Predictive analytics for vehicle collisions through the use of a neural network

Anthony Asilo
Artificial Intelligence
Computer Science Department
Georgia State University

***Abstract* – Vehicles, whether speaking of cars, trucks, airplanes, drones, or even ships, serve a single purpose: to provide a means of transportation from one point to another. Vehicles are generally constructed while keeping safety in mind, providing some sort of feature to minimize harm in the event of a collision. However, as humans are the ones controlling these vehicles most of the time, there is always possibility for error to be involved. With this in mind, this report will explain how to combat collisions by using a neural network to potentially detect predict collisions before they occur, to assist in object avoidance.**

Keywords and phrases – reinforcement learning, neural-network, autonomous vehicle, IoT, data stream, real time processing, location-based services, geofence, proof of location, azimuth.

I. Introduction

Approximately 1.35 million people worldwide die each year from a motor vehicle accident [1], and some 90 percent of motor vehicle accidents are caused by human error [2]. As the world population continues to grow, more vehicles will be utilized, which inevitably will lead to more human error. Autonomous vehicles have gained more traction in the automobile industry recently, such as Tesla's Models or Google's Waymo project, and such autonomous vehicles use deep neural networks and deep learning algorithms for path planning with the help of cameras and ultrasonic sensors and radar.

Looking at the vehicles created by Tesla for example, the autopilot and full self-driving capability functions work with features such as traffic-aware cruise control, autosteer, auto lane change, auto park, and traffic and stop sign control, just to name a few. Although these are great features that are not to be taken lightly, there is still a lot of progress that must happen, and a breakthrough needs to occur by means of reducing and avoiding collisions.

In a perfect world, all vehicles, whether they be road, sea, or air, could essentially have an IoT device that utilizes your geographical coordinates for location-based services. This would promote efficiency of these vehicles when it comes to path planning, minimizing traffic, and reducing collisions by implementing V2V (Vehicle to Vehicle) communications.

II. Background

In this section, a few of the keywords that have been previously mentioned will be described in a bit more detail.

A location-based service is a service that considers one's geographic location to provide a user with accurate and factual information. This can be used for locating friends or alerting one of potential traffic ahead [11]. A proof of location is a certificate that gives someone's attendance at a specific geolocation at a specific time. Using a PoL with vehicles would help provide accurate information on where surrounding vehicles are located [12]. In order to actually distribute information from one vehicle to another for a PoL, each vehicle would need to be connected with a device to share information with, which is where the Internet of Things (IoT) comes into play. Data would be streamed and sent in real-time when a vehicle is detected in other vehicles geofence, which can be thought of a perimeter that defines the area or bounds of a location-based service.

Once we have some information regarding each vehicle's location, decision making processes will occur to determine the next action to take. Plenty of things could be considered, including but not limited to distance or time for vehicle to reach the intersection of potential collision, size of vehicle and size of road, etc., to avoid or continue to the goal destination. These variables would be in addition to path planning algorithms and would be implemented through reinforcement learning with neural networks.

Reinforcement learning refers to how an intelligent agent will perform an action in an environment to maximize reward, and it does not need labeled input and output pairs present but will focus on balancing exploration and exploitation of knowledge. When implemented using a neural network, reinforcement learning does not have to specifically design a state space [13].

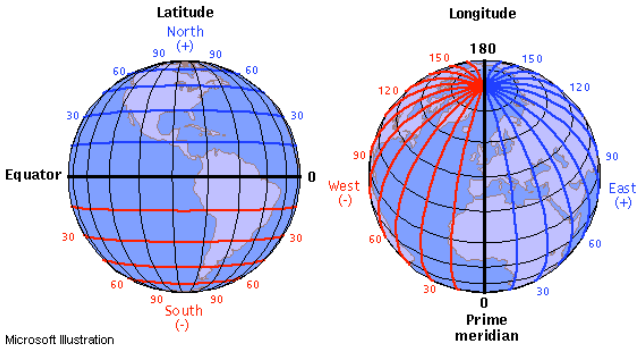
The objective is to learn and predict if a collision will occur if a host vehicle detects a guest vehicle within the radius of the vehicle's geofence through the use of a deep neural network. The data fed as inputs to the NN will be based on geographical and vehicular data (velocity, coordinates, etc.) to predict and perform necessary actions.

