Earthquake prediction using machine learning

1 Introduction

Predicting an earthquake is the ultimate goal of seismology. For a long time researchers have tried to reach it using different methods. In the 1970s, multiple techniques, mostly statistical, were developed. However, after multiple years the interest started to decrease as constructing such a model seemed extremely difficult and some believed that it was in fact impossible to predict these events. One of the best performing statistical model is the Epidemic Type Aftershock Sequence (ETAS) which gives probabilities of an earthquake happening in an area, in a certain time window with a magnitude threshold. However, this field has regained new interest recently with the improvement of Machine Learning (ML) models. The prediction of earthquakes were deemed extremely difficult or impossible for a few main reasons:

- the seemingly random events. There is no clear pattern over time for earthquakes which is the main difficulty of this problem.
- a limited amount of data. The precise observations and recordings of earthquakes is pretty recent in history. Compared to geological time, the time frame we work with is extremely limited.
- 3 characteristics have to be predicted: the time, the location and the magnitude. Getting all these 3 accurately is what is considered an actual earthquake prediction.
- the structures at the origin of these events are very complex and hard to study since they are under the earth crust.

ML models are efficient in solving some of these problems and this is why they have gained attention from researchers. Finding hidden patterns from the point of view of humans is one of the main aspect of some of those models. Many ML models have been tested to predict earthquakes [1]. The latest Deep Learning (DL) models can learn complex structures and make links over long period of times in order to make predictions. Those model that are categorized as Recurrent Neural Networks (RNN) are the ones I focused on.

2 Underlying technology

2.1 Recurrent Neural Network

RNN are a type of artificial neural network specialized in dealing with temporal sequence. Thanks to a loop in their layer, the information persists. However, simple RNNs suffer from a problem: their inability to take into account long-term dependencies.

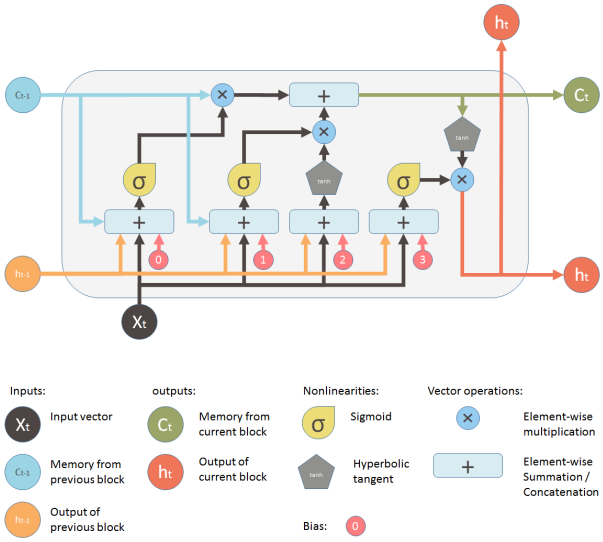

Figure 1: Detailed LSTM model

2.2 Long Short-Term Memory

This is where the Long Short-Term Memory (LSTM) [2] model comes in. It is a special type of RNN specially designed to handle those long-term dependencies. A cell state transfers information almost directly. It is possible to add or remove information to it only through certain gates. Those gates are composed of a sigmoid layer with a multiplication operation. Then the model can be described in 3 different parts.
The forget layer decides what we keep from the previous information. The ouput of the previous LSTM and input of the current LSTM are considered.
The new memory layer is composed by an hyperbolic tangent layer to compute new candidates values and a gate that decides how much of those new values are going to be added to update the current memory.
Finally, the output layer takes into consideration the new cell state, the previous output and the current input. The new cell state is re scaled thanks to a tanh layer between -1 and 1 and then a gate decides how much of that new state is output.

2.3 Gated Recurrent Units

Gated Recurrent Units (GRU) is another type of RNN which uses gates like the LSTM. It can also be broken down in different parts.

