Earthquake Prediction Using Sequence Models

Matthew Avallone
Department of Computer Science
New York University
mva271@nyu.edu

Parul Raj
Department of Computer Engineering
New York University
pr1498@nyu.edu

Victor Zheng
Department of Computer Science
New York University
vz365@nyu.edu

Abstract

Earthquakes cause havoc in different regions and are difficult to predict. Although many advances have been made in machine learning and deep learning across various fields, very few of these techniques have been targeted towards geophysical problems. This project aims specifically to predict the time until the next laboratory earthquake based on seismic data observed in a lab environment. Our project is part of the Los Alamos National Laboratory (LANL) Earthquake Prediction competition on Kaggle [6].

32nd Conference on Neural Information Processing Systems (NeurIPS 2018), Montréal, Canada.

1 Introduction

The aim of our project is to predict earthquake occurrences with high accuracy based on seismic data that is emitted by laboratory-generated faults. The data in this project has been collected by LANL researchers from a well-defined laboratory earthquake experiment, and is provided in the form of acoustic_data (chunks of seismic signal) as input and time_to_failure as output. This lab-generated data mimics the faults generated in earth, which ultimately lead to earthquakes. Our objective is to find hidden patterns within the data which are indicative of when an earthquake will happen. The approach used in this project is a stacked generalization ensemble model, which takes the outputs of pretrained sequence models as input and attempts to learn how to best combine the input predictions to improve the final prediction [2].

2 Method

2.1 Input Data

The seismic data was collected using a piezoceramic sensor, which outputs a voltage upon deformation by incoming seismic waves. These voltage readings are stored in floating point format. The training data (9.6GB) is a continuous signal of approx. 600 million points, while the test data (8.6MB) is given in chunks of 150,000 points. In order to achieve the same data format for training and test sets, we divide the input data into segments of size 150,000 (150x1000) to train the models. The input is 0.0375 seconds of seismic data (ordered in time), which is recorded at 4MHz, hence 150,000 data points. The output is the time remaining at each point until the following lab earthquake, recorded in seconds [11]. The training data includes sixteen earthquakes, unevenly distributed across the data set. The size of the data is feasible to work with since our notebooks take around an hour to run on a GPU.

2.2 Feature extraction

For each of the input segments used in training, we are extracting some features used in rare time series prediction problems since earthquake events can be considered rare and unpredictable [12]. We are also referencing two Kaggle kernels for feature selection and implementation, "Seismic Data EDA and Baseline" [8] and "Shaking Earth" [1] which is based on *Machine Learning Predicts Laboratory Earthquakes* [12] and *Estimating Fault Friction From Seismic Signals in the Laboratory* [13].

We use 32 features for 4 of the models in the stacked ensemble. These features are mean, standard deviation, minimum value and maximum value of a input segment. For the fifth model we do Fast Fourier Transform (FFT) of the input and then extract 104 features. These features are mean, standard deviation, minimum value and maximum value of the transformed data along with the same set of features extracted from the original data. We first sub-sample each input chunk into eight signals by selecting every 5, 10, 20, 25, 40, 50, 100 and all points from the original signal. From there, the features are calculated from the remaining data points and concatenated together, creating (150x32) and (150x104) format input to train the sub-models. By sub-sampling the data multiple times, we are filtering out possible noise injected during the experiments. It improves prediction performance compared to just using the original signal.

2.3 Evaluation criteria

The test set consists of segments of data, with each segment represented by a segment ID. The time to failure gives the time between the last row of the segment and the actual time that the next lab earthquake occurred. The competition score is evaluated based on mean absolute error between the predicted time to failure and the actual time to failure. The top score is 1.282 (as of May 12) and our current best model achieved a score of 1.486. It originally ranked among the top 26% of all submissions, but currently ranks within the top 34%. Our first target was to reduce mean absolute error to get a rank among top 30%. Since we were able to accomplish this goal before our milestone, we aimed at ranking among the top 10% by the final project submission. This equates to a score of at least 1.425 (as of May 12).

2.4 Previous attempts

There were many previous iterations of our approach before settling on a stacked ensemble model. We will provide a description of some past submissions to the competition and the insight behind them. They follow a chronological order.

2.4.1 GRU based RNN

For the first approach, we implemented a Recurrent Neural Network (RNN) inspired by the kernel "RNN Starter For Huge Time Series" [9], which uses a Gated Recurrent Unit (GRU) layer to achieve a score of 1.516. The model has an architecture as shown in Figure 1.

