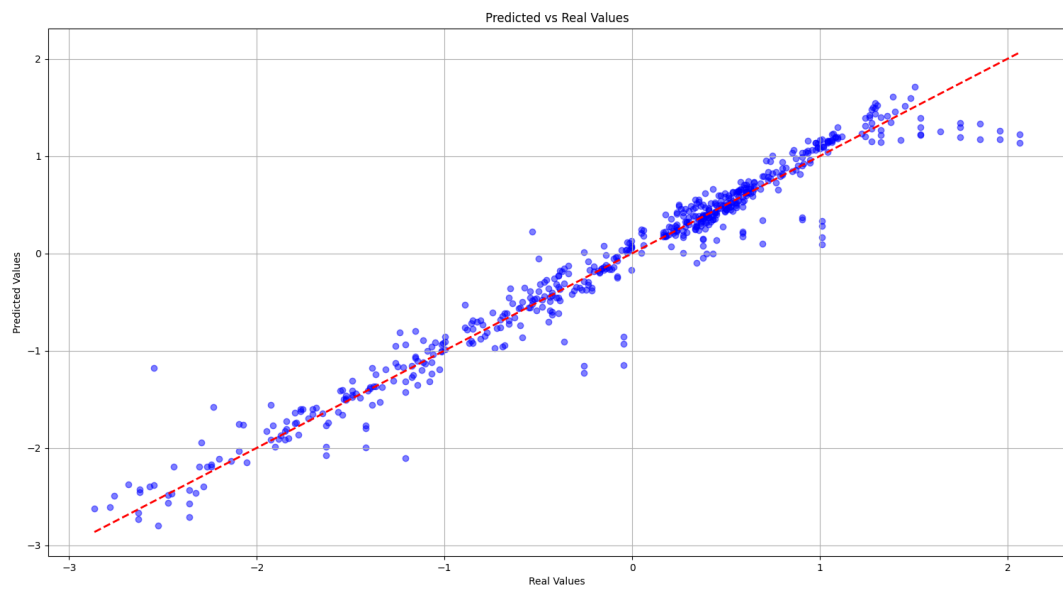Figure 8: Scatter plot of Predicted vs. Real values (PyTorch Implementation).

Figure 9: Scatter plot of Predicted vs. Real values (Scikit Implementation).

2.4   Discussion and Analysis

The results of the test show that the backpropagation implementations from scratch and the PyTorch implementation are better when it comes to predicting the output, with the scratch implementation outperforming the PyTorch implementation with similar hyperparameters in the training.

However, it must be said that the training of the neural network was significantly faster on the PyTorch implementation than the scratch implementation. It also provides more error-proof code, as in the training of my scratch implementation I faced some issues with the loss value overflowing and causing the code to crash for certain learning rate values; on the other hand, this did not happen with PyTorch.

The reason why the backpropagation models outperformed the multi-linear regression model, could be due to the inherent complexity of the dataset which do not provide a linear relationship between the multiple data inputs (194 inputs) and the final values.