Predicting success of global terrorist activities

Trisha Jani

Abstract

The purpose of this project is to predict the success of a terrorist attack given a set of input features. The Global Terrorism Database (GTD) is an open-source database including information on over 170,000 terrorist events around the world from 1970 through 2016. The National Consortium for the Study of Terrorism and Responses to Terrorism (START) has maintained the database and created several visualizations and performed statistical analysis with the data. However, the data has not extensively been used for prediction purposes. In this project, we use the GTD to predict the success of a terrorist attack using deep learning techniques. We explore three different fully connected neural network architectures, and find that our strongest model achieves over 91% accuracy. Our results suggest that it is indeed possible to train effective neural networks to predict the success of a terrorist attack.

I. Introduction & Motivations

Since the turn of the century, we have seen an uptick in the number of terrorist activities on a global scale, as seen in Figure 1. The aim of this project is to better understand terrorist activities around the world and use deep learning techniques to predict the success of attacks.

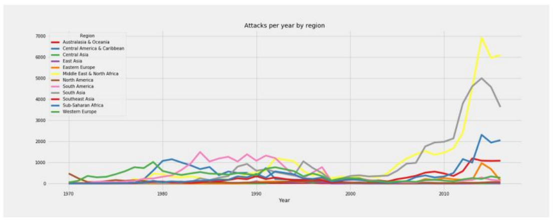

Figure 1: *There has been a rise in the number of terrorist attacks since the turn of the century.*

For the project, we use the Global Terrorism Database, an open-source database including information on terrorist attacks around the world from 1970 through 2016. The database describes over 170,000 incidents, both domestic and international, via a set of over 130 descriptive features. The database is maintained by the National Consortium for the Study of Terrorism and Responses to Terrorism (START), which has its headquarters at the University of Maryland.

While there exist a lot of visualizations and statistical analysis based off this dataset, no predictive analysis via machine learning or deep learning have been applied to better understand the data. For this reason, we thought it would be an interesting break this barrier and attack this problem with deep learning.

II. Dataset & Features

i. Global Terrorism Database

The initial uncleaned dataset consists of 170,350 terrorist incidents described by a list of 135 features. These features are both numerical and categorical and are divided into 8 basic groups: event data, incident information, incident location, attack information, weapon information, target/victim information, perpetrator information, and casualties/consequences. Most importantly, each incident is also labeled as successful or unsuccessful. We build a model to try to predict the success of an attack given a subset of these features.

ii. Exploratory data analysis

Before building our model, we decided to first perform basic exploratory data analysis to gain a high level understanding of the our features. We highlight a few of our finding and plots below.

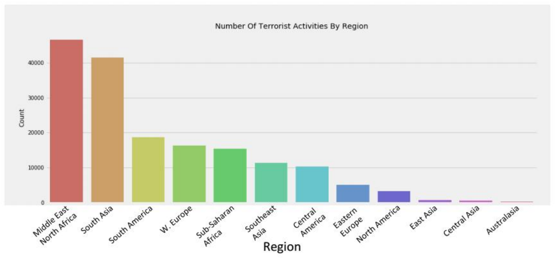

Figure 2: *The number of terrorist attacks differs by region.*

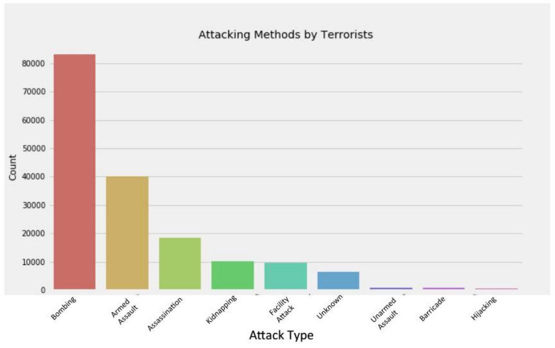

Figure 3: *Terrorist prefer bombing and explosions when attacking.*

We first looked at the geographical spread of attacks and note that region is an important factor in determining the number of terrorist attacks. As Figure 2 shows, there are significantly more attacks in the Middle East, North Africa, South Asia, and South America than other regions in the world.