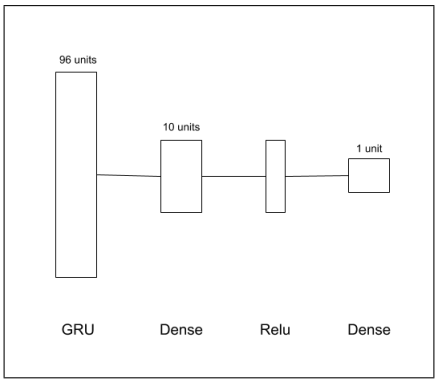
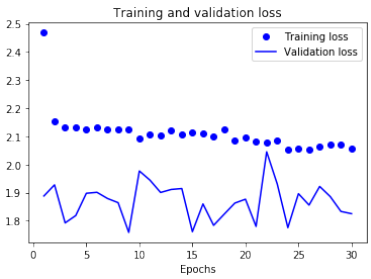


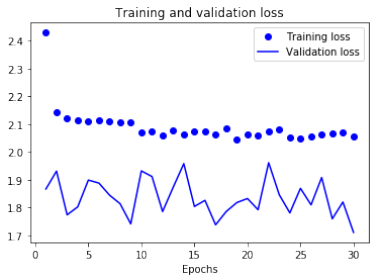
Figure 1: GRU based RNN model

After our team increased the amount of features extracted, the model achieved a score of 1.486, which is an improvement of 0.03 or about 2%. Figure 2 shows a comparison of training and validation loss for different number of features. The optimizer chosen is Adam and the loss function is mean absolute error. For training, a batch size of 32 was used and ran for 30 epochs.

We also tried Fast Fourier transform for feature extraction. This was done to extract frequency based features from the input data, and the model achieved a score of 1.597.



(a) 12 features



(b) 32 features

Figure 2: Loss graphs for original and modified RNN starter

2.4.2 Autoencoder paired with inference model

Another approach we tried was performing feature extraction using an autoencoder. The inspiration behind this idea came from a paper published by Uber in 2017 on extreme event forecasting [5]. Uber was interested in trying to predict the number of ride requests on certain holidays (e.g. New Year's Eve) when there was a high variance in the data. This type of situation is similar to the LANL Earthquake competition since here we are focusing on a small handful of rare events in a large data set. The paper describes an approach that uses an LSTM encoder-decoder network to learn the features rather than manually computing them. This allows for identifying more complex and unique features

that a human may not be able to see. Uber then used a second network (LSTM-RNN) to perform the predictions based on the learned features.

The architecture for the models are shown in Figure 3. Two separate models were used: an autoencoder to aid in feature extraction, and a GRU based RNN similar to our first approach to make inferences.

