LANL Earthquake Prediction

Golam Md Muktadir*
muktadir@ucsc.edu
University of California, Santa Cruz
CA, USA

Cagan Bakirci†
cbakirci@ucsc.edu
University of California, Santa Cruz
CA, USA

George Redhead‡
gredhead@ucsc.edu
University of California, Santa Cruz
CA, USA

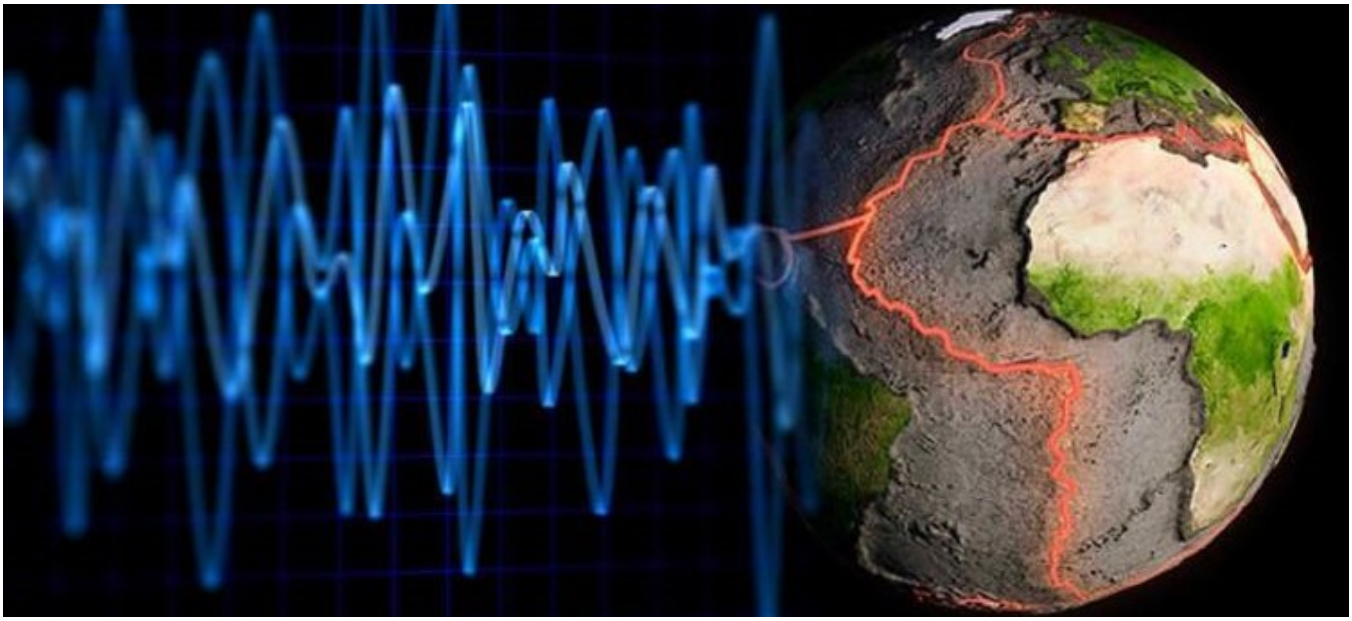

Figure 1: predict the time remaining before laboratory earthquakes occur from real-time seismic data

ABSTRACT

Forecasting earthquakes is one of the most important problems in Earth science because of their devastating consequences. 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. In this project, we develop neural network models to predict the **time remaining before laboratory earthquakes** [1] occur from real-time seismic data [5]. In a classic laboratory experiment of repeating earthquakes, we find that the seismic signal follows a specific pattern with respect to fault friction, allowing us to determine the fault's position within its failure cycle. Our best model scored among top 5% in a recent Kaggle Competition [4]. All our final models scored better than baseline models.

CCS CONCEPTS

• **Mathematics of computing** → **Time series analysis**; *Exploratory data analysis*; *Computation of transforms*; • **Computing methodologies** → **Neural networks**; *Supervised learning by regression*; *Feature selection.*

KEYWORDS

Regression, neural networks, Time series, earthquake prediction, seismic data analysis, CNN in Signal Analysis, RNN in Signal Analysis, Large-scale analysis

ACM Reference Format:
Golam Md Muktadir, Cagan Bakirci, and George Redhead. 2018. LANL Earthquake Prediction. In *Proceedings of Course Project.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/1122445.1122456

1 MOTIVATION AND OBJECTIVE

Predicting the timing and magnitude of an earthquake is a fundamental goal of geoscientists. Recently Los Alamos National Laboratory, hich enhances national security by ensuring the safety of the U.S. nuclear stockpile, developing technologies to reduce threats

*CNN models and Preprocessing
†RNN Models
‡Data Analysis, FFN Model

Course Project, June 2019, CMPS 242, UC Santa Cruz, CA, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/1122445.1122456

from weapons of mass destruction, and solving problems related to energy, environment, infrastructure, health, and global security concerns, hosted a competition on Earthquake prediction based on seismic signals.