III. Methodology

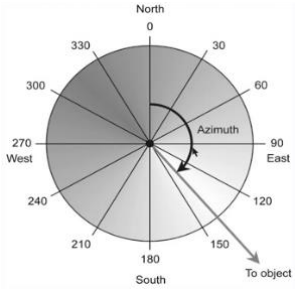
In order to accurately depict what this projects purpose is, a program will be created to simulate what will be calculated when a vehicle detects another vehicle in its geofence. To start off, a vehicle class will be created, where each vehicle can be declared and then initialized with a unique hash code, coordinates, and velocity.

Coordinates include a latitude and a longitude, which are a grid map system that instead of being straight lines on a flat surface, they encircle the earth horizontally or vertically, respectively. Latitudinal lines show how much north or south of the equator a place is located, and the equator is our starting point for measuring latitude, which is 0 degrees latitude. Latitude increases to 90 degrees as you move towards the north or south pole. Longitudinal lines show how east or west

of the Prime Meridian a place is located, which is a universal line that is marked as 0 degrees longitude. Longitude increases to 180 degrees as you move east or west [3].



Velocity represents the speed and azimuth in which the vehicle would be moving. Speed will be measured in kilometers per hour, while the azimuth, which is a bearing staring from north that goes clockwise from 0 to 360, will be measured in degrees. This will allow consistency when measuring what direction a vehicle is moving.



This figure illustrates an azimuth that measures approximately 140 degrees, which is in the general Southeast direction

For the purpose of generating data and understanding the relationship between coordinates, velocities, and collisions, a Host vehicle will be declared and initialized, and stay constant throughout the program. A second vehicle, which will be named Guest, will be spawned a random azimuth away from the Host vehicle at a distance that matches the radius of double the geofence, which has been set to 100 meters.

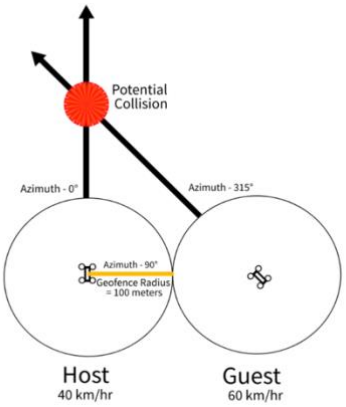
Step 1 - Declare and initialize Host vehicle at specific coordinate and as well as an azimuth and speed. (In this case it is 0° and 40km/hr)

Step 2 - Declare and initialize Guest vehicle at coordinate based on a random azimuth from the Host vehicle and 2 times the distance of the geofence radius. This is because if all vehicles have a geofence of 100 meters, the maximum distance they can be from each other before being undetected is 200 meters. Then initialize an azimuth for the Guest vehicle that will intersect with the Host Azimuth with some trigonometric functions and initialize a random speed.

Step 3 - Calculate the coordinates of the possible intersection where a collision may occur.

Step 4 - Calculate the time it takes for each vehicle to get to the intersection. If the time it takes for each vehicles is within a time relative to their speed, a collision has occured. Other wise, they missed eachother.

Step 5 – Feed local data into training data set and repeat steps 1 through 5



To understand how the data is being generation, it is necessary to a few methods that assist coordinate calculations.

The method inverseCoords takes four parameters, two of which are the start latuitude and longitude, and two of which are the end latitude and longitude[5]. This method returns a distance and azimuth between two coordinates, relative to the first coordinate. This is possible by the Haversine formula [4]:

Haversine formula:
$a = \sin^2(\Delta\varphi/2) + \cos\varphi_1 \cdot \cos\varphi_2 \cdot \sin^2(\Delta\lambda/2)$
$c = 2 \cdot \text{atan2}(\sqrt{a}, \sqrt{(1-a)})$
$d = R \cdot c$

where φ is latitude, λ is longitude, R is earth's radius (mean radius = 6,371km); note that angles need to be in radians to pass to trig functions!

The second function is terminalCoords, which returns a end coordinate's latitude and longitude, given a start coordinate's latitude, longitude, azimuth, and distance [4].