Then, we also examined the most common methods of attack. From Figure 3, We see that bombings and explosions are the preferred method of attack, while unarmed assault, barricades, and hijacking are less common.

By performing this exploratory data analysis, we gained a better understanding of the dataset and decided to build our model using a set of the following 9 features: `country, region, attack type, target type, weapon type, group name, suicide, multiple, hostage`. Although we could have used more features, these 9 are a good representation of information that would be known in a real-world scenario. Since these features were categorical, we used a one-hot representation of each of them in our neural network model, where each input is of dimension 3512.

After cleaning our dataset, we had a total of 151,254 incidents. We reserved 10% for our test set (15,126 examples). Of the remaining 136,128 examples, we set aside 10% for our development set (13,613 examples), meaning the remaining 122,515 examples were used for training.

We note that we have a class imbalance within our dataset. In particular, of our 151,254, nearly 90% have class label '1' (successful attack). For this reason, we are careful to ensure that roughly 90% of our training, development, and test sets each also have examples of class '1'.

III. Methods

i. Baseline Model

Because we have a significant class imbalance in our data, our baseline model is simply a predictor that outputs '1'. This simple model achieves nearly 90% accuracy on the train, dev, and test sets. Therefore, moving forward with out more complex neural network models, we want to ensure that we have at least 90% accuracy. If not, our very simply model that always predicts '1' does a better job, and there is no need to use such a complex and computationally expensive model.

ii. Fully Connected NN with two hidden layers

Our first neural network architecture was a fully connected network with 2 hidden layers (Figure 4). We wanted to see how a relatively small network would perform on our data, and therefore chose to make our first hidden layer have 12 units and our second layer have 6 units.

Since we are trying to solve a binary classification problem, we evaluated our model on the basis of binary cross-entropy loss, defined as follows: $Loss = \frac{-1}{N}\sum_{i=1}^{N}[y_i log(\hat{y}_i) + (1 - y_i)log(1 - \hat{y}_i)]$ where N is the number of attacks, y_i denotes the success of an attack (0 is unsuccessful and 1 if successful), and $\hat{y}_i$ is the predicted label for an attack (0 is unsuccessful and 1 if successful).

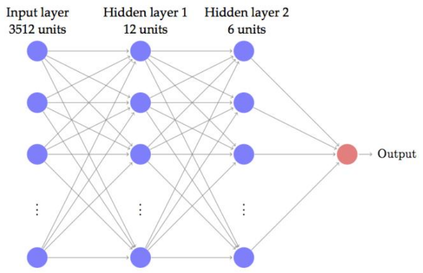

Figure 4: *Our first model has an input layers, 2 hidden layers, and an output layer*

We had to tune several hyperparameters to get reasonable model performance. We configured our model with the Adam optimizer ($\beta_1 = 0.9$, $\beta_2 = 0.999$) and a mini-batch size of 50. After adjusting our learning rate and activation functions, we settled on $\alpha = 0.001$ and opted to use ReLU activation for our hidden layers and sigmoid activation for our output layer.

After training for 50 epochs (Figure 5), we had a model that achieved 92.34% accuracy on our training set. However, the dev set accuracy was 90.58%, which suggests that our model overfit to our training data. For our upcoming models, we decided it would be necessary to implement regularization to prevent overfit.