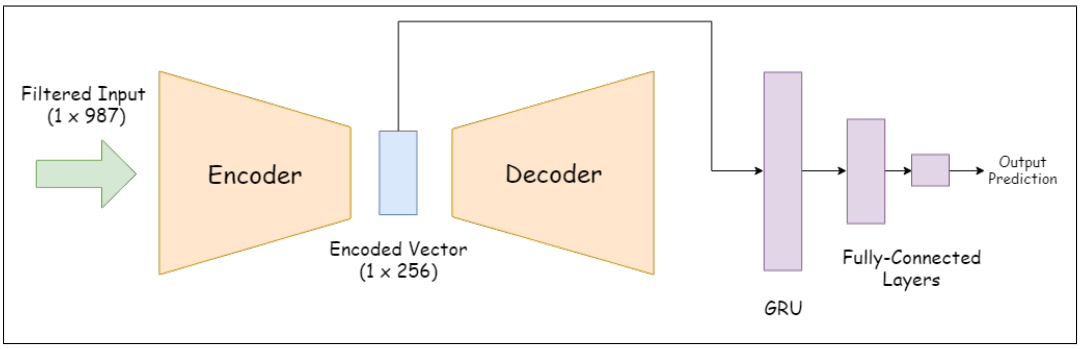
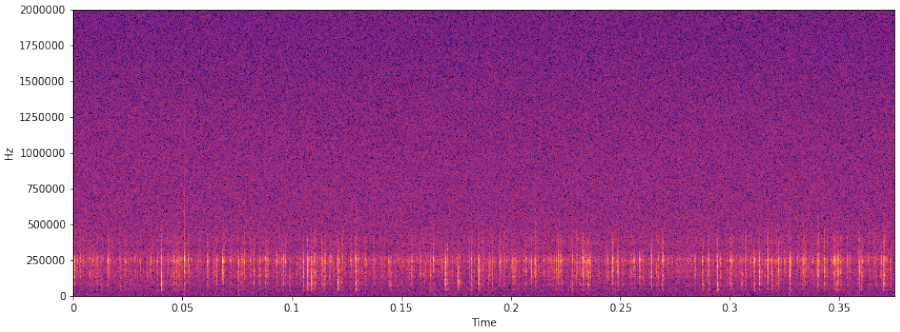
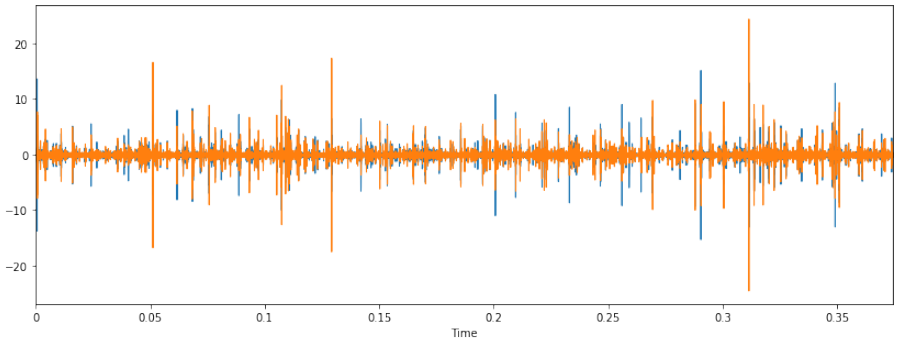


Figure 3: Feature Extractor and Inference Model

The input segments in this approach were augmented before training the two models. A band pass filter was implemented based on the kernel "LANL Finding L-estimators via PCA" by Michael Ganger [4]. Ganger describes the problem as a signal processing problem, identifying which frequencies we are interested in from the data. By analyzing the spectrum of the signal, he determined anything over 300kHz can be seen as noise. He isolated a small band from 240 kHz to 260 kHz using a linear-phase FIR filter, de-modulated the signal to baseband, and decimated it by a factor of 90 (45 kHz).



(a) Noisy signal

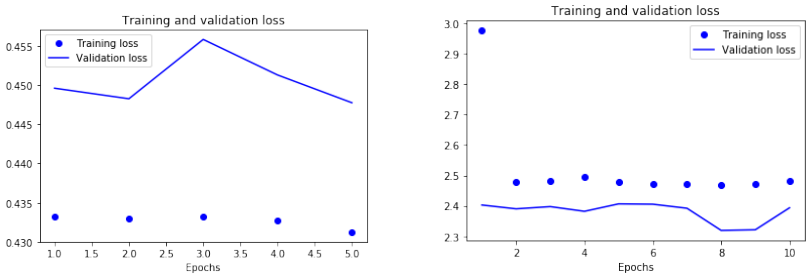


(b) Filtered signal

Figure 4: Input before and after signal processing

This filtering was done on the input segments (the chunks of 150,000 points), reducing them to only 987 points afterwards. Originally, filtering was not performed and the approach scored 2.357 on the

leader board (much worse than before). After implementing the band pass filter, the score jumped to 1.962. However, this score is still inferior to the approach described in section 2.4.1 .



(a) Autoencoder (b) Inference model

Figure 5: Training and validation loss

2.5 Current approach

The current implementation is a stacked ensemble model, which uses the predictions from multiple sub-models to generate a combined optimal prediction. Stacked generalization works by deducing the biases of the generalizer(s) with respect to a provided learning set. This deduction proceeds by generalizing in a second space whose inputs are the guesses of the original generalizers when taught with part of the learning set and trying to guess the rest of it, and whose output is the correct guess [14]. Essentially, the sub-models act as the first layer to the stacked model, and are pretrained on a subset of the training data. The other portion of the training data (i.e. the validation set) is used to train the stacked model.

