Predicting Laboratory Earthquakes using Deep Learning

Kyra Wang 1002176 | Joel Huang 1002530 | Yap Wei Lok 1002394

Abstract

The ability to forecast earthquakes by creating prediction models based off laboratory earthquake experiment data is useful in learning more about earthquake precursors. In this project, we built models to predict the time remaining before laboratory earthquakes occur using seismic data. We found that simple temporal convolutional neural nets are capable of achieving competitive results with no manual feature extraction. By further applying state-of-the-art techniques in convolutional neural net training, we were able to produce results comparable to non-deep learning techniques such as LightGBM and XGBoost. We present our code at `https://github.com/kyrawny/LANL_PWaves`*. The GUI code can be found at* `https://github.com/ericywl/lanl-gui`*.*

1. Introduction

Forecasting earthquakes is one of the most important problems in the Earth Sciences because of the devastating consequences of earthquakes. Current scientific studies related to earthquake forecasting focus on three key points: when the event will occur, where it will occur, and how large it will be. A laboratory earthquake is an experiment where similar seismic signals are recorded in the time leading up to a slip. In this project, we will be building models to predict the time remaining before laboratory earthquakes occur (`time_to_failure`), given a segment of seismic data (`acoustic_data`).

In the laboratory, a rock in a double direct shear geometry is subjected to bi-axial loading. Two fault gouge layers are sheared simultaneously while subjected to a constant normal load and a prescribed shear velocity. After some time, the layers split. They fail in repetitive cycles of stick and slip that is meant to mimic the cycle of loading and failure on tectonic faults. The experiment is obviously simpler than a real tectonic fault, but shares many similar characteristics.

2. Exploratory Data Analysis

The dataset, which contains the train-validation and test data, can be obtained from Kaggle. For more information, please see `https://www.kaggle.com/c/LANL-Earthquake-Prediction/data`.

The signals recorded from this experiment are a time series with 16 failures. Fig. 1 shows the training data, sampled at a frequency of 0.01. The training data is a continuous time series of the 16 failures, with 629M data points, whereas the test set consists of segments of 150k points which may or may not overlap with a failure event. The labels for the training points are the times-to-failure (TTF), in seconds, till the next laboratory earthquake.

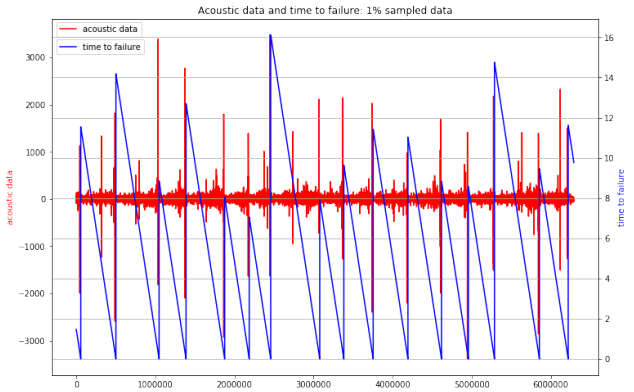

Figure 1. The training dataset containing acoustic signals (in red), and the time-to-failure for each point (in blue), sampled at a frequency of 0.01.

3. Methods

The test set consists of batches of 150k points. For each segment in the test data, we need to predict a single time to failure corresponding to the time between the last row of the segment and the next laboratory earthquake. We thus split the continuous sequence of training data into segments of 150k points each, which produced around 4100 segments. Each segment was then labeled with the final time to failure value, which makes the training set equivalent in format to the test set. We then further split the training data into an 80/20 training/validation split.

As we are predicting a single value, all the models we trained utilized a fully connected layer at end to regress all the outputs into a single prediction for the TTF:

$$y = Wx + b$$

In order to use this training data as input, we considered two approaches:

1. Using the entire 150k points as input; or
2. Using statistical features as input.

We describe our methodology for extracting statistical features from the dataset in section 3.2.

3.1. Evaluation Metric

The regression performance for this Kaggle competition is measured using mean absolute error, or L1 loss, which is the average error over the entire dataset:

$$\text{MAE} = \frac{1}{n}\sum_{n} |\, y_n - \hat{y}_n \,|$$

We use this loss function to evaluate and train our models. The mean absolute error can also be interpreted as the average number of seconds a prediction is away from the true TTF.