There has been a few research on this topic since 2017, and researchers has shown that laboratory earthquakes can be predicted based on seismic data.A classical approach to determining that an earthquake may be looming is based on the interevent time (recurrence interval) for characteristic earthquakes, earthquakes that repeat periodically [8]. But the problem with classical approach is the period is not consistent and varies significantly every cycle. Because of advancement in instrumentation quality and density, we have come to know previously unidentified patterns regarding earthquakes and fault slips.

Interestingly, acoustic/seismic precursors to failure appear to be universal phenomenon in materials. Based on this, authors successfully applied Machine Learning predicting earthquakes based on seismic data in [7]. But their process (random forest) actually overfits and does not generalize well for predictions. In [6], researchers corroborate that laboratory faults do not fail randomly but in a highly predictable manner. But the prediction accuracy still needs to be improved and generalized.

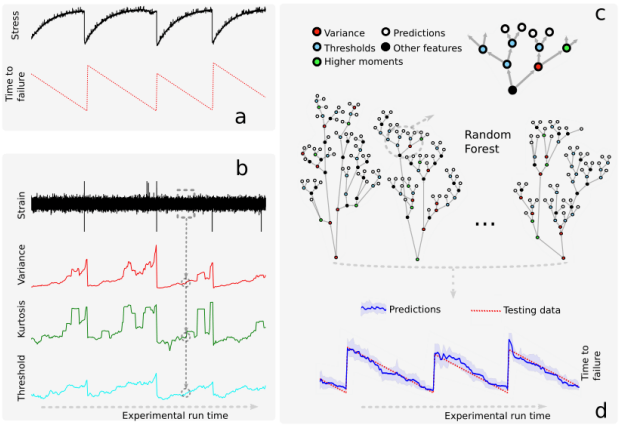

Figure 2: Random Forest Approach [7]

A recent research [2] shows that it's possible to predict both slow and fast earthquakes using the similar approach as [7] using gradient boosted trees algorithm, based on decision trees. But the problem remains the same, these methods over fits severely. Also, we observed the similar overfitting nature in the Kaggle competition where competitors used different gradient boosting algorithms such as LightGBM, CatBoost, XGBoost, AdaBoost. They performed very well with training and validation, but failed to show any significant improvement in the final test.

So, based on our observation, we decided to build a solution with generalization as the goal rather than scoring the highest.And in this work, we present several neural network based solutions which generalizes well over both the training and test stages.

2 DATASET

The data comes from a well-known experimental setup used to study earthquake physics. The acoustic_data input signal is used to predict the time remaining before the next laboratory earthquake (time_to_failure).

2.1 Data files

- **train.csv** - A single, continuous training segment of experimental data. It contains 650+ Millon samples.
- **test** - A folder containing many small segments of test data. Each segment consists of 150,000 samples.
- **sample_sumbission.csv** - A sample submission file in the correct format.

2.2 Data fields

- **acoustic_data** - the seismic signal [int16]
- **time_to_failure** - the time (in seconds) until the next laboratory earthquake [float64]
- **seg_id** - the test segment ids for which predictions should be made (one prediction per segment)

2.3 Simple Exploration

Here goes some characteristics of the data.

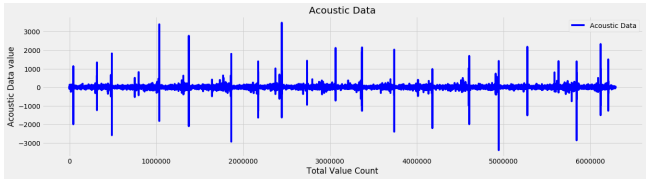

Figure 3: Seismic signal of the training samples

Figure 4: Time to failure of the training samples

Next bin has next 4096 data points/records

Bin	Acoustic_data	Time_to_failure
0	12	1.469099983
	6	1.469099981
	...	.....
1	-5	1.250955433
	2	1.250955100
	...	.....
2	3	0.335005446

Figure 5: Training data is grouped into Bins of 4096 samples.