The 5 sub-models in our approach are a GRU based RNN, a stacked GRU based RNN, an LSTM based RNN, GRU with FFT for feature extraction and Convolutional RNN. Each is pretrained on about 80% the training set (split along the 2nd earthquake occurrence). Afterwards, these networks become layer 0 of a stacked model, which has a new model in layer 1 composed of two fully-connected layers. The new model, acting like a "meta-learner", determines the amount of weight each prediction should receive and combines them to create the total. It emphasizes different aspects of each model to learn the optimal prediction of earthquake occurrences.

The sub-models used in the ensemble model are described in the subsections that follow.

2.5.1 LSTM model

A Long Short-Term Memory (LSTM) unit is a type of memory cell seen as an improvement on the original RNN. It performs better on longer sequences of input data than a standard RNN since it has a memory component to it.

An LSTM unit has multiple components and equations associated with it. Below is a description of them. W and U represent weight matrices, while x represents the input.

$$Forget\ Gate\ (f_t) = \sigma(W_f h_{t-1} + U_f x_t)$$
$$Input\ Gate\ (i_t) = \sigma(W_i h_{t-1} + U_i x_t)$$
$$Output\ Gate\ (o_t) = \sigma(W_o h_{t-1} + U_o x_t)$$
$$Candidate\ Memory\ (\widetilde{c}_t) = \sigma(W_f h_{t-1} + U_f x_t)$$
$$Memory\ Cell\ (c_t) = f_t c_{t-1} + i_t \widetilde{c}_t$$
$$Output\ (h_t) = o_t \phi(c_t)$$

Memory is updated by forgetting some existing memory and adding some new memory. The amount of existing memory that gets forgotten is controlled by the forget gate. The input gate controls whether to keep existing memory or overwrite it with candidate memory.

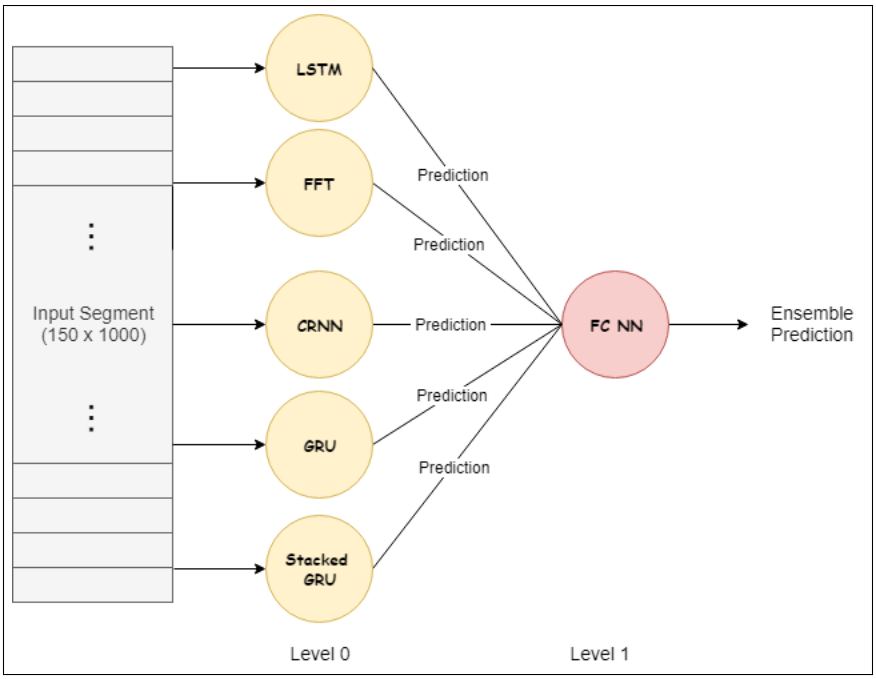


Figure 6: Stacked ensemble model

The LSTM model implemented for the stacked ensemble model is shown in figure 7. It used 48 units in the LSTM layer, followed by a 50% dropout layer and by two fully-connected layers. Its individual submission score was 1.543.

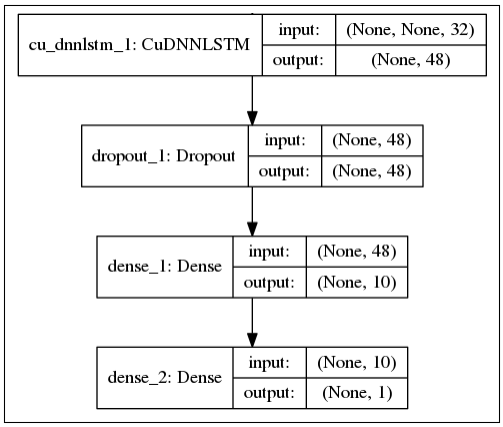


Figure 7: LSTM model

2.5.2 GRU model

A GRU is another type of memory cell using in sequence models. It behaves similarly to the LSTM but has a slightly different structure to it.

