Predicting Motor Vehicle Accidents using Machine Learning Techniques

A Technical Research Paper Submitted to the Department of Engineering and Society
Presented to the Faculty of the School of Engineering and Applied Science
University of Virginia – Charlottesville, Virginia

In the Partial Fulfillment of the Requirements for the Degree
Bachelor of Science, School of Engineering

By

Akanksha Alok

Spring, 2020

On my honor as a University student, I have neither given nor received unauthorized aid on this assignment as defined by the Honor Guidelines for Thesis-Related Assignments.

Signed:_____ Akanksha Alok ____ Date________04/14/2020____________
Akanksha Alok

Approved: _______Yuan Tian _________________ Date______04/16/2020_____________
Yuan Tian, Assistant Professor of Computer Science

Predicting Motor Vehicle Accidents using Machine Learning Techniques

Akanksha Alok, Yuan Tian
University of Virginia
aa8dp@virginia.edu, yt2e@virginia.edu

Abstract

Many times, threatening vehicle accidents go unnoticed for long periods of time – sometimes too late to save the harmed passengers or involved victims in the accident. Vehicles go off-roads in secluded areas, and are out of sight or reach of bystanders. Sometimes, the victims are in a critical condition where they are unable to call for help or dial for 911. In fact, 66% of deaths caused by motor vehicle accidents could have been prevented if the patient had received medical treatment or assistance earlier [1].

In an effort to reduce this time to alert authorities and receive medical aid, I have created a novel approach using different Machine Learning techniques and methodologies that will be able to detect impending vehicle crashes, using sensor data collected from an iOS device situated in the vehicle. While this project primarily focused on predicting mishaps with motor vehicles, this system can be applied to other predictive modeling cases, where Time-Series datasets are available for the situation.

The first component of the system was a Recurrent Neural Network (RNN), which was used with the Time-Series data to predict the future states of the motor vehicles' sensor readings. Once the future states of the vehicle were predicted using this neural network, I assessed whether these states were indicative of an accident, using an Autoencoder neural network. Since the dataset that I used for training contained samples of vehicles that were not involved in crashes (clean data) and had no data regarding vehicle accidents, I chose to use the Autoencoder instead of the standard Binary Classifier. Therefore, the Autoencoder trained on the patterns of the clean data, and was able to pinpoint the vehicle accidents as anomalies. Finding the relevant features from the datasets, creating these two neural network models, linking the two separate models, and optimizing them for the given datasets are covered in detail in this paper.

1 Related Work

GPS Spoofing and the hacking of sensors and cameras on transportation vehicles have become a prevalent problem in our society, as expressed in detail by Humphreys [2]. He describes the large-scale operations that have been accomplished through GPS spoofing, such as the event in which Iran misguided several US ships and aircrafts into unknown territory, as well as his own attempt of successfully misguiding the White Rose cruise liner one kilometer astray without the crew noticing. In the case of autonomous vehicles, several researchers have already perfected an algorithm to trick current GPS navigation systems used in cars, deflecting vehicles thousands of meters from their intended destinations without the knowledge of the drivers [3].

Any mishap with the gadgets used in an autonomous vehicle has the potential to put many lives in danger, whether it is the passengers in the autonomous car, passengers in the other cars on the road, or even pedestrians. Researchers have tackled this problem using the Kalman Filter algorithm, so that even when a sensor, camera or navigation system of the car is hacked, the car will be able to tune out the exterior noise and continue functioning normally. By training a neural network on input data from various cameras, sensors, and the GPS navigation system of the autonomous vehicle, it constructs a probabilistic model that is able to predict the next state of the vehicle based on the current state. When there is any external noise or extraneous data coming in from any of the hacked sensors, the algorithm will be able to tune out this noise and be able to continue on the rightful path determined by the probabilistic model [4].