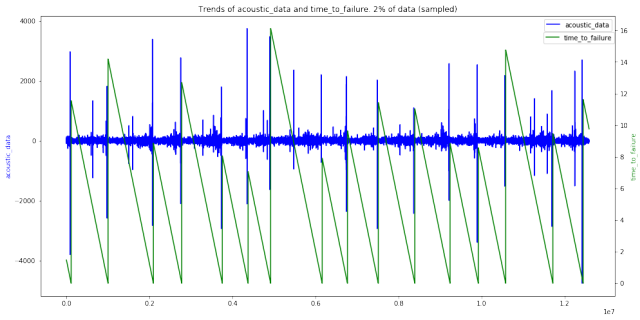

Figure 6: Trend of acoustic_data and time_to_failure for 2% of the data

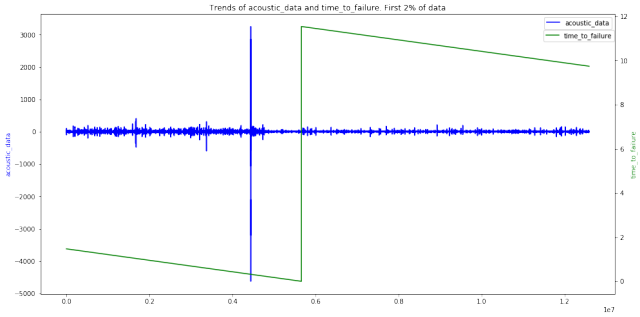

Figure 7: Samples near a failure

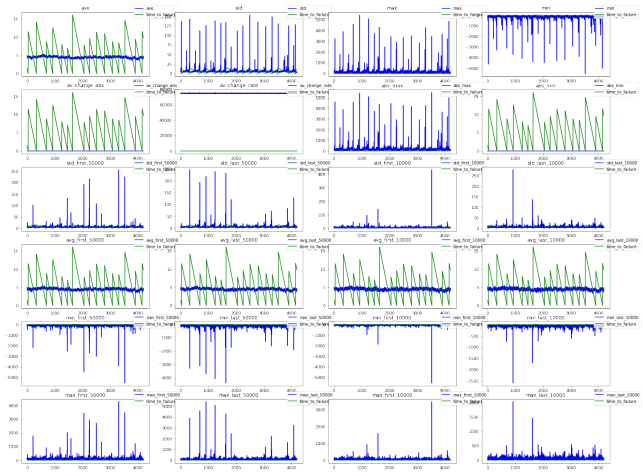

Figure 8: Statistical features

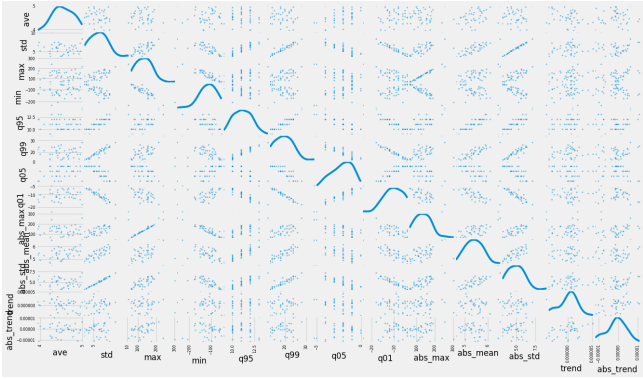

Figure 9: Co-relation of statistical features

3 MODELS AND ALGORITHMS

3.1 Overall Architecture

We have two main processes: one for pre-processing data which ultimately produces input embedding to the models, another for training the models.

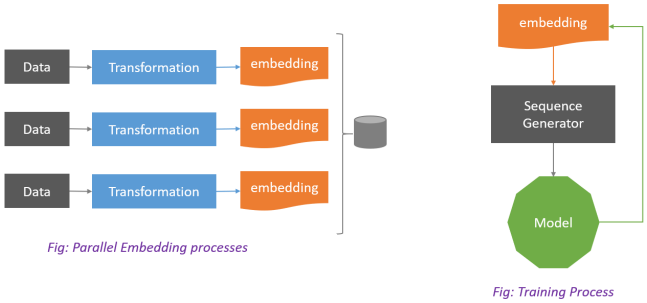

Figure 10: Two main processes of the overall solution

3.2 Pre-processing and Embedding

Steps:

- **Extracting bins from training data:** This is done by our bin creator processes. We created both actual bins and absolute bins. Absolute bins converts acoustic data to absolute values. It requires about 16 hours to extract bins.
- **Build Scalers:** We also used scikit-learn to train robust scalers from training data. This scalers were reused for test dataset, too.
- **One-Stat Embedding:** One Stat Embeddings have 1-to-1 relation with bins. So, for each bin we extracted statistical features and saved it as the embedding. It requires about a day to create the embeddings.
- **Multi-Stat Embedding:** These have 1-to-many relation with bins. In fact, for each 36 bins we create a single statistical feature set and saved it as the embedding.It requires about a day to create the embeddings.
- **CNN-Stat Embedding:** This embedding has 36 statistics for 36 consecutive bins as rows. It also have 1-to-many relation with bins, but it does not create a single statistical feature set. It requires about two days to create the embeddings.
- **Over-sampling:** To create more data for training, we used windowing-technique.