3.2. Time Series Feature Extraction

The authors of [10] recently identified a class of signal (see Fig. 2(a)) with high predictive capability, previously thought to be noise, early in the fault cycle. They claim that variance, kurtosis, and threshold over windows in the distant failure signal are important features for prediction, and achieve a high R^2 score using a random forest regressor.

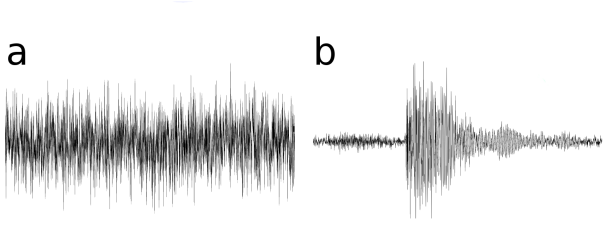

Figure 2. (a) Distant failure feature, (b) Imminent failure feature. Image from [10]

We compute the following features over three window sizes 100, 500, and 1000: mean, standard deviation, kurtosis, maximum and threshold. The features are computed as shown below, over a window w of n points . In particular, we calculate the threshold as how much the mean over the window exceeds some positive threshold value T:

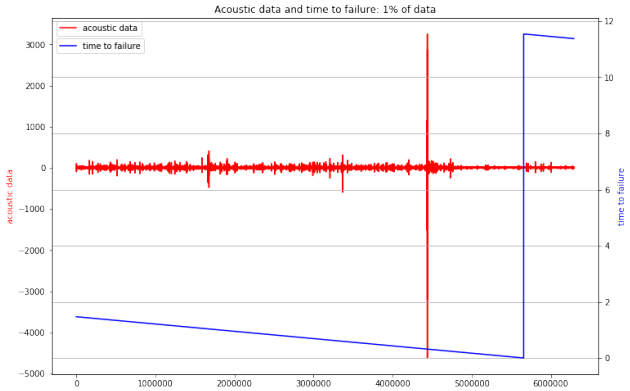

Figure 3. Zoom on the signal and TTF before a failure event.

$$\mu_w = \frac{\sum_n x_n^{(w)}}{n}$$

$$\sigma_w = \sqrt{\frac{\sum_n (\, x_n^{(w)} - \mu_w \,)^2}{n-1}}$$

$$\gamma_{2,w} = \frac{\sum_n (x_n^{(w)} - \mu_w)^4 / n}{{\sigma_w}^4} - 3$$

$$\max(w) = \max\{x_1^{(w)}, \ldots, x_n^{(w)}\}$$

$$\text{thresh}(w) = (\mu_w - T) \cdot [\,(\mu_w - T) > 0\,]$$

The features are then normalized according to the mean of the feature value over the window. This basic feature set replaces the use of raw data input to sequential models like RNNs, and has two advantages. Firstly, the sequence length is reduced to the length of the segment divided by the window size, making RNNs usable without vanishing gradients over extremely long sequences. Secondly, the dimensionality of each element in the sequence is increased.

Multiprocessing for Feature Engineering We implement Python multiprocessing to compute features over window sizes within the 150k frame in parallel using 16 CPU cores. Each process takes different windows and computes statistical features like the mean, variance, and kurtosis over the window. Since we experiment with different window sizes, multiprocessing with Pandas dataframes is highly necessary.

Window size	Sequential	Parallel
100	9h 26m	26m 07s
500	1h 35m	4m 38s
1000	48m 02s	2m 04s

Table 1. Comparison of feature extraction times with and without multiprocessing

3.3. Recurrent Neural Networks

Although RNNs are in principle intuitive solutions for time series modeling due to their ability to track temporal features, training them using gradient descent methods is very slow because the error vanishes as it gets backpropagated. However, Long Short-Term Memory (LSTM) networks and Gated Recurrent Units (GRUs) were developed to combat this vanishing gradient problem. They were our first choice in developing models for this dataset. However, to mitigate this potential problem of vanishing gradients further, we employed time series feature extraction to reduce the length of each segment.

LSTM/BiLSTM LSTMs avoids the vanishing gradient problem by augmenting a recurrent unit with "forget" gates, which allows errors to flow backwards through unlimited numbers of virtual layers unfolded in space. Since the acoustic signal segments are temporally related, we experimented with LSTMs and bi-directional LSTMs to capture forward and backward temporal relations across input sequences. However, the LSTMs were not able to learn any temporal relations in the data and performed poorly, as shown in Table 2, either because there were too much noise or our feature engineering were not able to capture the temporal relations properly.