The GRU unit has multiple components and equations associated with it. Below is a description of them. W and U represent weight matrices, while x represents the input.

$$Reset\ Gate\ (r_t) = \sigma(W_r h_{t-1} + U_r x_t)$$
$$Update\ Gate\ (z_t) = \sigma(W_z h_{t-1} + U_z x_t)$$
$$Candidate\ Memory\ (\widetilde{h}_t) = \phi(W(r_t * h_{t-1}) + U x_t)$$

$$Current\ State\ (h_t) = z_t h_{t-1} + (1 - z_t)\widetilde{h}_t$$

The reset gate influences how much of the previous state goes into the candidate memory. When it is zero, the GRU forgets the previous state entirely and reads the content, x, instead. The update gate dictates how much of the previous state contributes to the current state. When the reset gate equals 1 and the update gate equals 0, the GRU reduces to an RNN.

The GRU model implemented for the stacked ensemble model is shown in figure 8. It used 48 units in the GRU layer, followed by a 50% dropout layer and two fully-connected layers. Its individual submission score was 1.486.

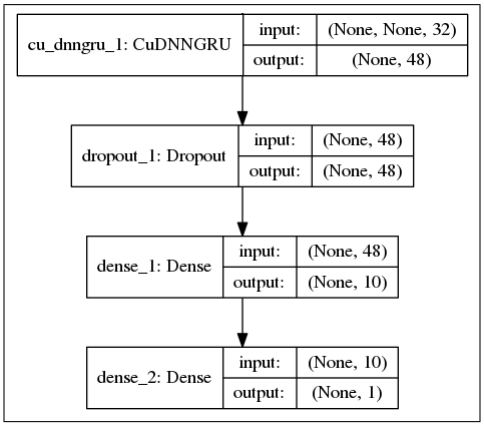


Figure 8: GRU model

2.5.3 Stacked GRU

A stacked GRU model was implemented by adding a second GRU layer after the first one and including 20% Dropout layers after each of them. It closely resembles the GRU model in Figure 8. The model achieved a score of 1.535.

Some hyperparamter tuning was attempted on this model. We tried switching from Adam to Nadam and doubling the learning rate from 0.0005 to 0.001, but saw no improvement in our score. We also tried decreasing the number of units in the GRUs from 96 to 48 while having one bidirectional GRU layer, but this did not improve our score as well. In the end, we did not keep these changes.

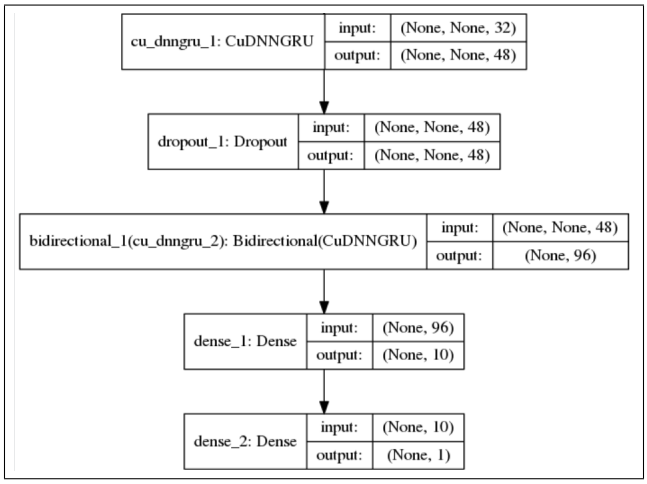


Figure 9: Stacked GRU model

2.6 Convolutional RNN

A CRNN model combines the advantage of CNN with RNN. CNNs are good for spatial structure information while RNNs are good for learning contextual dependencies [15]. In our model CNN layers are used to generated middle level features and then RNN layers are used to learn the dependencies between these features.

We used CRNN as one of the base models in our ensemble. It was implemented by connecting two convolution layers with LSTM and followed by two dense layers. The model achieved a score of 1.589.