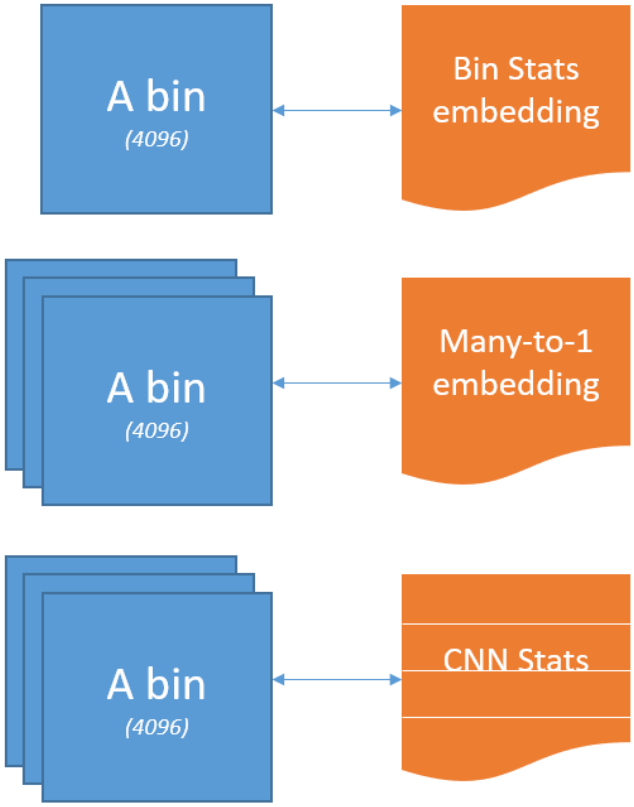

Figure 11: 3 different embeddings for different models.

3.3 Feature Engineering

Total number of features we developed: 275 for CNN models, 31 for RNN models.

3.3.1 Statistical Measures: We calculated 15-18 statistical features on different transformation of the raw data. The base features are:

- Mean
- Median
- Standard Deviation
- Minimum and Maximum
- Range
- Different percentiles
- Entropy
- Kurtosis
- Skewness
- Correlations
- etc.

3.3.2 Transformations:

- Absolute values
- FFT (Fast Fourier Transform)
- Spectogram
- Different Rolling Means [5, 10, 20, 40, 100, 1000]
- Linear seasonality
- First-order seasonality

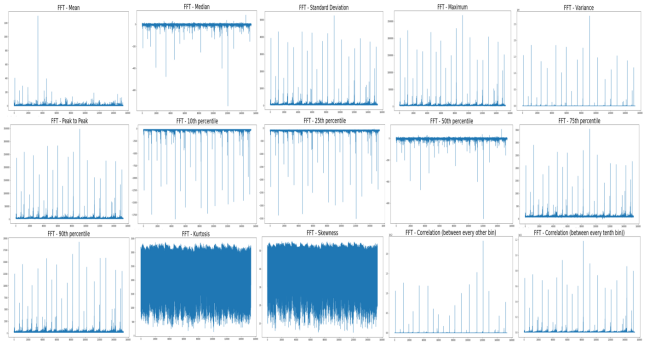

Figure 12: Statistical features on fft.

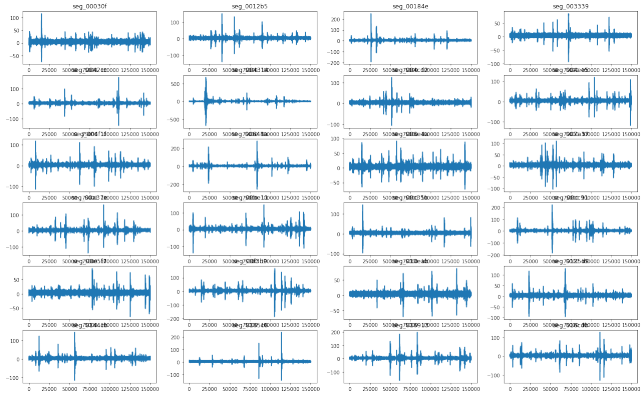

Figure 13: Statistical features on Test set.

