CITY
UNIVERSITY OF LONDON
EST 1894

Using Neural Networks to Predict Natural Disasters

IN3062

Authors

William Dibble - William.Dibble@city.ac.uk

Umer Faisal - Umer.Faisal@city.ac.uk

Inthirajith Sinniah - Inthirajith.Sinniah@city.ac.uk

Kaloyan Dimitrov - Kaloyan.Dimitrov@city.ac.uk

1. Introduction and Dataset

We are using artificial intelligence (AI) to predict the likelihood of future natural disasters, including the type of disaster, location, magnitude, potential number of casualties, and number of people affected. To do this, we are using a dataset from EM-DAT *(Dincer, 2021)*, which contains information on over 22,000 natural disasters from 1900 to the present. The data shows the following about each natural disaster: The type of disaster that it is, including floods, droughts, earthquakes, and epidemics. It shows the start and end dates of each disaster, as well as the number of people affected by them including the number of deaths and injuries. It shows the locations of the disasters with some being specific, and some more general. And finally, it shows the damage that was done as a result of the disaster in terms of monetary value. The dataset was compiled from information provided by insurance companies, United Nations agencies, non-governmental organisations, research groups, and news agencies.

We chose this dataset as if reliable predictions can be made, this information could be shared to potentially help countries to predict when and where there may be a natural disaster happening. By providing advance warning of potential natural disasters, the results could help people prepare for and potentially evacuate areas that may be affected, which could help save lives and reduce the overall impact of the disaster on communities.

In addition, our predictive model could be used to inform decision-making and resource allocation. For example, emergency management agencies could use our predictions to deploy resources, such as rescue teams and supplies, to areas that are likely to be affected by a natural disaster before it even happens, which would help ensure that people have access to the assistance they need during and after the disaster.

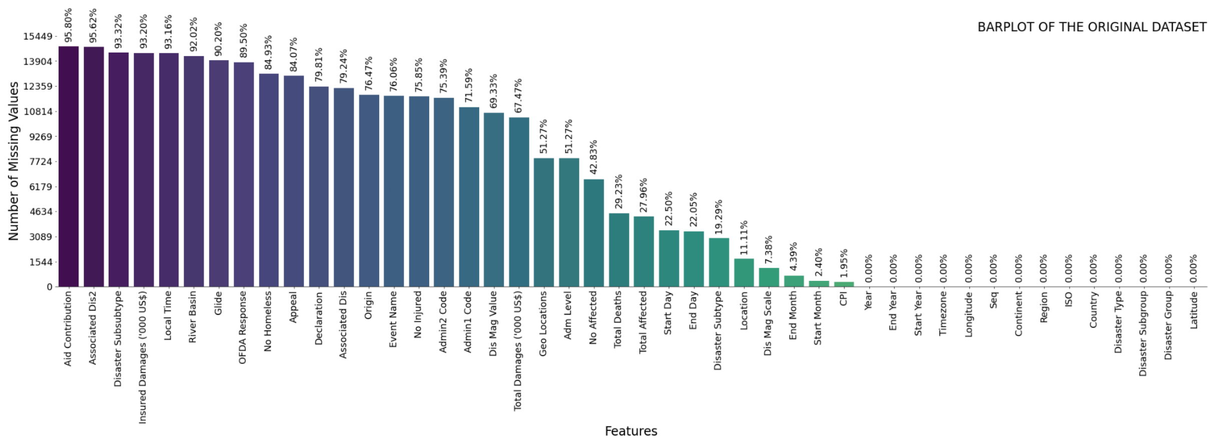
Overall, we believe that our predictive model has the potential to make a significant positive impact on society by helping people prepare for and mitigate the effects of natural disasters.

2. Data Cleaning and Preprocessing

Below is a summary of the code developed for the preprocessing of the dataset. We took these steps to optimise the data for the neural networks and thus should yield the most accurate and reliable results.

The first step in this process was to load the dataset using the pandas library and view the first few rows of the data using the head() method. This allowed us to get a sense of the structure and content of the data.