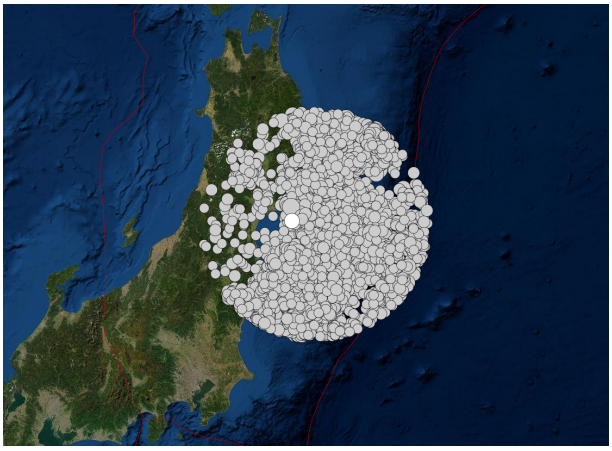

Figure 2: Dataset's earthquakes location

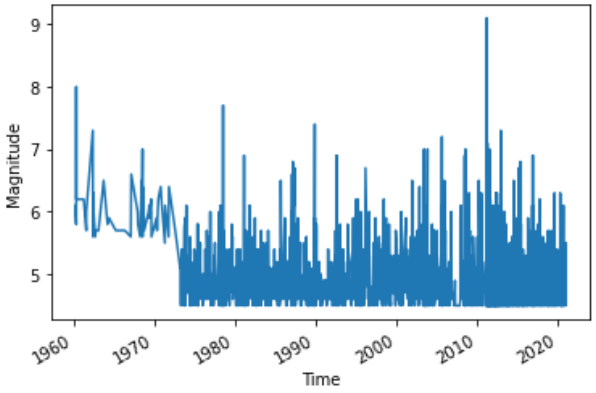

Figure 3: Magnitudes values over time

The reset gate decides how much of the previous state we want to keep. It helps capturing short-term dependencies.
The update gate controls how much of the old state we want copied to the new state. This one is more focused around the long-term dependencies.
Then we have a candidate hidden state which is calculated from the multiplication of the current input and the output of the reset gate which is then fed into a tanh layer.
That candidate state is used to produce the output. It is multiplied by the output of the update gate to determine how much from the hidden candidate is used. Same is done with the current unit memory. Both of those results are then added to produce the output.

2.4 Variational Recurrent Auto-Encoders

Variational Recurrent Auto-Encoders (VRAE) [4] is a type of RNN structured around an encoder and a decoder. First, the input sequence is fed to an RNN model (LSTM/GRU). This is the encoding. The output is used to get the mean and standard deviation. These two parameters define a distribution which is used to create a latent vector. That vector is passed through a linear layer. The result gives the initial state for the decoder RNN. The output of the decoder is the output of this model.

3 Proposed method

Since creating an actual earthquake prediction model is extremely complex, I put my attention first on having a working RNN model with earthquake data. I decided to work with an LSTM as its ability to take into account long-term dependencies seemed perfect for my application. LSTM works well with periodic input data. This is why a simpler problem is considered: predicting the number of earthquakes in the next month. In the case of my application a few other particularity are put in place. For the prediction, the difference in the number of earthquake from one month to another is used. This usually proves to be a better way to use data. Input data is also focused along a main fault line as we will see in the next part. Finally, the basic LSTM model will be improved by different means as to improve the result.

4 Experiments

4.1 Dataset

The data is focused around the Tohoku fault line. I chose this location as it is the most active region in Japan in terms of earthquake. The data is taken from the USGS website as it is the most complete catalog and one of the most widely used data source in this domain. One of the most complete earthquake prediction paper using a statistical method uses it [3]. All earthquakes with a magnitude superior to 3 that happened since the beginning of 1961 up until May 2021 are taken. This gives us around 5000 earthquakes for our data set. However, after analyzing the data a few particularities are found. First of all, the data seem to be incomplete before the year 1973. As shown on the graph with the magnitudes values over time, earthquakes below a magnitude of 5.5 weren't recorded before that year. This shows the difficulty to have complete data. The correctly recorded time period is extremely small compared to the geological times that earthquakes seem to scale in. All the data before the 1973 are removed in consequence.
Then the 2011 Great Tohoku earthquake is also part of this data. This is such a rare event that happens one time in a century or maybe even rarer that in our case it will just skew data in a unpredictable way. Around 2000 earthquakes are recorded around that time which is a bit below half the entire data set. I chose to remove all the

Parameter	Value
Total size	455
Train set size	407
Test set size	48
Epoch	1000
Hidden nodes	20
Batch size	1
Loss function	MSE
Optimization algorithm	ADAM