3.3.3 *Feature Importance:*

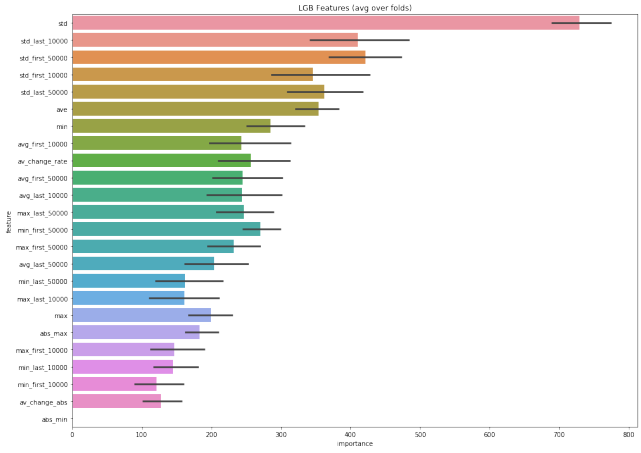

Figure 14: Importance of statistical features from [3]

3.4 Models

All our models are neural network-based. We built a big set of Fully Forward Networks (FFNs), Recurrent Neural Networks (RNNs), and Convolutional Neural Networks (CNNs). We list here the most important ones only. We excluded different variations, but those are available in our repositories.

Loss Functions:

- MSE
- Cosine Similarity

Optimizers:

- RMSProp
- Adam
- Nadam

Evaluation Metric:

- MAE

FFNs:

3.4.1 *First FFN:.* Uses raw data as input.

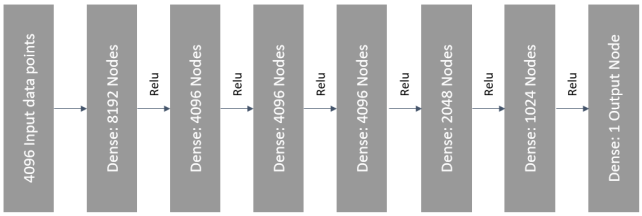

Figure 15: This FFN performs badly.

3.4.2 *A very well performing FFN:.* Uses One-Stat embeddings.

```
import logging, sys, math,os
exec(open("estimator/initKeras.py").read())

model_input = layers.Input( shape = ( 15 + 6 * 27 + 2 +
    15+ 3 * 27,  ) )
x = layers.Dense(64)(model_input)
x = layers.LeakyReLU(alpha=0.1)(x)
x = layers.Dense(32)(x)
x = layers.LeakyReLU(alpha=0.1)(x)

x = layers.Dense(48)(x)
x = layers.LeakyReLU(alpha=0.1)(x)
x = layers.Dropout(0.2)(x) # dropout

x = layers.Dense(8)(x)
x = layers.LeakyReLU(alpha=0.1)(x)

output = layers.Dense(1, activation=activations.relu)(x)

model = models.Model(model_input, output, name = "
    MixedFFN")
```

Listing 1: FFN final.

RNNs The obvious choice for modeling Time Series Data is using RNNs with LSTMs because the output depends on previous history of inputs.

3.4.3 Final RNN. Uses One-Stat embeddings.

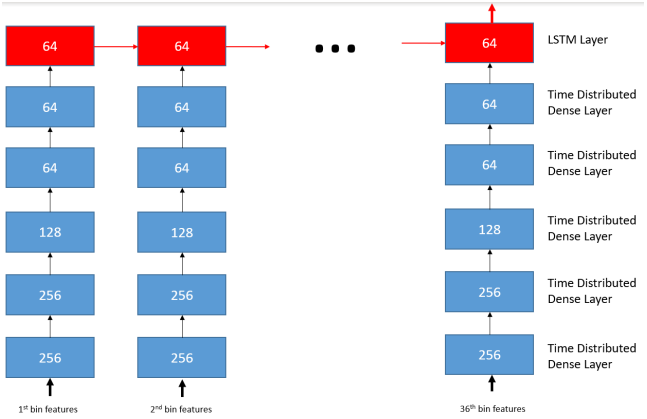

Figure 16: Final RNN scoring better than Baseline at Kaggle.

CNNs: While CNN's are not used in Time Series Modeling, we decided to explore it because of our development of CNN embedding:

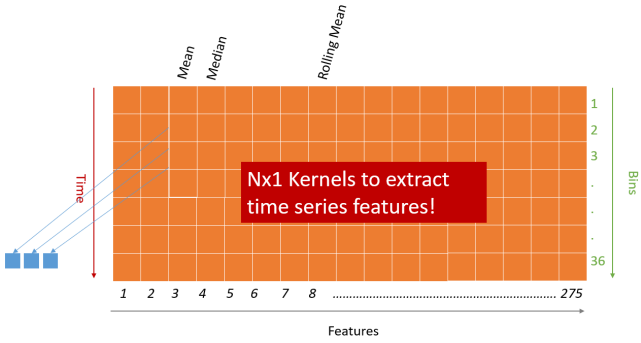

Figure 17: This FFN performs badly.