Next, we calculated the percentage of missing values in each column and plotted the results in a bar plot, with the missing values sorted in descending order. This provided a visual representation of which columns had the most values missing and helped identify any columns that needed to be dropped or have their missing values interpolated. By identifying columns with a high percentage of missing values, we ensured that these columns do not negatively impact the performance of the prediction models or neural network. We then proceeded to drop several columns that are deemed unnecessary or useless for the analysis, which in turn reduced the complexity of the dataset which should improve the efficiency of the neural networks.



We then replaced any missing values in the Day and Month columns using interpolation, which is a method of estimating the value of a point within a range of known data points. In this case, we used the interpolate() method of the pandas library to estimate the missing values in the Day and Month columns based on the known values in these columns. By replacing the missing values using interpolation, we can ensure that the neural networks are not negatively impacted by the presence of missing values.

After this, the validity of the dates in the dataset was checked by ensuring that the Day and Month values are within the correct range for each month, by creating a dictionary that maps each month to the number of days in that month, and looping through each row in the data frame to check the value of the Day column against the number of days in the corresponding month. This ensured that the dates in the dataset are all valid and that the neural networks that we developed were not negatively impacted by any invalid dates.

We then retrieved each location's longitude and latitude information using the geolocator library. The geolocator library allowed the code to retrieve this information by requesting it from a geocoding API. We first created an instance of the Nominatim class from the geolocator library, which is used to make the geocoding requests, and then looped through each row in the data frame to retrieve the longitude and latitude for the location of each disaster using the geocode() method. This took in the location as an argument and returned a Location object that contains the longitude and latitude of the location, which was then loaded into the existing longitude and latitude columns. By including the longitude and latitude information in the dataset, we could potentially improve the accuracy of the neural networks by allowing them to consider the specific geographic location of each disaster.

After filling in the longitude and latitudes, we calculated the z-score of the numerical columns 'Total Deaths', 'No Injured', and 'Total Damages'. The z-score, also known as the standard score, is a measure of how many standard deviations a value is from the mean. Calculating the z-score allowed us to standardise the values in these columns so that they are on the same scale and could be more easily compared. We did this to allow the network to easily identify patterns and relationships within the numerical data.

After calculating the z-scores, we created a new column called 'Total Impact' by summing the values in the 'Total Deaths', 'No Injured', and 'Total Damages' columns. The 'Total Impact' column provided a single value that represented the overall impact of a disaster in a particular location. This could be useful when working with the neural networks, as it allows it to more easily compare the impact of different disasters.

Overall, the pre-processing methods described in this report performed a number of important tasks that helped to clean and prepare the dataset for use in the neural networks. By identifying and removing unnecessary columns, replacing missing values using interpolation, checking the validity of the dates, filling in longitude and latitude information, and standardising numerical columns using z-scores, we could ensure that the dataset is in a form that is ready for use in the networks.

3. Modelling Methods and Results

Neural Network Model 1

Method

The following method explains how the method we used for neural network model 1 was formed, and why we chose to implement each technique to accurately predict natural disasters. We started by importing various libraries, including pandas, NumPy, scikit-learn, and TensorFlow. These libraries are needed to perform the various tasks required to build and train the neural network, such as data manipulation, numerical operations, and building and training the neural network itself. We then loaded the data from a CSV file to a DataFrame and removed certain columns with missing values. Removing columns with missing values ensures that the model has access to complete and accurate data, which is essential for making accurate predictions.

As we began working on the neural network, we encountered numerous errors due to incorrect data encoding and selection of input data and target features. It took multiple iterations and trial and error to resolve these issues and find the most accurate model. The attached modelling files demonstrate the various approaches we took in attempting to achieve the most accurate prediction result.