GRU Using similar features as in LSTMs, we experimented with GRUs as well, since we saw that one of the Kaggle submissions had decent results using CuDNNGRU on Keras. Alas, the GRUs also performed poorly like the LSTMs, which should have been expected since both of them are rather similar. Attempts to reproduce the Keras kernel mentioned above in PyTorch have also failed, possibly because the models work differently in the two frameworks.

3.4. Convolutional Neural Networks

Convolutional Neural Nets (CNNs) have become incredibly popular for their effectiveness in computer vision tasks, but only very recently has there been interest in using CNNs for temporal data. While CNNs are memoryless unlike RNNs, it is possible to imagine that CNNs would be able to capture patterns in temporal space like they do for images. Indeed, it has been shown that CNNs are a strong baseline for time series classification tasks, while requiring no heavy preprocessing or feature crafting to achieve competitive results [13]. We thus trained CNN-based models for this competition too.

Spectrogram 2D CNN We first tried attempting conversion of Signal conversion from the time to frequency space using the Fourier transform can be used to generate two-dimensional spectrograms, which can then, like images, be analyzed using CNNs. Both the raw signals and other features may be converted to spectrograms for further analysis. This method performed poorly. When encountering a pixel in an image, it can most often be assumed to belong to a single object. Acoustic data does not separate into layers on a spectrogram but sum together into a distinct whole, which means a particular observed frequency in a spectrogram cannot be assumed to belong to a single sound. This makes it difficult to separate features and events in spectrogram representations [14].

Thus, in the following models, we decided to follow the methodology described by Wang, Yan, and Oates [13], and only used the Z-Normalized raw acoustic data for training, allowing the convolutional layers to extract features from the data automatically rather than doing our own manual preprocessing.

Causal Dilated Convolutions Inspired by the recent success of WaveNet at autoregressive sequence-to-sequence generation tasks [8], we decided to develop a model using causal dilated convolutions. This model was a fully convolutional network, where the convolutional layers have various dilation factors that allow its receptive field to grow exponentially with depth and cover thousands of timesteps (Fig 4). At training time, each output step is only dependent on the input steps that occur before it in temporal space, leading to a causal relation bound in time.

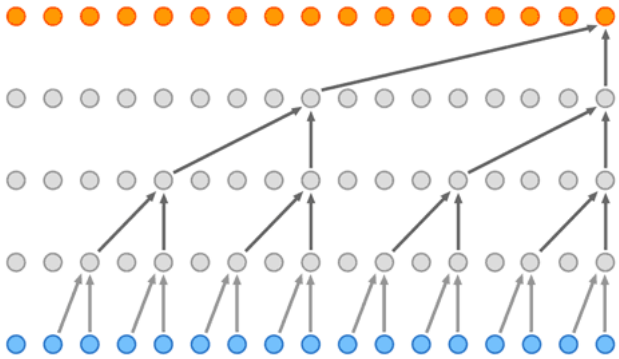

Figure 4. Causal dilated convolutional layers. Image from [8]

However, we found that there was little benefit enforcing causality for tasks other than autoregressive generation, because the task provides the data in full. Non-causal connectivity is strictly more powerful in such a case, as shown in the results of the models.

ResNet In a review of deep learning techniques for time series classification, the authors found that a simple residual net was capable of being competitive with state-of-

the-art approaches like Hierarchical Vote Collective of Transformation-based Ensembles (HIVE-COTE) [3]. We replicated this ResNet structure (Figure 5) and trained it on the Z-normalized raw data, and were impressed by the speed at which it trained and the final results that it achieved.

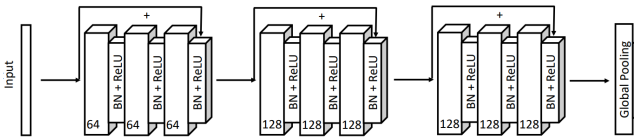

Figure 5. ResNet structure. Image from [13]

DenseNet Following up with the success of the residual net, rather than simply increasing the depth of the net, we changed to a DenseNet model that has become popular in computer vision circles due to its ability to achieve similar accuracy to ResNet but with far less parameters to learn, which is possible due to how each layer is connected to every other layer in a feed-forward fashion [5]. This DenseNet model trained even faster, and resulted in even better results.