3.4.4 First CNN:. We also tried different variations of this model with different depth and width. We use Leaky Relu as the hidden activation layers. All the simple models scored better than baseline models.

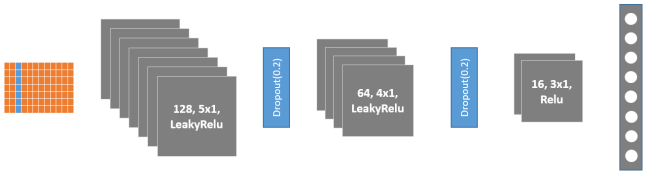

Figure 18: Fully CNN Base Code.

```
model_input = layers.Input( shape = ( 36, 275, 1 ) )
x = layers.Conv2D(64,
        kernel_size = (20,1),
        padding = 'same'
        )(model_input)

x = layers.LeakyReLU(alpha=0.1)(x)

x = layers.Conv2D(32, kernel_size=(10,1), padding='same')
    (x)
x = layers.LeakyReLU(alpha=0.1)(x)
x = layers.Dropout(0.2)(x)

x = layers.Conv2D(32, kernel_size=(10,1), padding='same')
    (x)
x = layers.LeakyReLU(alpha=0.1)(x)
x = layers.Dropout(0.1)(x)

x = layers.Conv2D(16, kernel_size=(3,1), activation=
    activations.relu, padding='same')(x)
x = layers.Flatten()(x)
x = layers.Dense(1)(x)

model = models.Model(model_input, x, name = "1DCNN-
    ColumnKernel")
```

Listing 2: Fully CNN

3.4.5 Upsampling CNN:. We also tried different variations of this model with different depth and width. We inferred that this architecture removes some noises in the signal and produces best results. We also experimented with different kernels. Long kernels seems to work better in our experiments.

Figure 19: Core upsampling framework.

```

model_input = layers.Input( shape = ( 36, 275, 1 ) )
x = layers.Conv2D(32,
        kernel_size = (20,1),
        padding = 'same'
        )(model_input)

x = layers.LeakyReLU(alpha=0.1)(x)
x = layers.Dropout(0.1)(x)

x = layers.Conv2D(64, kernel_size=(10,1), activation=
    activations.tanh, padding='same')(x)
x = layers.MaxPooling2D(pool_size=(4,1), padding='same')(
    x)
x = layers.UpSampling2D(size=(4,1))(x)

x = layers.Conv2D(16, kernel_size=(10,1), activation=
    activations.tanh, padding='same')(x)
x = layers.MaxPooling2D(pool_size=(4,1), padding='same')(
    x)
x = layers.UpSampling2D(size=(4,1))(x)

x = layers.Conv2D(8, kernel_size=(3,1), activation=
    activations.relu, padding='same')(x)
x = layers.Dropout(0.1)(x)
x = layers.Flatten()(x)
x = layers.Dense(1)(x)

model = models.Model(model_input, x, name = "1DCNN-
    ColumnKernel-Max-Pool-UpSampling")
model.summary()
```

Listing 3: Upsampling Base Code

3.4.6 Encoder Decoder Estimator CNN:. We also tried different variations of this model with different depth and width. Interestingly, this framework becomes more aggressive in reducing noise. It also has comparable performance to upsampling CNN.