In a similar manner, I wanted to use sensor data to predict the next state of the vehicle using a probabilistic model, as well. However, the project detailed above mainly pertained to autonomous vehicles and the sensors that those vehicles used for navigating the roads, and I wanted to focus on a problem with motor vehicles that we currently face and which continues to be a problem even in the future. While I also use a neural network for the predictive model using the sensor data, I have added other Machine Learning components to the system to be able to predict whether the next state is indicative of a vehicle accident or crash.

2 Data Manipulation

2.1 Data Collection

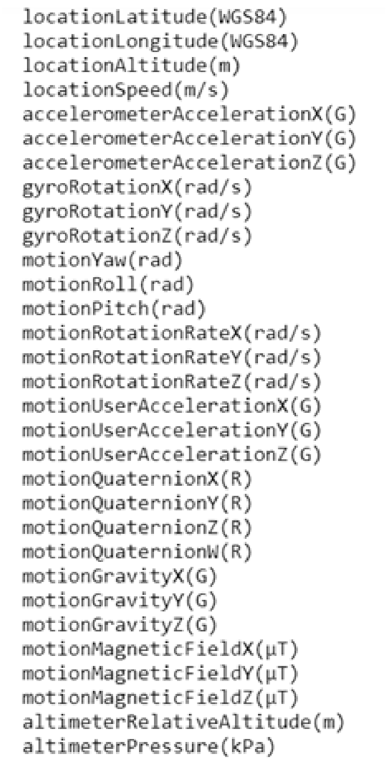
One of the biggest challenges of this project was finding the appropriate dataset for the Machine Learning models. I was originally planning to use the Waze Dataset, which provided continuous data regarding a subset of vehicles for a few months. However, the most important aspect of Time-Series data is that it needs to have consistent time-stamps that are spaced out with equal intervals in between each one. The Waze dataset, despite being a reputable source, provided one timestamp for each vehicle throughout the months and showed no progression for my Recurrent Neural Network to train on.

In order to work around this problem, I gathered our own data, by driving around the Charlottesville area, with an iOS device collecting all of the sensor data in the interior of the car. The iOS device tracked the same features with its sensors every few milliseconds, providing a progressive Time-Series dataset during the car rides. We drove around the same areas to keep the location data consistent, but at different times of the day to introduce more variance and to mimic real-life scenarios. After going on approximately two car rides at different times of the day on two seperate days, we were able to collect enough data for my project.

2.2 Feature Engineering

The iOS tracking system collected 72 different features using their sensors, many of which had no relevance to motor vehicle accidents. From this group of 72 features, I picked out the 36 features that had correlations to the project from the dataset.

From this set of 36 features, I then limited the dataset down to a subset of 31 features using a Pearson Correlation Matrix. The correlation matrix reported the covariance between the different features, and features which had high correlation or shared overall distribution patterns would be repetitive and add no additional value to the model. Any features that had more than 95% correlation with at least three other features in the dataset were removed, and I was left with a subset of 31 features for my models.



```
locationLatitude(WGS84)
locationLongitude(WGS84)
locationAltitude(m)
locationSpeed(m/s)
accelerometerAccelerationX(G)
accelerometerAccelerationY(G)
accelerometerAccelerationZ(G)
gyroRotationX(rad/s)
gyroRotationY(rad/s)
gyroRotationZ(rad/s)
motionYaw(rad)
motionRoll(rad)
motionPitch(rad)
motionRotationRateX(rad/s)
motionRotationRateY(rad/s)
motionRotationRateZ(rad/s)
motionUserAccelerationX(G)
motionUserAccelerationY(G)
motionUserAccelerationZ(G)
motionQuaternionX(R)
motionQuaternionY(R)
motionQuaternionZ(R)
motionQuaternionW(R)
motionGravityX(G)
motionGravityY(G)
motionGravityZ(G)
motionMagneticFieldX(µT)
motionMagneticFieldY(µT)
motionMagneticFieldZ(µT)
altimeterRelativeAltitude(m)
altimeterPressure(kPa)
```

Figure 1: Complete set of Features

Since the feature pool was already small, I did not have to apply other dimensionality reduction techniques, such as Principal Component Analysis (PCA) and Locally Linear Embedding (LLE) that I had used with the previous Waze dataset.