Dummy data were then created for some categorical columns using the get_dummies function, and the data was scaled using min-max scaling. Creating dummy data allows the model to represent categorical data in a way that it can understand to make reliable predictions, and scaling the data ensures that all the data is on a consistent scale, which can help improve the model's performance by allowing it to better process the data. We then selected the data and target features, and the data was split into training and test sets to allow the model to learn from the training data and be evaluated on the test data. Doing so means it's a more realistic representation of how the model will perform on new data.

The input data was then scaled using scikit-learn's StandardScaler function, which is necessary as this can help improve the model's performance by ensuring that the data is on a consistent scale, which as mentioned earlier, makes it easier for the model to process. We defined the model architecture using the Sequential class and added four dense layers to the model. This defines the structure and layout of the neural network, which determines how the model will process and use the data to make predictions.

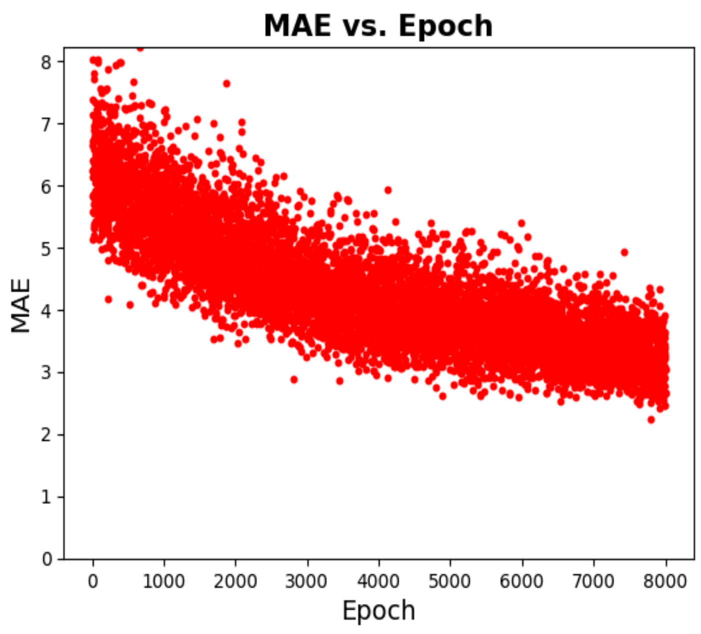
The model was then compiled with a specified loss function and optimiser, and the metric 'mae' (mean absolute error) is used to evaluate performance. This involved specifying the loss function and optimiser, which determined how the model's weights and biases were updated during the training and how well the model is doing. The model was trained using the fit function and the number of epochs and batch size were specified as arguments., which processed the training data and updated the model's weights and biases based on the loss function and optimiser. After training, the model's performance was then evaluated on the test data and the model was trained for an additional 8000 epochs with the verbose argument set to 0. Evaluating the model's performance allowed us to determine how well it was doing and identify any areas where it may have been underperforming.

The mean absolute error values from the training history were then extracted and plotted to visualise how the model's performance had improved over time and identified any patterns or trends in the data.

Results

In the graph below, we can see the mean absolute error over time compared to the number of epochs that have passed, in which it appears that the MAE starts high and decreases as the number of epochs increases. This is expected, as the model is being trained to improve its performance on the training data over time. As the model is trained, it adjusts its internal weights and biases to better fit the training data, which should result in a decrease in the MAE.

It is possible for the MAE to continue decreasing indefinitely as the number of epochs increases. This may not necessarily be a good thing though, as the model may be overfitting the training data and becoming too complex. When it comes to neural networks, it is important to find the right balance between model complexity and performance, which may involve stopping the training process at a certain point and selecting the best model based on the performance on the test set.



Unfortunately, we didn't have enough time to continue the network from here, but one potential next step in using these results to predict future natural disasters would be to use the trained neural network model to generate predictions given a start date, disaster type, and location (latitude and longitude). This could be done by preprocessing the input data in the same way that was done in the original code (e.g., scaling, creating dummy data, etc.), and then feeding this processed data into the model using the predict function.