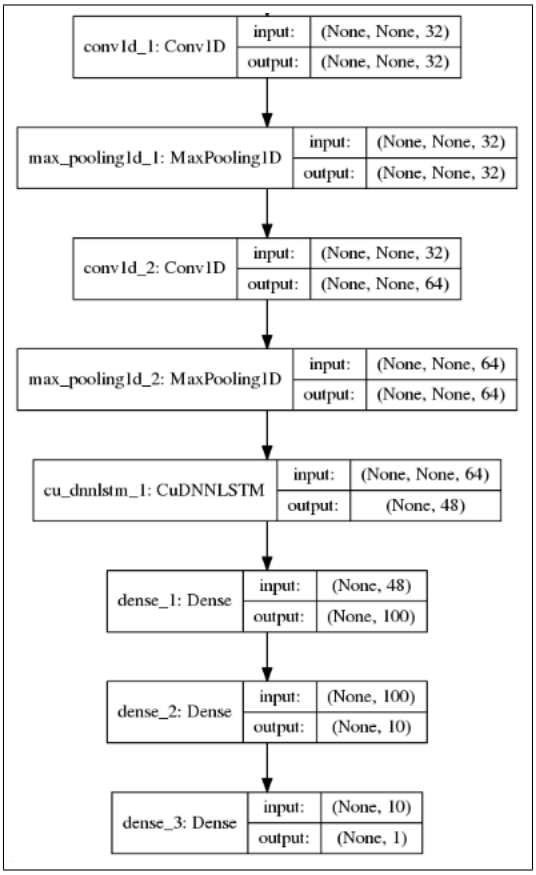


Figure 10: CRNN Model

3 Results

Our best submission is the stacked ensemble model that scored 1.476. It uses LSTM, GRU, GRU with FFT for feature extraction, stacked GRU and Convolutional RNN, as layer 0 models. It ranks 1181 out of 3767 submissions, which is among the top 30%. We were able to reach our first goal of ranking within the top 30% by our milestone report. Unfortunately, we were unable to achieve our second goal of finishing within the top 10% by our final submission.

Our second best submission was our first approach, which was a RNN with one GRU layer followed by 2 dense layers and trained on 32 features extracted from the data set. It achieved a score of 1.486.

A summary of results for various models is presented in Table 1. Comparison of training loss of the ensemble model along with its sub-models is in Figure 11. Comparison of validation loss of the ensemble model along with its sub-models is in Figure 12.

For the purpose of research, we are making our code publicly available [10] .

Table 1: Observations on different models

Model	MAE
Stacked Ensemble Model	1.476
GRU	1.486
Stacked GRU	1.535
LSTM	1.545
CRNN	1.589
GRU with FFT	1.597
GP	1.481
Autoencoder-Inference Model with filtering	1.962