3 Recurrent Neural Network

3.1 Background

As humans, our thoughts persist for a period of time (this period of time can vary depending on memory and attention span!) and then fade away into oblivion. We never start our thought process from the moment that we first opened our eyes as a newborn, but instead think back to the relevant periods of time to make an informed decision. Even as you read this paper, you are comprehending each word based on the previous words in this sentence. We keep the context for a period of time and when it becomes irrelevant, we move on. For example, if I am telling you a story about a boy named Charlie, and in the next phrase, mention a "He", your brain will know that I am referring to Charlie. The moment that a new noun, Sheila, appears in the story, our mind gets rid of Charlie and associates the next pronoun with Sheila. Our thoughts have persistence until a certain point, and to some extent, we choose which information from the past is relevant to store [5].

Unfortunately, traditional neural networks are unable to do this, and is a major shortcoming for use with time-series data where we want to use information from the past to inform later decisions. Recurrent Neural Networks (RNNs) address this problem. In simplest terms, they are networks with loops within them that allow information to persist. The predictive component of my model, which provided the future states of the vehicle, involved a Recurrent Neural Network.

In Figure 2, the Neural Network component, A, takes in the input x, and outputs a value, h, at time t. The loop allows for the information to be passed from one step to the next.

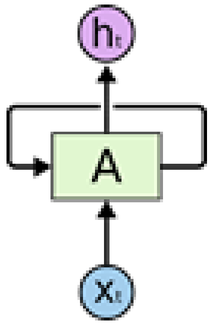


Figure 2: Looped RNN structure

When unrolled (as seen in Figure 3), the RNN closely resembles a normal neural network. Essentially, an RNN can be viewed as multiple copies of the same network, with each iteration providing information to its successor.

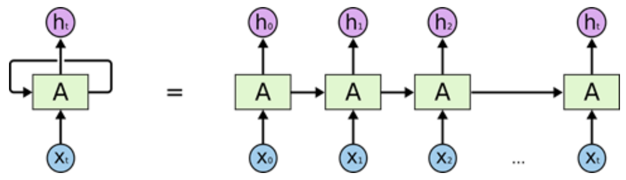


Figure 3: Unrolled RNN structure

The iterative nature of the RNN lends itself to be used in sequences or chains of events, and in the case of my project, a sequence of time, where each subscript in the diagram above represents a timestamp [6].

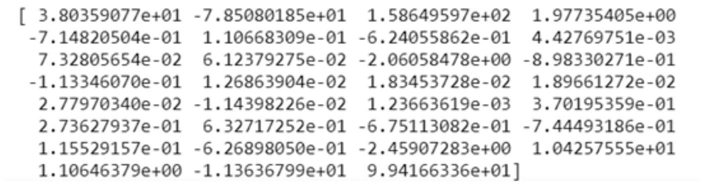
3.2 Long Short Term Memory Models

Most of the time, traditional RNNs suffer from not being able to pick up on long-term trends and are easily skewed by the most recent data that it is presented with. For my project, I utilized a special type of RNN, known as Long Short Term Memory Models (LSTM) that are capable of learning long-term dependencies and patterns within the datasets. Like explained above, the LSTM also follows an iterative progression, but each repeating Neural Network Module has a special structure of four neural network layers that interact in a special way [6].

For my project, I implemented a layered LSTM model to make predictions for 153 timestamps in the future for the given vehicle that I had collected data on. The LSTM was able to predict values for all of these features in parallel, despite these features being completely unique.

Given the array of features that I was using (as detailed in Figure 1), the LSTM was able to make a prediction for each of these features simultaneously for each timestamp. The last prediction at the 153rd timestamp (or approximately 2.5 minutes after the first timestamp) is shown in Figure 5, where each prediction corresponds to the same feature at the same index in Figure 1.