Formula:
$\varphi_2 = \text{asin}(\sin\varphi_1 \cdot \cos\delta + \cos\varphi_1 \cdot \sin\delta \cdot \cos\theta)$
$\lambda_2 = \lambda_1 + \text{atan2}(\sin\theta \cdot \sin\delta \cdot \cos\varphi_1, \cos\delta - \sin\varphi_1 \cdot \sin\varphi_2)$

where φ is latitude, λ is longitude, θ is the bearing (clockwise from north), δ is the angular distance d/R; d being the distance travelled, R the earth's radius

Additionally, There are two functions, cartesian() and LatLon(). The method cartesian() takes a latitude and longitude as input and converts the coordinates to a Cartesian <x, y, z> vector point. Assume that Phi is latitude and Theta is longitude[6]:

$$x = cos(\theta)cos(\phi)$$
$$y = sin(\theta)cos(\phi)$$
$$z = sin(\phi)$$

LatLon() converts a Cartesian point to a latitude and longitudinal coordinate:

$$\theta = arctan2(y, x)$$
$$\phi = arctan2(z, \sqrt{x^2 + y^2})$$

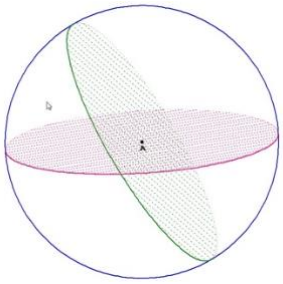
Next, the GCI function, which stands for Great Circle Intersection, takes 8 parameters: a start and an end latitude/longitude for the Host vehicle, and a start and end latitude/longitude for the Guest vehicle. Each coordinate is converted to a cartesian point, and then each vehicle's arc paths has their start and end coordinate multiplied as a cross product. The cartesian intersection is derived from the following calculations [7]:

$$h = \frac{dc - fa}{ea - db}, g = \frac{-bh - c}{a}, k = \sqrt{\frac{r^2}{g^2 + h^2 + 1}}, where$$
$$< a, b, c > is\ Host_{start} \times Host_{End},$$
$$< d, e, f > is\ Guest_{start} \times Guest_{End},$$
$$and\ r = Radius\ of\ Earth \approx 6371.137m$$

The two great circles intersect at the following points:

$$(gk, hk, k), (-gk, -hk, -k)$$

Because these are Cartesian, we use the method LatLon() to convert them back to latitudinal and longitudinal coordinates. It needs to be determined next which point is the correct POI, better known as a point of intersection.



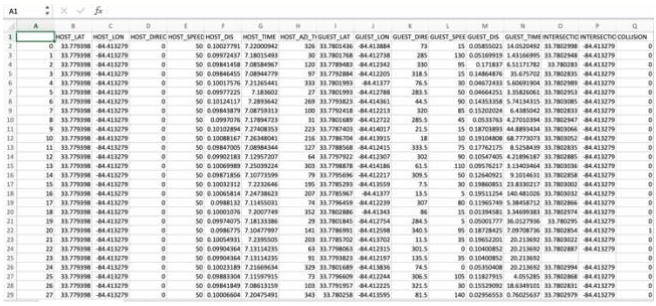
The POI needed is found in our last method, checkIfLies(), which takes two points on the sphere and

determines if these intersection points lies within arc segments by doing an angle test, by normalizing the dot product [8].

$$\theta = \arccos\left(\frac{a \cdot b}{||a||||b||}\right)$$

In the main method, we simply utilize all of these previously defined functions to randomly generate a Guest vehicle around a pre-initialized Host vehicle. Because the coordinates for each of the vehicles as well as the intersection, the speed of the vehicles, and the azimuth are known, we can easily calculate the time it takes for each vehicle to reach the calculated intersection. If the time it takes for both vehicles to get to the intersection is relatively close, which for the purpose of this data, is set to .035 seconds, we can assume that the vehicles have indeed collided.

Now that there is a working coordinate system, a large dataset is generated that contains information regarding each vehicles latitude, longitude, speed, azimuth, distance to intersection, time to intersection, intersection coordinates, and whether a collision occurred or not.