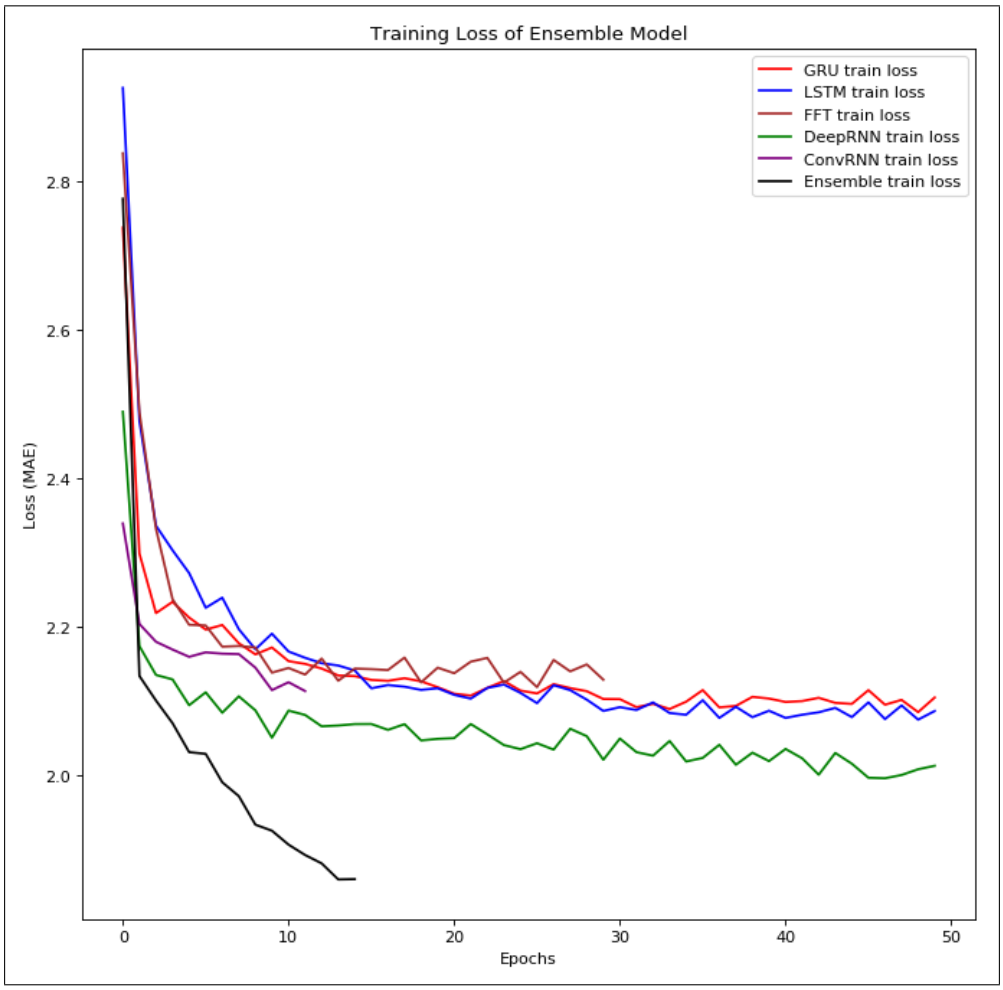


Figure 11: Training Loss Comparison

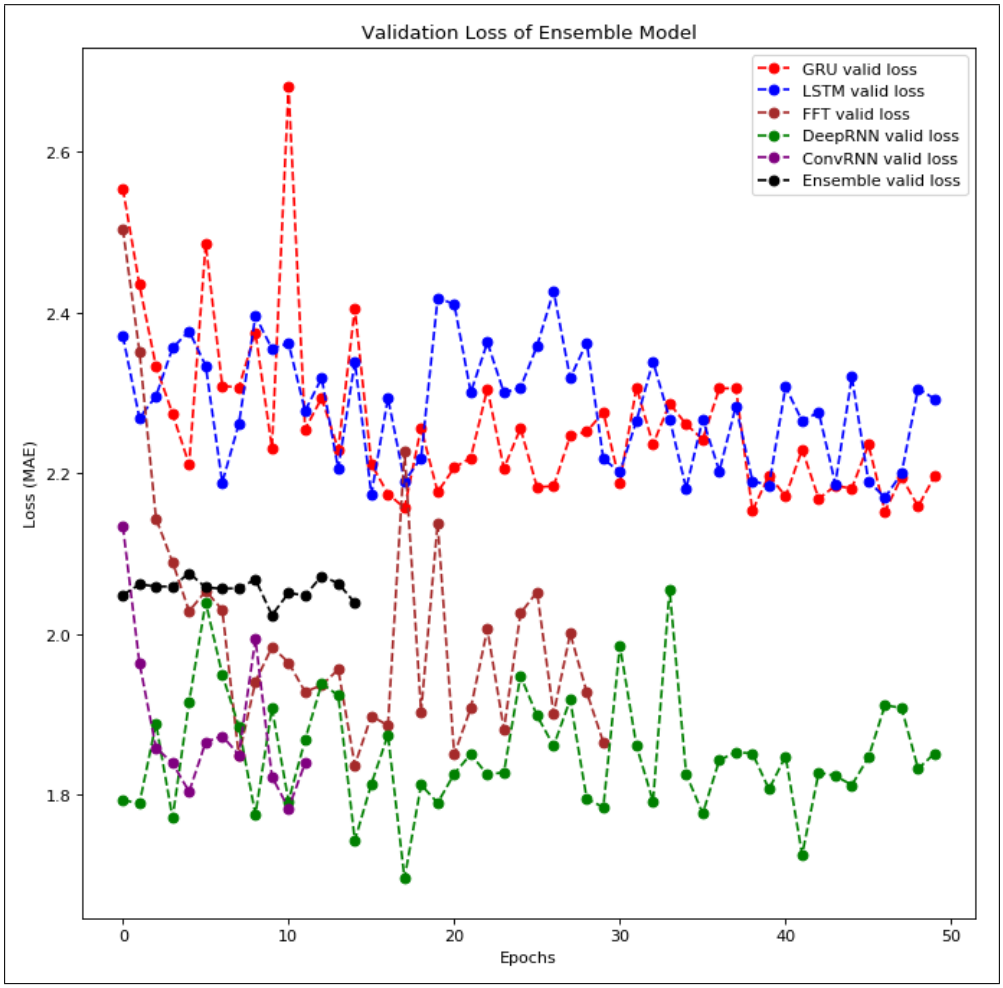


Figure 12: Validation Loss Comparison

4 Related Works

4.1 Random Forest

The original paper for the LANL earthquake project used Random Forest model [12]. This is an ensemble technique which requires creating a collection of decision trees to perform prediction. For regression problems, inference is done by taking the mean of the outputs from each model. Decision trees reduce error by reducing variance. In order to keep the bias low, the trees must be large.

The best public kernel using Random Forest achieved a score of 1.512 [3].

4.2 Gradient Boosting

Gradient Boosting (GB) is a similar method to Random Forest, in which a collection of regression trees are used in an ensemble to boost the overall prediction. The regression trees are considered "weak learners", which means they are shallow decision trees. Boosting is done to reduce bias, which in turn reduces error.

This technique was a popular choice for the competition. The best public kernel achieved a score of 1.442 [7].

Bibliography

[1] Allunia. Shaking Earth | Kaggle. https://www.kaggle.com/allunia/shaking-earth.

[2] Jason Brownlee. How to Develop a Stacking Ensemble for Deep Learning Neural Networks in Python With Keras. https://machinelearningmastery.com/stacking-ensemble-for-deep-learning-neural-networks.

[3] AC datascientist. Earthquake Forecast with Random Forest | Kaggle. https://www.kaggle.com/azc2019/earthquake-forecast-with-random-forests.

[4] Michael Ganger. LANL Earthquake Prediction — Finding L-estimators via PCA. https://www.kaggle.com/mrganger/lanl-finding-l-estimators-via-pca.