Figure 5: The 153rd prediction of features using LSTM



```
[ 3.80359077e+01 -7.85080185e+01  1.58649597e+02  1.97735405e+00
 -7.14820504e-01  1.10668309e-01 -6.24055862e-01  4.42769751e-03
  7.32805654e-02  6.12379275e-02 -2.06058478e+00 -8.98330271e-01
 -1.13346070e-01  1.26863904e-02  1.83453728e-02  1.89661272e-02
  2.77970340e-02 -1.14398226e-02  1.23663619e-03  3.70195359e-01
  2.73627937e-01  6.32717252e-01 -6.75113082e-01 -7.44493186e-01
  1.15529157e-01 -6.26898050e-01 -2.45907283e+00  1.04257555e+01
  1.10646379e+00 -1.13636799e+01  9.94166336e+01]
```

3.3 Optimizations

The main problem that I was encountering was that the RNN was not able to update itself quickly enough to report predicted values that pertained to the most recent timestamps. It was predicting values that appeared most frequently in the dataset and values that were at earlier timestamps.

Since I had been working with the Waze dataset that was considerably larger than the current dataset, I had previously created a complicated model to cater to the large size of the dataset. This complicated model usually creates a tendency for the neural network to closely adapt to the smaller training dataset, and I had wrongly predicted that the model was overfitting to the training data (matching the pattern of the training dataset so closely that it is unable to generalize to the other data points). As expected, the regularization methods that I applied to the model to combat overfitting were further amplifying the problem and the model was still not predicting the correct values. The hyperparameters that I changed were very sensitive, so I decided that I should first diagnose and confirm the model's problem before making any changes based on my intuition**.**

Essentially, I had to figure out if the model was overfitting or underfitting to the data set. The only way to do this is to plot out the validation set error and training set error (the validation set is a small subset of the training set). These errors are the errors that the model makes when training on the training set (training loss) and the errors that the partially trained model makes on a small testing set (validation set). When I started doing this, I realized that the model was actually underfitting. The validation loss was much higher than the training set error, and the validation error showed no signs of decreasing over time (as shown in Figure 6).

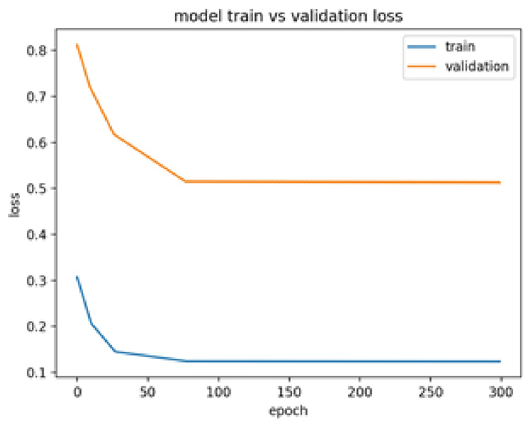


Figure 6: An Underfitting Model Performance

A "good-fit" model (shown in Figure 7) is one where the validation loss steadily decreases to the point of stabilization and intersects (or comes close to) the training loss at some point. The training loss should also stabilize to the point where it no longer decreases.

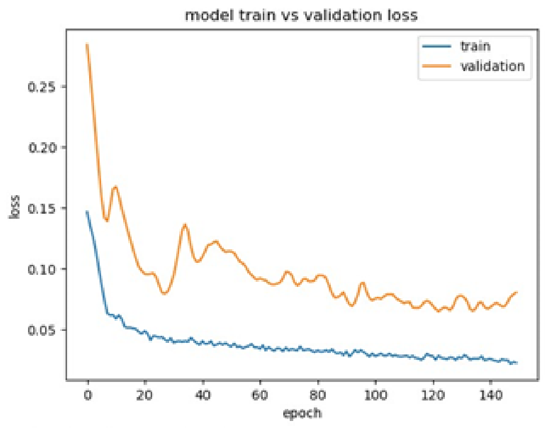


Figure 7: A Good Fit Model Performance

I used these graphs as a metric to assess how the model was performing, and developed my own methodology to find a stopping point within the training period to minimize the difference between the model's training and validation loss.

