Matam Mini Project

Halil Ibrahim Kanpak

June 22, 2024

Abstract

In this project, a model is developed to predict earthquake magnitude from four-second segments of earthquake waves. Various optimization techniques and machine learning models are used to achieve the best results. This report summarizes model training, the optimization process, comparison of models, and results.

1 Introduction

Our goal is to develop a model that can predict the earthquake magnitude from these four-second segments. To achieve this, I have used Python language version 3.12.3 with PyTorch, Pandas, and Sklearn libraries, and also Julia language version 1.10.4 for cross-checking the models and preprocessing the data.

2 Dataset and Preprocessing

The dataset consists of three files:

- `DATA_4sn_trn.txt`: Training data
- `DATA_4sn_val.txt`: Validation data
- `DATA_4sn_tst.txt`: Test data (with magnitudes hidden as zeros)

I have added:

- `DATA_4sn_tst_pred.txt`: Test data filled with model results having the best accuracy

2.1 Loading and Splitting Data

I inspected the data to consider the cleanness and usability of data first and performed the necessary cleaning and conversions for consistent results. The data is loaded and split into features and labels for training and validation purposes. The test data contains features only, with the target magnitudes hidden.

2.2 Preprocess

Considering preprocessing, I chose PCA as a dimension reduction method due to the spikes in data. Despite my lack of knowledge in seismograph sensors and architecture results, I considered the resemblance among the parameters of the data and chose PCA to pick the most relevant parts in the recordings. Using Julia, I noted that 20 principal components explained $> 95\%$ of the variance of the data, which allowed me to compare my results with simpler models and verify the results against raw data results.

3 Model Development

Two neural network models are developed to predict earthquake magnitudes from the given data. Various optimizers are used to improve the models' performance. I chose sigmoid among the well-known activation functions as it provided the best results. Architectures 3 and 4 are the same network architectures used to process the PCA-applied data. I varied different optimizers since considering the tests among other

hyperparameters, the most varying and rich results were given through the choice of different optimizers due to the simplicity of data. I used MSE for calculating the loss among the results.

The neural network architectures used in this project are as follows:

- **Architecture 1:** 200-128-64-32-1
- **Architecture 2:** 200-128-64-1
- **Architecture 3:** 20-128-64-32-1
- **Architecture 4:** 20-128-64-1

After obtaining the PCA results, I trained and tested the neural network models with PCA data, but did not achieve better results, as shown in results in table 1. Nevertheless, developing a multivariate linear regression model both in Julia and Python yielded better results. This was most probably because the DNNs already reduce the dimension, and further reducing the dimension with PCA resulted in worse outcomes. This was not the case with multivariate linear regression, which yielded 1.37× better results for the validation set.

4 Results

The results of training the neural network models with different optimizers are summarized in table 1. Considering the results, I achieved the best outcome with Architecture 1 using the RMSprop optimizer on raw data, yielding a validation loss of 0.1897. Although tests with neural networks on PCA-applied data gave worse results, the linear regression model provided better outcomes as discussed previously.

For further verification, I compared results among the models, which showed grounded results. Applying the MSE loss, I obtained 0.004191 MSE loss among the model results using my linear regression model and Architecture 1 with RMSprop optimizer on raw data.

5 Conclusion

In this project, I developed models to predict earthquake magnitudes from four-second segments of earthquake waves using both neural networks and multivariate linear regression. The best performance was achieved with the RMSprop optimizer on raw data using the neural network architecture 200-128-64-32-1, yielding a validation loss of 0.1897. While PCA did not improve the neural network results, it significantly enhanced the performance of the linear regression model, reducing the validation loss by approximately 1.37 times compared to the raw data model. This highlights the importance of selecting suitable preprocessing techniques for different types of models.

6 Acknowledgments

I have highly used ChatGPT for implementations I have done on Julia and Python and for forming the LaTeX document.

7 Table of Tests

ML Model	Optimizer	Learning Rate	Training Loss	Validation Loss	Preprocess
DNN $200 \times 128 \times 64 \times 32 \times 1$	Adam	0.001	0.2480	0.1955	None
DNN $200 \times 128 \times 64 \times 32 \times 1$	RMSprop	0.001	0.2276	0.1897	None
DNN $200 \times 128 \times 64 \times 32 \times 1$	Adagrad	0.01	0.2499	0.2004	None
DNN $200 \times 128 \times 64 \times 32 \times 1$	Adadelta	1.0	0.2440	0.1988	None
DNN $200 \times 128 \times 64 \times 32 \times 1$	AdamW	0.001	0.2505	0.1943	None
DNN $20 \times 128 \times 64 \times 32 \times 1$	Adam	0.001	0.2562	0.2146	PCA
DNN $20 \times 128 \times 64 \times 32 \times 1$	RMSprop	0.001	0.2846	0.2142	PCA
DNN $20 \times 128 \times 64 \times 32 \times 1$	Adagrad	0.01	0.2677	0.2380	PCA
DNN $20 \times 128 \times 64 \times 32 \times 1$	Adadelta	1.0	0.2816	0.2407	PCA
DNN $20 \times 128 \times 64 \times 32 \times 1$	AdamW	0.001	0.2788	0.2221	PCA
DNN $200 \times 128 \times 64 \times 1$	Adam	0.001	0.2480	0.1955	None
DNN $200 \times 128 \times 64 \times 1$	RMSprop	0.001	0.2276	0.1897	None
DNN $200 \times 128 \times 64 \times 1$	Adagrad	0.01	0.2499	0.2004	None
DNN $200 \times 128 \times 64 \times 1$	Adadelta	1.0	0.2440	0.1988	None
DNN $200 \times 128 \times 64 \times 1$	AdamW	0.001	0.2505	0.1943	None
DNN $20 \times 128 \times 64 \times 1$	Adam	0.001	0.2513	0.2017	PCA
DNN $20 \times 128 \times 64 \times 1$	RMSprop	0.001	0.2760	0.2120	PCA
DNN $20 \times 128 \times 64 \times 1$	Adagrad	0.01	0.2695	0.2380	PCA
DNN $20 \times 128 \times 64 \times 1$	Adadelta	1.0	0.2463	0.2058	PCA
DNN $20 \times 128 \times 64 \times 1$	AdamW	0.001	0.2544	0.2076	PCA
Linear Regression	MLE	_	0.2681	0.3568	None
Linear Regression	MLE_PCA	_	0.2771	0.2591	PCA

Table 1: Comparison of different ML models, optimizers, and preprocessing techniques. Note that I have trained all of the Dense Neural Networks "DNN"s with 50 epochs.