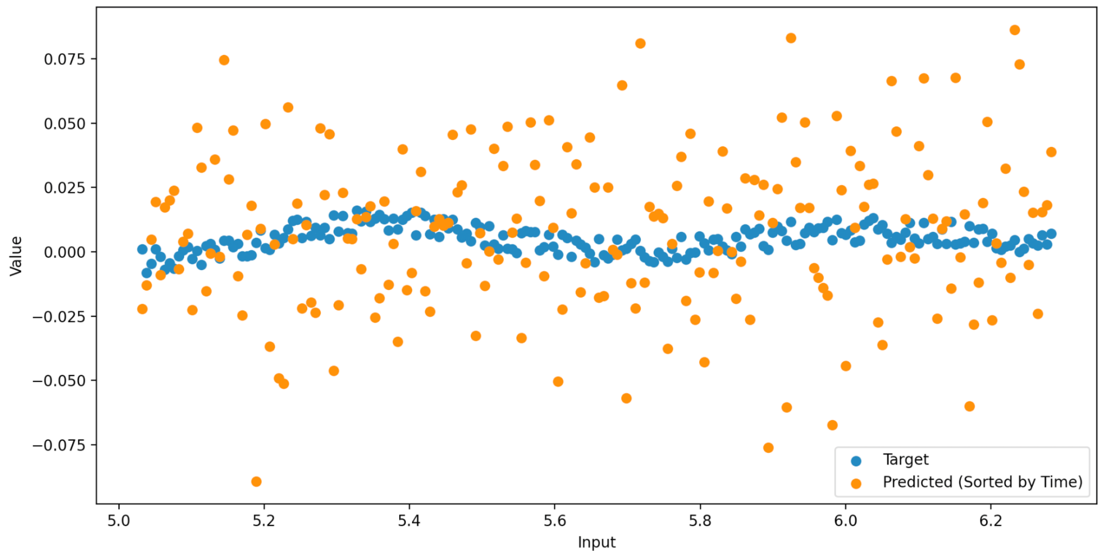
# task2

- Task 1 utilized a PyTorch MLP neural network to approximate measurements from measurements.csv. The model aims to learn the mapping between input and output data. The data depicts a wavering pattern with diminishing magnitude over time, transitioning toward a linear trend.



- Functions and parameters
  - NN :
    - The neural network has three fully connected layers: the first and second layers each have 100 neurons, while the last layer, serving as the output layer, has one neuron. This architecture was selected to capture complex nonlinear relationships between input and output data. With 100 neurons per layer, the model can learn diverse patterns and trends in the data without excessive computational complexity, making it suitable for medium-sized datasets.
    - The LeakyReLU activation function is used because LeakyReLU can handle and improve the problem of dead neurons that may be

        encountered by neural networks during training. In this experiment, LeakyReLU does perform better than ReLU or tanh, with the loss value reduced from 0.03 to 0.02.

        - Dropout layers were added after each of the two hidden layers, with the ratio set to 0.2, in order to reduce model overfitting and enhance the generalisation of the model.Dropout is an effective regularisation tool that significantly reduces the risk of overfitting the model by randomly dropping some of the network connections during training.
        - The MSE is used as a loss function because it is effective in measuring the gap between predicted and actual values in regression tasks. It averages the squared difference between the predicted and actual values to provide a comprehensive assessment. This penalty for larger errors improves the accuracy of the model.
        - RMSprop was selected as the optimizer with a dynamic learning rate scheduler, outperforming Adam and SGD. RMSprop adjusts learning rates for each parameter, making it suitable for non-smooth objectives and preventing convergence to local minima.

- Understanding
    - Model accuracy reflects circuit response to inputs. High accuracy suggests good capture of dynamic characteristics, fitting most data trends.
    - Poor performance in intervals suggests unlearned non-linear or complex dynamics, needing more data or complex models for further exploration.
    - If the model's generalization ability is poor, the circuit's performance might be limited or highly variable in real-world applications, indicating instability and unpredictability under unknown inputs.
- conclusions
    - According to the results of task 1, the overall trend in this experiment is wave , and with the increase of time, the amplitude of wave decreases

gradually and tends to a straight line.

## Task3

- It is possible to improve the performance of a part of the code by implementing it in assembly
    - Calculation of the activation function
        - In the neural network model, LeakyReLU is used as the activation function. Although LeakyReLU is relatively simple（The formula is `f(x) = max(0.01*x, x)`） However, for the processing of large-scale data, optimising its computation directly at the assembly level may still result in weak performance gains. In particular, manually writing vectorised code through assembly language may further reduce CPU cycles when processing each neuron output.
    - . Matrix operations
        - Forward propagation in neural networks involves a large number of matrix multiplications, especially in the fully connected layer （`nn.Linear`）。While these operations have been accelerated by highly optimised mathematical libraries (e.g. Intel MKL, OpenBLAS), under extreme performance demands, it may be considered to manually optimise these underlying operations for a specific piece of hardware （AMD processors: writing assembly code to reduce cache misses and increase computing speed.）using assembly language, especially if the general-purpose optimisations of the framework fail to take full advantage of the characteristics of a specific CPU architecture.
    - Data loading and pre-processing

        - Although this part is usually not a bottleneck in neural network computation, data preprocessing (e.g., normalisation, resizing, etc.) occupies some CPU resources in the data pipeline. Optimisation in assembly language during the data loading phase, especially when complex transformations or real-time data augmentation are involved, may reduce latency and improve overall data processing efficiency.

- however,PyTorch and other modern machine learning frameworks have been highly optimised to take advantage of modern CPU features such as multi-threading, vector instruction sets (e.g. AVX, SSE), etc.
    - Poor portability: Assembly language is highly dependent on the specific processor architecture. This means that assembly code written for one CPU may not be applicable to another, which is unacceptable in multi-platform application development.
    - Maintenance difficulties: Assembly code is difficult to write, debug and maintain. In complex applications, such as deep learning models, using assembly language may significantly increase the difficulty and cost of development and maintenance.
    - Inefficient development: Development in assembly language is far less efficient compared to high-level languages.
- In general, while it's possible to optimize portions of the code using assembly language (as in Task 1), it's generally not recommended. This is because the benefits of using assembly language often don't outweigh the added complexity it introduces.