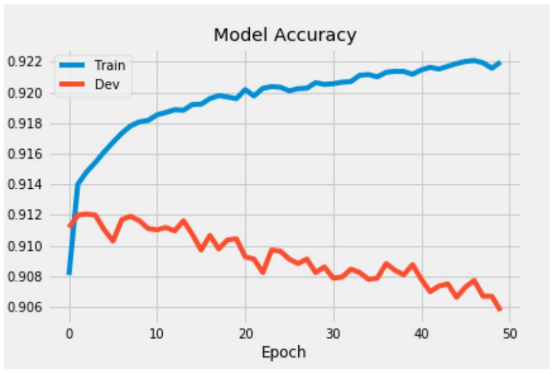

Figure 5: *Training Model 1 for 50 epochs*

iii. Fully Connected NN with three hidden layers

Our next neural network model was a fully connected architecture with 3 hidden layers, as shown in Figure 6. Our first hidden layer had 100 units, second hidden layer had 50 units, and third hidden layer had 20. Again, we used the ReLU activation for our hidden layers and sigmoid activation for our output layer. Our final model used the Adam optimizer with a learning rate $\alpha = 0.01$.

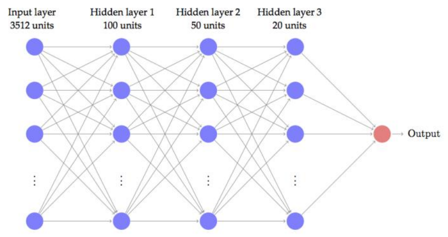

Figure 6: *Our second model has an input layers, 3 hidden layers, and an output layer*

Since our data has a class imbalance, we adjusted our loss to penalize false positives by using a cross weighted loss with parameter α that we manually define: $Loss = \frac{-1}{N}\sum_{i=1}^{N}[\delta y_i log(\hat{y}_i) + (1 - \delta)(1 - y_i)log(1 - \hat{y}_i)]$. Since we have a 9:1 class imbalance, we chose $\delta = 0.1$ This further penalizes false positive errors.

To combat the problem of overfitting, we decided to implement dropout regularization. After rounds of tuning dropout fractions rates for each layer, we found that the rates (0.5, 0.2, 0.2) worked well.

After 50 epochs of training (Figure 7), our model achieved 91.98% accuracy on our training set and 91.18% accuracy on the dev set. This suggests that our model did not overfit to our training data and generalized pretty well.

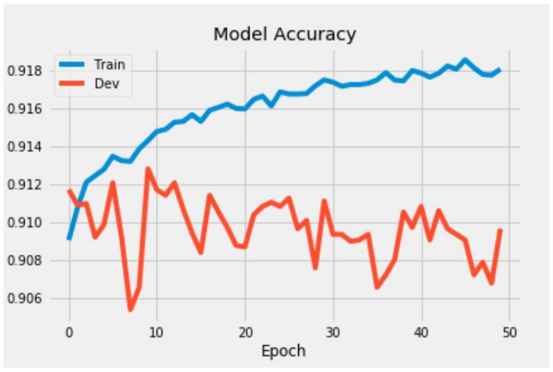

Figure 7: *Training Model 2 for 50 epochs*

iv. Fully Connected NN with four hidden layers

Finally, we wanted to test out an even larger fully connected neural network to see if our performance would increase. Our last neural network model was a fully connected architecture with 4 hidden layers, as shown in Figure 8. Our first hidden layer had 50 units, second hidden layer had 20 units, third hidden layer had 10 units, and fourth layer had 5 units. Again, we used the ReLU activation for our hidden layers and sigmoid activation for our output layer. Our final model used the Adam optimizer with a learning rate $\alpha = 0.01$.

Given the strong performance of our weighted loss function, we decided to use it again for this model. Again, we used $\delta = 0.10$. Similarly, since we saw the benefit of dropout regularization to preventing overfit, we decided to also implement dropout for this model. After rounds of tuning dropout fractions rates for each layer, we found that the rates (0.5, 0.2, 0.2, 0.2) worked well.

After 50 epochs of training (Figure 9), our model achieved 90.83% accuracy on our training set and 90.54% accuracy on the dev set. We notice that this model has lower performance the both previous models on the training set. However, the gap between the training set accuracy and the dev set accuracy is pretty small, suggesting that our model did not overfit to our training data and generalized pretty well. In our to better improve this model, we would likely need more data or more features to increase training accuracy.