Figure 4: Simple LSTM parametersS

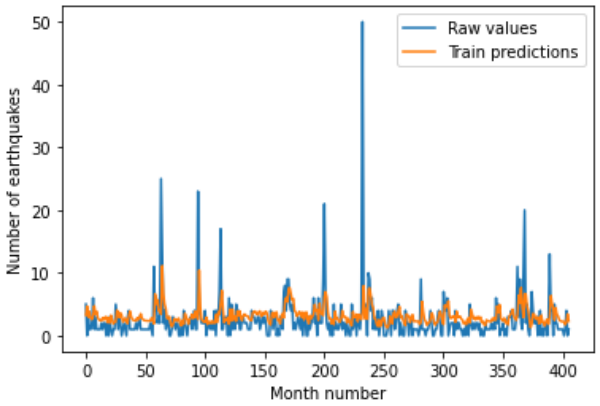

Figure 5: Simple LSTM train set prediction result

data after 2011 as well so that our simple model can try to learn on an easier set of data. Obviously the ultimate goal would be to have a model that can predict those rarest and strongest events. In the end there are around 1500 earthquakes distributed over 455 months.

4.2 Simple LSTM

The first implementation is a plain LSTM with default parameter values. The point is to see what kind of result we obtain with this kind of data. The train result show that the trend is followed pretty well. Since the difference cannot be seen clearly on the graph, looking at raw results the prediction is on time at first but start to have an offset as more time goes on. The loss has a rapid descent before stagnating. The few spikes are due to the fact that the batch size is very small. The test result shows more clearly the problem that started to appear more and more in the train result as the predictions went on. Even though the trend is followed correctly, there is a right-shift in the data. This is a huge issue for prediction problem as having results that shows predictions lagged behind true values are basically not usable. This is a very common problem with RNN research which is

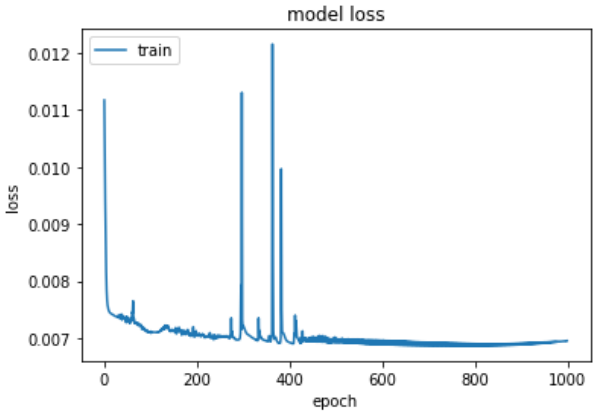

Figure 6: Simple LSTM Loss over time

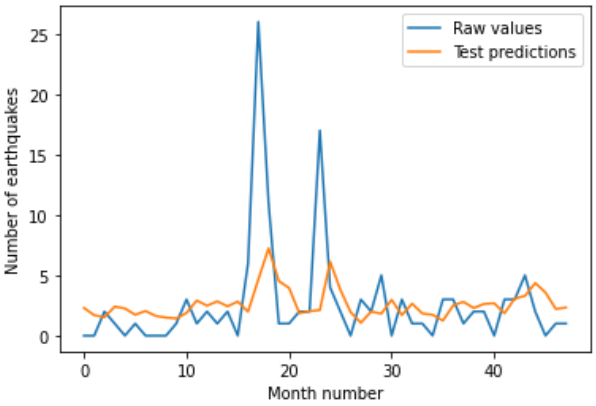

Figure 7: Simple LSTM train set prediction result

often ignored by researchers even though it is essential to many applications. Since this problem that I did not expect appeared in my research, I decided to focus my efforts in solving this issue from then on.

4.3 Time window LSTM

The first attempt at solving this problem was to increase the number of months that were taken into account for each prediction. For the simple LSTM model only the previous month was considered each time making the model limited. A time window parameter is added. I tuned this parameter with Optuna obtaining a value of 3. The trend is now followed almost perfectly. However, the main problem of the right-shift hasn't been solved. Different tries with bigger time window values were also tested since maybe even if it's less accurate it might have been able to solve the main problem. All the other tests were unsuccessful as well.