I tried a series of different experiments to combat the problem of underfitting, such as increasing the number of training epochs, adding neurons to each layer, changing the Dropout percentage (of nodes) in each layer, and adding an additional layer in the Neural Network node, until I finally found the methodology that worked: reducing the number of LSTM layers. Usually, having too many layers causes the model to overfit to the data. However, in this case, the number of layers was obscuring the model from learning the pattern of the dataset and was making it too sensitive, and the model was therefore underfitting. I was finally able to find a good-fit model for the dataset.

Another problem that I ran into was that the model was reporting back NAN values for the training loss, and this was caused by the Exploding Gradient problem. Essentially, exploding gradients are a problem where large error gradients accumulate and result in very large updates to neural network model weights during training (the weights of the neural network are changed during a back-propagation step). This has the effect of the model being unstable and unable to learn from the training data. The easy fix to this problem was introducing the idea of Gradient Clipping to the model, where the values of the error gradient are checked against a threshold value and then are clipped or set to that threshold value if the error gradient exceeds the threshold. This prevents the weights from being heavily updated, and allows for the updates to happen in increments.

5 Autoencoder

5.1 Background

Autoencoders are a type of feed-forward neural network whose purpose is to serve as an identity function, where the output of the neural network tries to resemble the input. They serve as a dimensionality reduction technique, where it compresses the input into a lower-dimensionality vector, and tries to rebuild the output from this compressed representation.

The Autoencoder consists of three components: Encoder, Code, and the Decoder. The Encoder, which is essentially a simple Neural Network, parses the input given and produces a Code, as well. This Code is a formula that the Decoder will use to parse the compressed version of the input to reproduce the original input. The Decoder is a mirror image of the Encoder, and resembles the Encoder's structure. The goal of the Autoencoder as a whole is to produce an output that is as close to the input as possible.

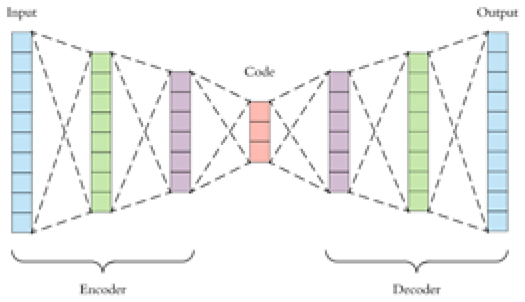


Figure 8: Autoencoder structure

The reconstruction error is the difference between the output vector of the Autoencoder and the compressed vector form of the input. Throughout the training process, the Autoencoder tries to minimize this error using backpropagation, where several hyperparameters of the model are slightly modified [7].

5.2 Implementation

One of the major challenges that I faced during my project was that the data that I had only contained information about a car that was not involved in any mishaps or accidents. Therefore, I could not create a standard Binary Classifier that would be able to group vehicles as either safe or involved in an accident.

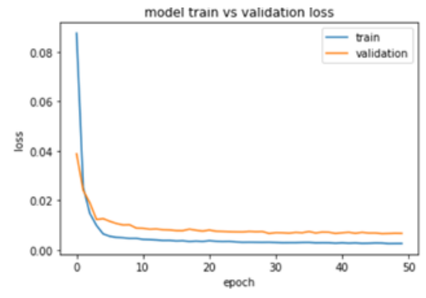
In order to work around this problem, I utilized the capability of the Autoencoder to train on the patterns and trends of the clean data, and to ultimately learn how to precisely reproduce the most frequent characteristics of the observations of the clean data. Once the Autoencoder is trained and it is faced with a vehicle undergoing an accident, the model should worsen its reconstruction performance and would yield a high reconstruction error, labeling that data point as an anomaly.