4. Results, Tuning, and Discussion

In Table 2, the validation mean absolute error (MAE) scores of each of the methods we tried is listed.

Algorithm	Features	vMAE	tMAE
FCN	Raw data	3.02	-
RNN-LSTM	Raw data	3.11	-
RNN-LSTM	Statistics	3.03	3.16
RNN-BiLSTM	Statistics	2.94	-
RNN-GRU	Statistics	3.02	3.24
CNN-Spec	Transform	3.11	-
WaveNet	Raw data	4.15	4.64
PWaves-ResNet	Z-Norm	2.27	1.87
PWaves-DenseNet	Z-Norm	2.20	1.69

Table 2. Preliminary results for model selection using validation MAE and test MAE.

Despite their temporal nature, RNN-based models do not perform well on this regression task, while the CNN-based models produce results that are competitive with the other entries on the leaderboard. In the following sections, we will describe why the RNN-based models may have failed, and also detail our further efforts at improving the results of the DenseNet model.

4.1. The Flaws of RNNs

When dealing with massive sequences like the ones in this dataset, RNN-based techniques suffer from two crucial flaws that we think resulted in their poor performance when compared to CNN-based ones.

Unlimited Context Provides No Advantage Recent theoretical work offers evidence that the unlimited context offered by RNNs do not contribute significantly to prediction tasks [6]. The "infinite memory" advantage of RNNs is largely absent in practice, which Google showed using the example that a LSTM n-gram model with n = 13 words of memory is as good as an LSTM with arbitrary context [2]. This could be either because the truncated backpropagation through time of RNNs cannot learn patterns significantly longer than k steps, or because models trained through gradient descent simply cannot have long term memory. Thus, over a massive 150000 sequence like the one in this dataset, it seems unlikely that RNNs are capable of determining any long term patterns that CNNs cannot.

Resource Bottleneck Given that RNN models trained in practice are effectively feed-forward networks like CNNs, they lose out due to their inefficient use of resources. During inference when using a RNN-based model, the input batch is only 1 or a small number, and the dependence of each timestep's computation on the previous timestep's output limits parallelism, resulting in slow training on very long sequences. CNNs are capable of processing entire sequences in parallel, resulting in a far more efficient use of computational power. This results in shorter training times for CNNs, while still resulting in similar accuracy [1].

4.2. Finetuning the DenseNet

Having selected the DenseNet as our best model, we proceeded with improving the model to improve our loss. One problem that we faced with the first DenseNet was that it overfitted very quickly. We employed several techniques to mitigate this problem.

Dropout We used a dropout layer after each dense layer, which has been shown to be an effective regularization technique by randomly omitting feature detectors, preventing complex co-adaptations [4].

AdamW Using stochastic gradient descent to optimize the model was slow, but using an adaptive method like Adam seemed to result in severe overfitting. We thus implemented AdamW, or Adam with decoupled weight decay regularization, which decouples the optimal choice of weight decay factor from the setting of the learning rate [7]. We set the initial learning rate of the model to a high value determined by the learning rate range test, described by Smith [11]. This large initial step size allowed for the

model to search over a larger search space for a better minima, and the adaptive nature of AdamW prevented underfitting.

Memory-Efficient DenseNet The original DenseNet implementation produces feature maps that grow quadratically with network depth. By using shared memory allocations, the memory cost is reduced from quadratic to linear. With no GPU bottleneck, we could train a far deeper DenseNet [9].

Final Parameters and Results Using the improvements listed above, our final DenseNet model achieved a validation loss of 2.18, and a test loss of 1.659, a slight improvement. The final DenseNet has depth of 100, and a growth rate of 48, resulting in 3.4 million trainable parameters. Each dropout layer has a dropout chance of 0.2.

The hyperparameters we used are listed in Table 3. We implemented our code in PyTorch 1.0.1.

Hyperparameters	
Batch size	128
Optimizer	AdamW
Initial Learning rate	5e-1
Weight Decay	0.001

Table 3. Hyperparameters for reproducing our results.

5. Conclusion

In this project, we trained several RNN-based and CNN-based models for predicting time to laboratory earthquakes using acoustic data, and compared their results. Despite time series prediction being a more novel and recent application for CNNs, the CNN-based models were significantly more successful than the RNN ones, while requiring no manual feature extraction from the data. We were able to achieve competitive results on the test set using a CNN comparable to the results achieved using gradient boosting machine learning techniques such as LightGBM, which is the state-of-the-art technique for time series modeling tasks such as this.