[5] Nikolay Laptev, Jason Yosinski, Erran Li Li, and Slawek Smyl. Time-series Extreme Event Forecasting with Neural Networks at Uber. *Int. Conf. Mach. Learn.*, (34):1–5, 2017.

[6] Los Alamos National Laboratory. LANL Earthquake Prediction | Kaggle. https://www.kaggle.com/c/LANL-Earthquake-Prediction.

[7] Andrew Lukyanenko. Feature Selection, Model Interpretation and More | Kaggle. https://www.kaggle.com/artgor/feature-selection-model-interpretation-and-more.

[8] Andrew Lukyanenko. Seismic data EDA and baseline | Kaggle. https://www.kaggle.com/artgor/seismic-data-eda-and-baseline.

[9] Michael Mayer. RNN starter for huge time series | Kaggle. https://www.kaggle.com/mayer79/rnn-starter-for-huge-time-series.

[10] Parul Raj, Matthew Avallone, and Victor Zheng. Deep-Learning-Project. https://github.com/pr1498/Deep-Learning-Project, May 2019.

[11] Bertrand RL. Additional info | Kaggle. https://www.kaggle.com/c/LANL-Earthquake-Prediction/discussion/77526.

[12] Bertrand Rouet-Leduc, Claudia Hulbert, Nicholas Lubbers, Kipton Barros, Colin Humphreys, and Paul A. Johnson. Machine Learning Predicts Laboratory Earthquakes. Feb 2017.

[13] Bertrand Rouet-Leduc, Claudia Hulbert, David C. Bolton, Christopher X. Ren, Jacques Riviere, Chris Marone, Robert A. Guyer, and Paul A. Johnson. Estimating Fault Friction From Seismic Signals in the Laboratory. *Geophys. Res. Lett.*, 45(3):1321–1329, Feb 2018.

[14] David H. Wolpert. Stacked generalization. *Neural Networks*, 5(2):241–259, Jan 1992.

[15] Zhen Zuo, Bing Shuai, Gang Wang, Xiao Liu, Xingxing Wang, Bing Wang, and Yushi Chen. Convolutional recurrent neural networks: Learning spatial dependencies for image representation. In *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, pages 18–26, 2015.