Parameter	Value
Total size	441
Train set size	393
Test set size	48
Epoch	500
Hidden nodes	20
Batch size	1
Loss function	MSE
Optimization algorithm	ADAM
Timesteps	3

Figure 8: Time window LSTM parameters

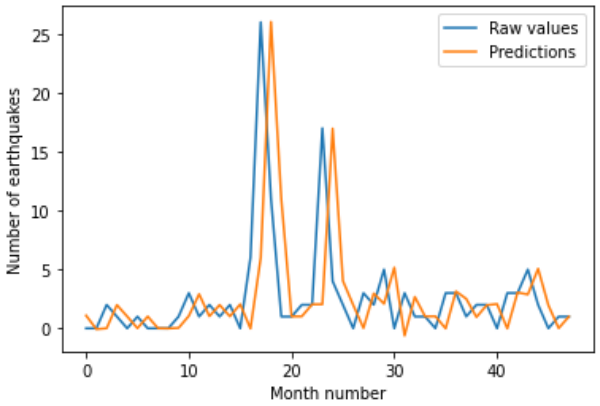

Figure 9: Time window LSTM prediction result

4.4 Custom loss LSTM

The next attempt to solve the right-shift problem was to create a custom loss function. Currently the Mean Squared Error (MSE) is used. It is the most commonly used loss function. However, it's simplicity might be an issue. The new custom loss takes into account the previous true value and calculate another MSE value with the current prediction. The two MSE values are then put together depending on an alpha parameter and if the current true value MSE is bigger than the previous true value MSE. The alpha parameter is set to 1 as a test. This sadly does not solve the right-shift problem again. Other alpha values were tested (0.1, 10, 100) but the right-shift stayed. Other measures must be taken to tackle this problem.

5 Conclusions and future challenges

While this research started with a specific problem to solve, being the prediction of the number of earthquakes per month using an LSTM, it ran into a bigger problem: a lag observed in the predictions compared to the true values. Solving this problem is crucial for all the future predictions done using RNN models but it is very complex hence why many researchers ignore the issue. Next attempt is to try a different approach to solve this problem by implementing a VRAE model and see how we could improve that one. Trying the prediction of other aspects of the earthquake can also be considered.

Algorithm 1 Custom loss function

Require: (y_true: true value, y_pred: predicted value)
Ensure: Loss value
Global y_old // y_old contains the true value from the previous month
$alpha \Leftarrow 1$
$d1 = mean(\sqrt{y\_pred - y\_true})$
$d2 = mean(\sqrt{y\_pred - y\_old})$
$y\_old \Leftarrow y\_true$
return $d1 + alpha * relu(d1 - d2)$

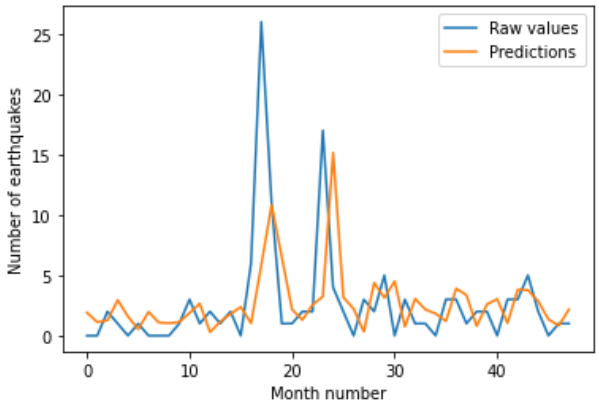

Figure 10: Custom loss LSTM prediction result Alpha=1

References

[1] M. H. A. Banna et al., *Application of Artificial Intelligence in Predicting Earthquakes: State-of-the-Art and Future Challenges,* IEEE Access, vol. 8. (2020)

[2] Hochreiter Sepp, Schmidhuber Jürgen, *Long Short-term Memory,* Neural computation, 9, 1735-80. (1997)

[3] Kamer Yavor, Nandan Shyam, Ouillon G., Hiemer Stefan, Sornette Didier, *Democratizing earthquake predictability research: introducing the RichterX platform,* The European Physical Journal Special Topics, 230, 451-471. (2021)

[4] Fabius Otto, Amersfoort Joost, Kingma Diederik, *Variational Recurrent Auto-Encoders,* (2014)