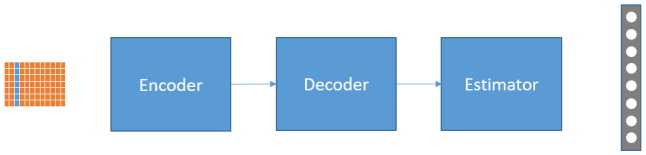

Figure 20: Encoder-Decoder-Estimator framework.

```
model_input = layers.Input( shape = ( 36, 275, 1 ) )

#encoder
x = layers.Conv2D(64, kernel_size = (20,1), padding = '
    same', activation=activations.relu)(model_input)
# x = layers.LeakyReLU(alpha=0.1)(x)
x = layers.MaxPooling2D(pool_size=(4,1), padding='same')(
    x)

x = layers.Conv2D(24, kernel_size=(10,1), activation=
    activations.relu, padding='same')(x)
x = layers.MaxPooling2D(pool_size=(4,1), padding='same')(
    x)

x = layers.Conv2D(8, kernel_size=(5,1), activation=
    activations.relu, padding='same')(x)
x = layers.MaxPooling2D(pool_size=(4,1), padding='same')(
    x)

#decoder
x = layers.Conv2D(8, kernel_size=(5,1), activation=
    activations.relu, padding='same')(x)
x = layers.UpSampling2D(size=(4,1))(x)

x = layers.Conv2D(24, kernel_size=(10,1), activation=
    activations.relu, padding='same')(x)
x = layers.UpSampling2D(size=(4,1))(x)

x = layers.Conv2D(64, kernel_size=(20,1), activation=
    activations.relu, padding='same')(x)
x = layers.UpSampling2D(size=(4,1))(x)

#estimator
x = layers.Conv2D(16, kernel_size=(5,1), activation=
    activations.relu, padding='same')(x)
x = layers.MaxPooling2D(pool_size=(4,1), padding='same')(
    x)
x = layers.Dropout(0.4)(x)
x = layers.Flatten()(x)
# x = layers.GlobalMaxPooling2D()(x)
x = layers.Dense(1)(x)

model = models.Model(model_input, x, name = "1DCNN-
    Encoder-Decoder-Estimator-100epoch")
```

Listing 4: Encoder-Decoder-Estimator Base Code

3.4.7 2D Kernel Models: We also tried 2D Kernels. They have similar performance as 1D Kernels. We conclude that the model eventually learns column trends.

```
model_input = layers.Input( shape = ( 36, 275, 1 ) )
x = layers.Conv2D(64,
        kernel_size = (5,1),
        padding = 'same'
        )(model_input)

x = layers.LeakyReLU(alpha=0.1)(x)
x = layers.MaxPooling2D(pool_size=(2,2), padding='same')(
    x)

x = layers.Conv2D(64, kernel_size=(1,5), padding='same')(
    x)
x = layers.LeakyReLU(alpha=0.1)(x)
x = layers.MaxPooling2D(pool_size=(2,2), padding='same')(
    x)

x = layers.Conv2D(32, kernel_size=(3,3), padding='same')(
    x)
x = layers.LeakyReLU(alpha=0.1)(x)
x = layers.Dropout(0.2)(x)


x = layers.Conv2D(16, kernel_size=(3,1), activation=
    activations.relu, padding='same')(x)
x = layers.Flatten()(x)
x = layers.Dense(1)(x)

model = models.Model(model_input, x, name = "2DCNN-
    MixedKernel")
model.summary()
```

Listing 5: 2D Kernel model

4 RESULTS AND ANALYSIS

4.1 Analysis of Dataset

The most significant challenge we had to face as a team was understanding the data. Our initial perception of the data was that it was one continuous datastream as shown in figures 3 and 5. We assumed the data was one continuous datastream since that is how it was described by the competition host. But in reality data is collected in bins of size 4096 (which is not constant either) and has a gap of approximately 1ms in between two consecutive bins.

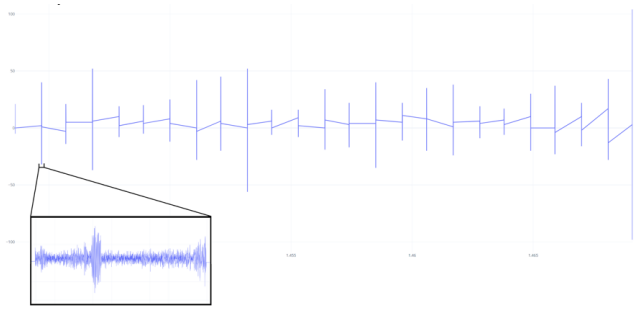

Figure 21: Gap in the time series data.

Then after we extracted bins, we found bins of sizes 4096, 4095, and 8192. We fixed that before any other pre-processing.

4.1.1 Convergence Problem: The hyperspace seems to be flat around a local minima. We tried different optimizers, learning rates, and loss functions. But it seems the loss averages out at around 7.5 MSE.

4.1.2 Performance: Both the training and inference processes are pretty fast after data pre-processing. Pre-processing takes a huge amount of time (in days for each type of embedding and bin creation). Our feature engineering was the slowest process. In a 6 core and 32 GB machine it takes 16 - 24 hours for each run. Any mistakes slowed us down by at least a day.