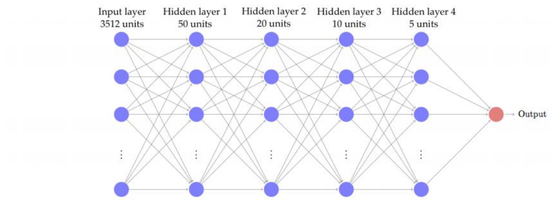

Figure 8: *Our first model has an input layers, 4 hidden layers, and an output layer*

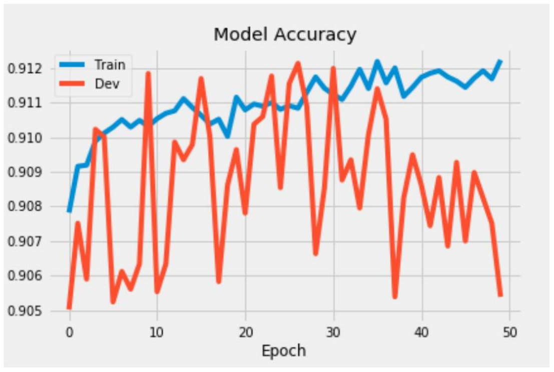

Figure 9: *Training Model 3 for 50 epochs*

IV. Conclusions

Table 1 shows the performance of our three models on the training, dev, and test sets. To summarize, we see that Model 1, the simplest neural network with the least parameters, has the lowest training error, but does not generalize that well. When we increase our model complexity, adjust our loss function, and add regularization to build Model 2, we see it has the highest performance on the test set. This means the model generalizes well. Finally, Model 3, our most complex model, generalizes well but has the lowest accuracy on the training set.

Table 1: *Comparing model accuracy*

Model type	Train	Dev	Test
Model 1	92.34%	90.58%	90.85%
Model 2	91.98%	91.18%	91.17%
Model 3	90.83%	90.54%	90.69 %

Table 2 compares the performance of the three models on statistical tests on the test set. Since our classes are imbalanced, we specifically look at the recision, recall, and F1 scores. All three models have a high recall, which means they have a low false negative rate. All three models have pretty high precision (greater than 0.9), which means the false positive rate is pretty low. However, the precision is lower than the recall, which suggests that the models can still be improved to reduce the false positive rate. All three models have similar F1 scores.

Table 2: *Statistical Performance on test set*

Model type	Precision	Recall	F1 Score
Model 1	0.921	0.980	0.950
Model 2	0.916	0.994	0.953
Model 3	0.905	0.999	0.950

There are several avenues for future work on this problem. First, we could explore with even deeper architectures to see if prediction accuracy increases. Alongside that, we could also try more regularization techniques (such as adjusting the cost function) and spend more time further tuning dropout fraction rate parameters. Next, we could also trying including more features from the original dataset. We originally chose to build a model with 9 hand chosen features because they are interpretable and actionable in a real-world scenario. However, it would be interesting to see how much the model improves by introducing additional features.

Lastly, we note that we can further decrease our false positive rate by constructing a more "real-world" loss function that takes into account the social, financial, and even psychological loss incurred by a false alarm for a terrorist attack. This data may be hard to come by, but if any estimates exist, it can be used to form a loss function that better mirrors real world circumstances.

References

[1] National Consortium for the Study of Terrorism and Responses to Terrorism (START). (2017). Global Terrorism Database Codebook. Retrieved from https://www.start.umd.edu/gtd
[2] Prabhakar Misra, Raul Garcia-Sanchez and Daniel Casimir. "Development and Optimization of Machine Learning Algorithms and Models of Relevance to START Databases," Report to the Office of University Programs, Science and Technology Directorate, U.S. Department of Homeland Security. College Park, MD: START, 2016.
[3] Martin Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Rafal Jozefowicz, Yangqing Jia, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mane, Mike Schuster, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viegas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.