5.1. Future Work

Despite the lower effectiveness of the RNNs in our experiments, we aim to continue searching for better features in the data, as manual feature extraction might be useful for CNN analysis as well. The distant failure features in Fig. 2 might not be picked up with our set of statistics, nor is there any guarantee of a correct window size. With better feature selection, we might be able to carry out activation maximization to recover signals that maximally explain the predicted TTFs, aiding seismologists and geologists in visually identifying some early onset signals that might predict an earthquake. Another possible approach we could take would be to image the time series using Gramian Angular Summation/Difference Fields (GASF/GADF) as described by Wang and Oates [12].

We thank Dr. Alexander Binder and teaching assistants Chen Zihan and Flavio Toffalini for their assistance in the deep learning course and support for this work.

References

[1] James Bradbury et al. “Quasi-Recurrent Neural Networks”. In: *CoRR* abs/1611.01576 (2016). arXiv: 1611.01576. URL: http://arxiv.org/abs/1611.01576.

[2] Ciprian Chelba, Mohammad Norouzi, and Samy Bengio. “N-gram Language Modeling using Recurrent Neural Network Estimation”. In: *CoRR* abs/1703.10724 (2017). arXiv: 1703.10724. URL: http://arxiv.org/abs/1703.10724.

[3] Hassan Ismail Fawaz et al. “Deep learning for time series classification: a review”. In: *CoRR* abs/1809.04356 (2018). arXiv: 1809.04356. URL: http://arxiv.org/abs/1809.04356.

[4] Geoffrey E. Hinton et al. “Improving neural networks by preventing co-adaptation of feature detectors”. In: *CoRR* abs/1207.0580 (2012). arXiv: 1207.0580. URL: http://arxiv.org/abs/1207.0580.

[5] Gao Huang, Zhuang Liu, and Kilian Q. Weinberger. “Densely Connected Convolutional Networks”. In: *CoRR* abs/1608.06993 (2016). arXiv: 1608.06993. URL: http://arxiv.org/abs/1608.06993.

[6] Sham M. Kakade et al. “Prediction with a Short Memory”. In: *CoRR* abs/1612.02526 (2016). arXiv: 1612.02526. URL: http://arxiv.org/abs/1612.02526.

[7] Ilya Loshchilov and Frank Hutter. “Fixing Weight Decay Regularization in Adam”. In: *CoRR* abs/1711.05101 (2017). arXiv: 1711.05101. URL: http://arxiv.org/abs/1711.05101.

[8] Aäron van den Oord et al. “WaveNet: A Generative Model for Raw Audio”. In: *CoRR* abs/1609.03499 (2016). arXiv: 1609.03499. URL: http://arxiv.org/abs/1609.03499.

[9] Geoff Pleiss et al. “Memory-Efficient Implementation of DenseNets”. In: *CoRR* abs/1707.06990 (2017). arXiv: 1707.06990. URL: http://arxiv.org/abs/1707.06990.

[10] Bertrand Rouet-Leduc et al. “Machine Learning Predicts Laboratory Earthquakes”. In: *Geophysical Research Letters* 44.18 (2017), pp. 9276–9282. DOI: 10.1002/2017GL074677. URL: https://agupubs.onlinelibrary.wiley.com/doi/abs/10.1002/2017GL074677.

[11] Leslie N. Smith. “No More Pesky Learning Rate Guessing Games”. In: *CoRR* abs/1506.01186 (2015). arXiv: 1506.01186. URL: http://arxiv.org/abs/1506.01186.

[12] Zhiguang Wang and Tim Oates. “Imaging Time-Series to Improve Classification and Imputation”. In: *CoRR* abs/1506.00327 (2015). arXiv: 1506.00327. URL: http://arxiv.org/abs/1506.00327.

[13] Zhiguang Wang, Weizhong Yan, and Tim Oates. “Time Series Classification from Scratch with Deep Neural Networks: A Strong Baseline”. In: *CoRR* abs/1611.06455 (2016). arXiv: 1611.06455. URL: http://arxiv.org/abs/1611.06455.

[14] Lonce Wyse. “Audio Spectrogram Representations for Processing with Convolutional Neural Networks”. In: *CoRR* abs/1706.09559 (2017). arXiv: 1706.09559. URL: http://arxiv.org/abs/1706.09559.