It may have also been useful to fine-tune the model by adjusting the hyperparameters (e.g., the number of layers, the number of neurons in each layer, the learning rate, etc.) and training the model on additional data, which could further improve the model's performance and lead to more accurate predictions. Another potential approach could have been to evaluate the model's performance on new data using different evaluation metrics (e.g., root mean squared error, accuracy) to get a better understanding of the model's strengths and weaknesses to identify any areas where the model may be performing poorly, and allow for targeted improvements to be made.

Overall, there are many potential ways to use these results to predict future natural disasters, and this neural network could be used to create accurate predictions if tuned properly.

Neural Network Model 2

Method

Similarly, to our first neural network, we started by loading the dataset from the CSV file produced by preprocessing the original dataset to a DataFrame and formatting the start date column. We then used the pd.to_datetime() function to convert the 'Start Date' column to datetime objects. The LabelEncoder class from sklearn was then used to encode the 'Disaster Type' column into numerical values (i.e., 0 to 6 in this case).

Next, we extracted the year, month, and day values from the 'Start Date' column and added them as separate columns in the data frame. This allowed us to use these values as our target features when training the model.

Once we had prepared the data, we again selected the data and target features that we would use for our model in the neural network. The data included a variety of categorical and numerical columns much like our first model, such as 'Disaster Subgroup', 'Disaster Subtype', and 'Country', while the target features included the start year, start month, start day, longitude, latitude, and disaster type. The categorical features were then encoded and the numerical features were scaled using the pd.get_dummies() function and a MinMaxScaler object, respectively. This ensured that all of the data were on the same scale, which can improve the model's performance.

We then split the pre-processed data into a training set and a test set using the train_test_split() function from sklearn, which allowed us to evaluate the model's performance on unseen data. After this, we built the model using the Sequential class from tensorflow.keras. The model then uses a sequential neural network with two hidden layers with the first player being 64 neurons taking the input data dimension of the X.shape[1] with a kernel regulator of l2 with a regularisation strength of 0.001.The second layer was also built with 64 neurons and the same kernel regulariser as the first layer with both having batch normalisation and dropout layers with a dropout rate of 0.5. Finally, the output layer was built with 6 neurons and the overall model was optimised using an Adam optimizer with a learning rate of 0.001. It was compiled using the mean squared error as the loss function instead of the custom-defined one and metrics such as MAE and MSE for evaluation.

After building the model, we compiled it using the compile() method, the Adam optimiser, and the MeanSquaredError loss function. We specified that we wanted to track the mean absolute error and mean squared error as metrics during training. We then trained the model using the fit() method and specified the number of epochs and the batch size. The validation data was passed in so that we could track the model's performance on the test set during training. This allowed us to monitor the model's progress and ensure that it was not overfitting or underfitting the data. Once the model was trained, we used it to make predictions on the test set and evaluated its performance by computing the mean absolute error and root mean squared error as they are both common metrics for evaluating the performance of regression models.

There were several advantages to using the methods and techniques implemented in this code. One advantage was that we were able to encode the categorical features and scale the numerical features, which would improve the model's performance. Another advantage was that we used the train_test_split() function to split the data into a training set and test set, which allowed us to evaluate the model's performance on unseen data. We also used regularisation to help prevent overfitting and selected appropriate loss and optimization functions for our model.

However, there are also some disadvantages to consider. One disadvantage is that the model's performance may be sensitive to the choice of hyperparameters such as the number of neurons in each layer and the learning rate. Another disadvantage is that the model may not generalise well to new data if the patterns in the training data are not representative of the patterns in the real world.

In addition, the code could have been improved by using more advanced techniques such as deep learning and ensemble methods. These methods could further be used to help better capture the complex relationships between the date and the target features, as well as improve the accuracy of predictions. Furthermore, it is important to consider any factors that may influence natural disasters such as climate change and human activities or even weather data. By incorporating these factors into the model, one can possibly gain a better understanding of how natural disasters occur and how they can be predicted in the future.