Using a small sample set of "anomaly" data, where I greatly varied the values for each feature by at least two standard deviations, I was able to find the exact reconstruction error threshold for marking a sample as an anomaly. I ran the Autoencoder on a subset of the clean training data, and took note of the reconstruction error, and then ran it on the anomaly dataset. I was able to find a clean reconstruction error threshold of 0.01, that did not yield for any false negatives or false positives (high precision) among the datasets that I had. Essentially, a reconstruction error that exceeded the threshold of 0.01 would be marked as an anomaly, or as a car accident.

6 Results

6.1 Predicting the Next State of Vehicle

Using the LSTM model, I was able to predict up to 153 future timestamps from the last given timestamp. Since the data given to me was spaced by millisecond time-intervals, the model was able to predict the future states up to 2.5 minutes after the first timestamp. As shown in Figure 9, the validation and training losses indicate that the model was a perfect fit for the dataset.



```
172/172 [==============================] - 0s 1ms/sample - loss: 0.0067
0.006663311749350193
```

Figure 9: Performance of Prediction Model

Fortunately, I had two separate datasets that were spaced out by 2 minutes – one dataset collected vehicle information from 08:02:25 to 08:21:47, and the other dataset started collecting information at 08:23:34 until 08:31:25 the same day. To fully test the capabilities of the RNN, I trained the neural network on the first dataset, and used its predictions to test its accuracy on the second dataset and to compensate for the two-minute time gap between the datasets.

The 153rd prediction at timestamp at 08:26:00 was compared to the actual value in the second dataset and yielded a very small mean square error (MSE). I also graphed the predicted and actual values over the 08:23:34 to the 08:24:00 timestamps. The graphs to the right clearly indicate that the RNN was able to track the general trend and provide a highly accurate prediction for each feature, where the red trend line indicates the actual values and blue trend line is the predicted values.

In order to check if the autoencoder was functioning, these 153 future timestamps were run through the autoencoder, as well, and each one had a reconstruction error of about 0.005 (less than the 0.01 threshold), which correctly gauged that these timestamps were not indicative of a vehicle accident or crash.