4.1.3 Kaggle Scores: We also submitted our test results to Kaggle and here goes a small list of those:

Architecture	Kaggle Score
Small 3 Layer FFN	2.86
1DCNN-ColumnKernel	2.74915
1DCNN-ColumnKernel-UpSampling.csv	**2.49779 (top 5%)**
1DCNN-ColumnKernel-Tanh.csv	2.66118
1DCNN-ColumnKernel-Avg-Pool-UpSampling	2.58314
1DCNN-Encoder-Decoder-Estimator.csv	2.73287
2DCNN-MixedKernel.csv	**2.55181**
RNN-First	3.39999
RNN-Poisson	3.46080
RNN-Final	**2.88203**

Figure 22: Kaggle private scores (MAE) between 2.2 and 7 (Final leaderboard standing)

4.1.4 Learning Outcomes:

- Plotting hundreds of millions of datapoints using different tools. Common tools like matplotlib fails frustratingly.
- Statistical Data Analysis
- Basic Time Series Analysis
- Working with huge amount of data
- Keras Sequence Generator
- Using Scikit-Learn Robust Scalers
- Denoising CNNs, different CNN architectures
- RNNs and LSTMs
- Pandas - dataframe for python
- Google Colab, Jupyter Notebook
- Using multiple cores with multiple pre-proprocessing OS processes.
- Advanced image positioning in Latex.
- Source code highlighting in Latex.

5 CONTRIBUTION

5.0.1 George Redhead - 10-15 hours a week.

- Analyzed raw datasets
- Initial data visualization tasks.
- First neural model with raw data.
- Contributed with statistical feature engineering.
- Helped Cagan to build RNN models and create train generator.
- Project presentation slides.
- participated a project meeting and a coding session every week.

5.0.2 Cagan Bakirci - 10-15 hours a week.

- Extracted FFT features
- Tried Spectogram
- Built RNN models
- Project presentation slides.
- participated a project meeting and a coding session every week.

5.0.3 Golam Md Muktadir - 15-25 hours a week.

- Managed progress and skill development in team members.
- Wrote the **resumable** pre-processing pipelines
- Found the inconsistencies with raw data
- Planned and implemented embeddings.
- Built CNN models
- Built the training sequence generator
- Efficient plotting with hundreds of millions of data points.
- Participated a project meeting and a coding session every week.
- Project presentation slides and designs.
- Report in Latex.

6 FUTURE WORK

First, we would like to investigate more on material failure properties, disaster patterns, and on-going research for predictions of catastrophes. These knowledge will help us to engineer related features from seismic data.

Second, there is a huge scope of improving the models. We experimented with small networks and small number of epochs. With more computing resources available, we would like to build robust models that may yield even better results.

Third, we would like to explore both CNN and RNN based solutions and share features among them to see which performs better in the context.

REFERENCES

[1] Menlo Park Science Center. 2017. Earthquake in a lab. https://www.youtube.com/watch?v=m_dBwwDJ4uo. (2017). [Online; accessed June 7th, 2019].
[2] Claudia Hulbert, Bertrand Rouet-Leduc, Paul A. Johnson, Christopher Ren, Jacques RiviÃĺre, David C. Bolton, and Chris Marone. 2019. Similarity of fast and slow earthquakes illuminated by machine learning. *Nature Geoscience* 12 (01 2019). https://doi.org/10.1038/s41561-018-0272-8
[3] Andrew Lukyanenko. 2019. Seismic data EDA and baseline. https://www.kaggle.com/artgor/seismic-data-eda-and-baseline. (2019). [Online; accessed June 7th, 2019].
[4] Kaggle Competition on Earthquake. 2019. LANL Earthquake Prediction. https://www.kaggle.com/c/LANL-Earthquake-Prediction/overview. (2019). [Online; accessed June 7th, 2019].
[5] Horst Rademacher. 2016. How Does a Seismometer Work? https://www.youtube.com/watch?v=P7h0JfQ-oHg. (2016). [Online; accessed June 7th, 2019].
[6] Bertrand Rouet-Leduc, Claudia Hulbert, David C. Bolton, Christopher Ren, Jacques RiviÃĺre, Chris Marone, Robert Guyer, and Paul A. Johnson. 2018. Estimating Fault Friction From Seismic Signals in the Laboratory. *Geophysical Research Letters* 45 (01 2018). https://doi.org/10.1002/2017gl076708
[7] Bertrand Rouet-Leduc, Claudia L. Hulbert, Nicholas Lubbers, Kipton M. Barros, Colin Humphreys, and Paul A. Johnson. 2017. Machine Learning Predicts Laboratory Earthquakes. *Geophysical Research Letters* 44 (02 2017). https://doi.org/10.1002/2017gl074677
[8] David Schwartz and K Coppersmith. 1984. Fault Behavior and Characteristic Earthquakes: Examples From the Wasatch and San Andreas Fault Zones. *Journal of Geophysical Research* 89 (07 1984). https://doi.org/10.1029/JB089iB07p05681