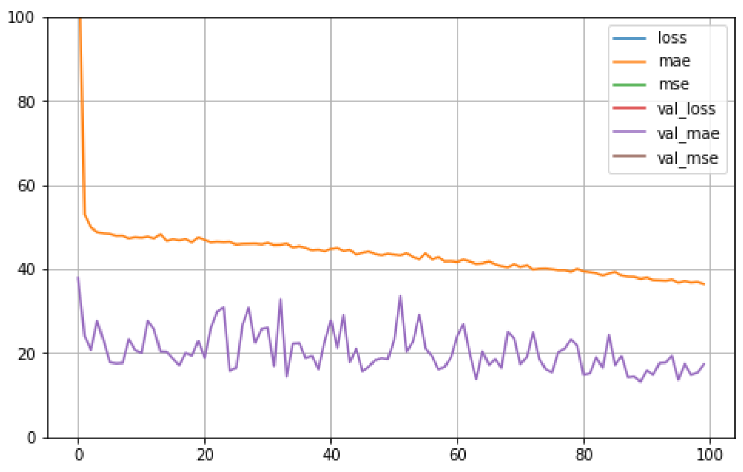
Results

The mean absolute error (MAE) is a measure of the difference between the predicted values and the true values. In the graph below, the MAE starts very high around 2500 and quickly drops down to nearly 0. This rapid decrease in the MAE is likely due to the fact that the model was able to learn patterns in the data and make more accurate predictions as it was trained for more epochs.

The fact that the MAE quickly decreases and then levels off around 20 is generally good but also indicates that the data might be under-fitted within the model. However, it is important to note that the MAE was not the only metric used to evaluate the model's performance, as other metrics such as the root mean squared error (RMSE) and the model's performance on the test set need to also be considered.

One potential issue with this graph is that the MAE may be too low, which could indicate that the model is overly confident in its predictions and may not generalise well to new data. This is known as overfitting and can occur when the model has learned patterns in the training data that do not hold in the real world.

Overall, the rapid decrease in the MAE and the levelling off around 20 is usually a good sign, but it is important to consider other metrics and evaluate the model's generalisation performance to ensure that it is not overfitting the data.


100
80
60
40
20
0
0
20
40
60
80
100
loss
mae
mse
val_loss
val_mae
val_mse

4. Evaluation and Conclusion

Based on the results of our neural networks, it appears that they were able to learn some patterns in the data and make predictions of some natural disaster features. The mean absolute error (MAE) decreased over time in both of our networks, indicating that they were both able to fit the data well. But, Neural Network 1 may be overfitting and Neural Network 2 may be underfitting their data. It is also important to note that the MAE should not be the only metric used to evaluate the model's performance. We could have used other metrics such as the root mean squared error (RMSE) to measure the model's performance on the test set.

One potential limitation of these networks is that they may not generalise well to new data. This could be due to a variety of factors such as a lack of representative data in the training set or overfitting to the training data. To mitigate this risk, we could have used more regularisation techniques such as dropout, because by randomly setting neurons to zero, dropout helps to ensure that the model does not rely too heavily on any one neuron or group of neurons and instead relies on a combination of neurons to make predictions. This can reduce the risk of overfitting and improve the model's ability to make accurate predictions on unseen data, but may have resulted in underfitting.

In conclusion, the neural network implemented in this code was able to predict some future natural disaster features using a dataset from 1900 to 2021. However, further evaluation is needed to ensure that the model is able to generalise well to new data and to identify any potential limitations or areas for improvement.

References

EM-DAT, (2022) *The International Disaster Database* (no date) *public.emdat.be*. EM-DAT. Available at: https://public.emdat.be/ (Accessed: December 21, 2022) (Downloaded: December 21, 2022).

Dincer, B. (2021) *All Natural Disasters 1900-2021 / EOSDIS*, *Kaggle*. Available at: https://www.kaggle.com/datasets/brsdincer/all-natural-disasters-19002021-eosdis (Accessed: November 24, 2022) (Downloaded: November 25, 2022).