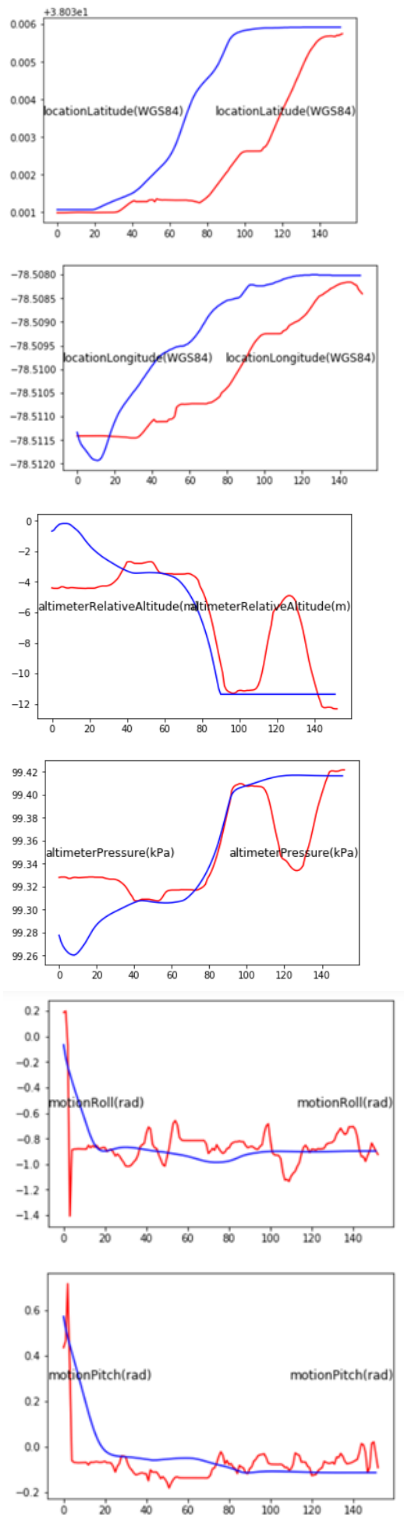


Figure 10: Predicted versus Actual Feature Values

6 Future Improvements

One of the biggest drawbacks to this project was that we could not find the datasets that best matched our needs. In fact, I was barely able to get enough data from gathering it from an iOS device myself, which could have introduced some human error. However, the system that I created can easily be adapted to other datasets, and if I were to locate the appropriate datasets in the future, we can potentially have a system that can predict every impending car crash.

I also only had access to clean data, where there were no motor vehicle accidents. Acquiring this dataset would serve as a good testing set to make sure that the threshold enforced for the autoencoder to detect car crashes is correct and functioning.

While the Recurrent Neural Network model is highly accurate and uses state-of-the-art calculations and techniques for predictions, it is unable to make accurate long-term predictions, as it tends to plateau with no new data being fed to it. In order to enable long term predictions, the model needs to be deployed so that it can receive a constant flow of updated data so that it can adjust itself and make predictions that are suited to the current data. Deploying the model to enable constant data flow would be something useful and worthwhile in the future, especially if we want to keep using the powerful RNN model.

7 Conclusion

In this project, I sought to understand the significance of the problem of delayed response to motor vehicle accidents. In an attempt to mitigate this problem, I tried to create a system that would predict whether a vehicle was involved in an accident in advance.

I created a Machine Learning algorithm consisting of a LSTM Recurrent Neural Network and Autoencoder to predict future states of the sensors in the vehicle and assessed whether these sensor values were indicative of a car crash.

While the data collection portion proved to be a major problem, I was able to work around it by collecting and creating my own dataset for training and testing my system. I was able to create a system that did perform accurately for these given datasets.

However, with more accurate datasets and optimized RNN structure, the system can be improved upon and can be more accurate in long-term predictions.

8 Acknowledgements

I would like to acknowledge the support I received from Professor Yuan Tian. It is in large part due to her dedication and guidance that I was able to complete this project. Thank you.

References

[1] CDC. (2016, December 5). Cdc winnable battles progress report. Centers for Disease Control and Prevention. https://www.cdc.gov/winnablebattles/report/index.html

[2] Humphreys, M. L. P. and T. E. (2016, July 29). Protecting gps from spoofers is critical to the future of navigation. Retrieved March 30, 2019, from IEEE Spectrum: Technology, Engineering, and Science News website: https://spectrum.ieee.org/telecom/security/protecting-gps-from-spoofers-is-critical-to-the-future-of-navigation

[3] Zeng, K. C., Shu, Y., Liu, S., Dou, Y., & Yang, Y. (2017). A practical gps location spoofing attack in road navigation scenario. Proceedings of the 18th International Workshop on Mobile Computing Systems and Applications - HotMobile '17, 85–90. https://doi.org/10.1145/3032970.3032983

[4] Li, K., Hu, B., Chang, L., & Li, Y. (2015). Robust square-root cubature Kalman filter based on Huber's M-estimation methodology. Proceedings of the Institution of Mechanical Engineers, Part G: Journal of Aerospace Engineering, 229(7), 1236–1245. https://doi.org/10.1177/0954410014548698

[5] Britz, D. (2015, September 17). Recurrent neural networks tutorial, part 1 – introduction to rnns. WildML. http://www.wildml.com/2015/09/recurrent-neural-networks-tutorial-part-1-introduction-to-rnns/

[6] Understanding LSTM Networks—Colah's blog. (n.d.). Retrieved April 2, 2020, from https://colah.github.io/posts/2015-08-Understanding-LSTMs/

[7] Unsupervised feature learning and deep learning tutorial. (n.d.). Retrieved April 2, 2020, from http://ufldl.stanford.edu/tutorial/unsupervised/Autoencoders/