Assignment 1 Report

1st Bogdan Foca
4419936

2nd Leon Pirazzoli
student number

3rd Lucas van Koppen
7125038

4th Joen van de Vorle
student number

I. Introduction

This project aims to develop a neural network that can predict laser measurement data. The task involves predicting one-step-ahead data points based on provided data. The challenge includes choosing a well-suited neural network model, determining the optimal number of past time steps to use as input, and training the model to accurately predict future values. After training the model, its quality is assessed by letting it recursively generate future predictions and evaluate its performance using the Mean Absolute Error (MAE) and Mean Squared Error (MSE) against a test dataset.

II. Used Models

We took into consideration two different model architectures: a simple feed forward neural network and an RNN.

The first network has an input layer, 3 fully connected hidden layers with size 16 and a final output layer; the activation function used in all layers is ReLU. The size of the input layer is the window of previous data points to be taken into consideration, and it is the main hyperparameter that we focused on. We tried four different values: 8, 16, 32, and 64. The best results were obtained by setting the window size to 32, which had the lowest values for the loss function after 49 epochs.

The second model consists of an RNN with 3 internal layers and a hidden size of 64. It is followed by a fully connected layer with an output of size 1. We decided to use a Recurrent Neural Network (RNN) because of their suitability in handling sequential data. This is perfect for the provided dataset since the featured measurements are sequential in nature. We are using only 1 RNN layer since it keeps the model simple and lightweight, which makes it easier to train the NN and try different hyperparameters. This also makes it less prone to overfitting since it is a small dataset. We are using a hidden size of 64 since it is small enough to combat overfitting, but big enough to learn the patterns in the data. Our RNN features 3 layers in order to be complex enough to capture the underlying pattern in the data. Lastly, we are using an output size of 1 since the model predicts only 1 step ahead.

The second model ended up performing significantly worse than the first, likely due to problems inherent in our choice of architectural parameters. We proceed to report data relative only to the feed forward model.

III. Approach

The data points are scaled before they are used to train the model by dividing them by 300. The neural network is developed in Python using the PyTorch library. We created a custom dataset class that returned a pair of input and label. The input is a vector of *window_size* elements that are fed to the network and the label is the next element in the sequence. To choose the hyperparameters, we performed a grid search on the following set.

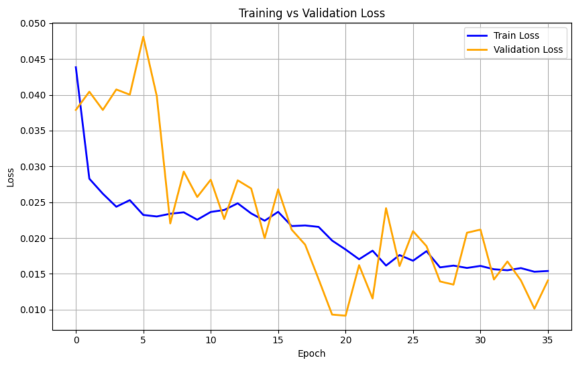


Fig. 1. Loss function with window size = 8

Window Size	8	16	32	64
Optimizer	Adam	SGD	NAdam	
Initial Learning Rate	0.001	0.01	0.1	
Loss Function	MSE	L1	SmoothL1	
Max Epochs	10	20	50	100

Table 1: Table of hyperparamaters

IV. Results

The network performed best with a window size of 16 (fig. 2), NAdam optimiser, an initial learning rate of 0.001, using an L1 loss with 50 max epochs.

Running the model on less epochs resulted in worse results. Because of early stopping, the model would stop training around 40 to 50 epochs, so increasing the maximum further would not produce any changes.

When using a smaller window size, the model predicts more repetitive patterns (fig. 1). While with bigger numbers it seems to learn less effectively, as the loss function values do not decrease as steadily or steeply (fig. **??**. We believe that this is due to the model not being complex enough to process larger numbers of data points at once.

In figure IV we can observe the performance of the best version of the network on the test dataset. The prediction seems to capture the oscillating behaviour of the data points,

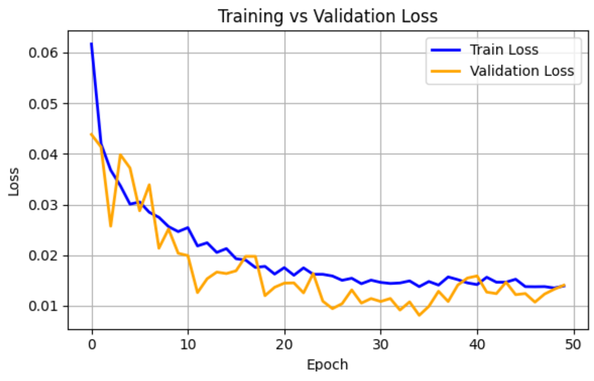


Fig. 2. Loss function with window size = 16

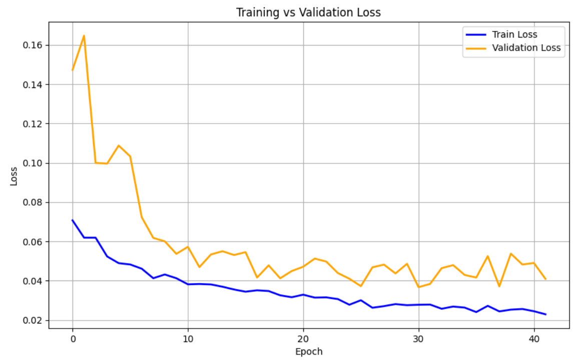
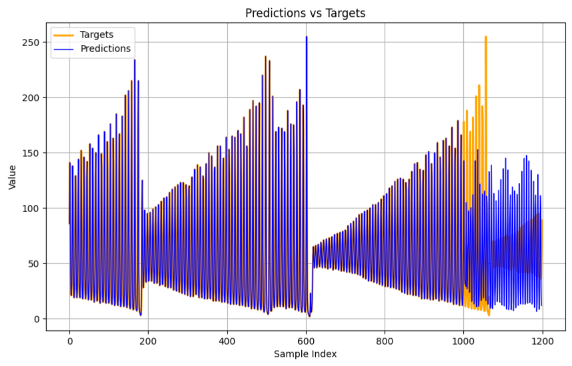


Fig. 3. Loss function with window size = 64

even though it fails to recognize the pattern of suddenly decreasing after reaching high values.

V. Conclusion

During this assignment, through systematic experimentation with different hyperparameters and model configurations, we found that the feed-forward network with the aforementioned parameters consistently outperformed the other variations and our RNN model in terms of prediction accuracy and loss reduction. The optimal setup used a window size of 16, the NAdam optimizer, an initial learning rate of 0.001, and the L1 loss function over 50 epochs.


Predictions vs Targets
Targets
Predictions
Value
Sample Index