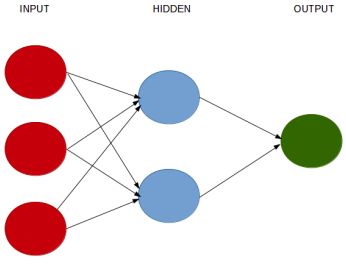


HOST_LAT	HOST_LON	HOST_DIREC	HOST_SPEED	HOST_DIS	HOST_TIME	HOST_AZI_TO	GUEST_LAT	GUEST_LON	GUEST_DIRE	GUEST_SPEE	GUEST_DIS	GUEST_TIME	INTERSECTIO	INTERSECTIO	COLLISION
0	33.779398	-84.413279	0	50	0.10027791	7.22000942	326	33.7801436	-84.413884	73	15	0.05855021	14.052049	33.7802998	-84.413279	0
1	33.779398	-84.413279	0	50	0.09972437	7.18015493	30	33.7801768	-84.412738	285	130	0.05169919	1.43166995	33.7802948	-84.413279	0
2	33.779398	-84.413279	0	50	0.09841458	7.08584967	120	33.7789483	-84.412342	330	95	0.171837	6.51171782	33.780283	-84.413279	0
3	33.779398	-84.413279	0	50	0.09846455	7.08944779	97	33.7792884	-84.412205	318.5	15	0.14864876	35.675702	33.7802835	-84.413279	0
4	33.779398	-84.413279	0	50	0.10017576	7.21265441	333	33.7801993	-84.41377	76.5	30	0.04672433	5.60691904	33.7802989	-84.413279	0
5	33.779398	-84.413279	0	50	0.09977225	7.183602	27	33.7801993	-84.412788	283.5	50	0.04664251	1.35820061	33.7802953	-84.413279	0
6	33.779398	-84.413279	0	50	0.10124117	7.2893642	269	33.7793823	-84.414361	44.5	90	0.14353358	5.74134315	33.7803085	-84.413279	0
7	33.779398	-84.413279	0	50	0.09843879	7.08759313	100	33.7792418	-84.412213	320	85	0.15202024	6.4385042	33.7802833	-84.413279	0
8	33.779398	-84.413279	0	50	0.0997076	7.17894723	31	33.7801689	-84.412722	285.5	45	0.0533763	4.27010394	33.7802947	-84.413279	0
9	33.779398	-84.413279	0	50	0.10102894	7.27408353	223	33.7787403	-84.414017	21.5	15	0.18703893	44.8893434	33.7803066	-84.413279	0
10	33.779398	-84.413279	0	50	0.10088167	7.26348041	216	33.7786704	-84.413915	18	10	0.19104808	68.7773073	33.7803052	-84.413279	0
11	33.779398	-84.413279	0	50	0.09847005	7.08984344	127	33.7788568	-84.412415	333.5	75	0.17762175	8.5258439	33.7802835	-84.413279	0
12	33.779398	-84.413279	0	50	0.09902183	7.12957207	64	33.7797922	-84.412307	302	90	0.10547405	4.21896187	33.7802885	-84.413279	0
13	33.779398	-84.413279	0	50	0.10069989	7.25039224	303	33.7798878	-84.414186	61.5	110	0.09576217	3.13402464	33.7803036	-84.413279	0
14	33.779398	-84.413279	0	50	0.09871856	7.10773599	79	33.7795696	-84.412217	309.5	50	0.12640921	5.1014631	33.7802858	-84.413279	0
15	33.779398	-84.413279	0	50	0.10032312	7.2232646	195	33.7795293	-84.413559	7.5	30	0.19860852	23.8330217	33.7803002	-84.413279	0
16	33.779398	-84.413279	0	50	0.10065814	7.24738623	207	33.7785967	-84.41377	13.5	5	0.19512254	140.481026	33.7803032	-84.413279	0
17	33.779398	-84.413279	0	50	0.0988132	7.11455031	74	33.7796459	-84.412239	307	80	0.11961749	5.38458712	33.7802866	-84.413279	0
18	33.779398	-84.413279	0	50	0.10001076	7.2007749	352	33.7802886	-84.41343	86	15	0.01394581	3.34699383	33.7802974	-84.413279	0
19	33.779398	-84.413279	0	50	0.09974075	7.18193386	29	33.7801845	-84.412754	284.5	5	0.05001777	36.0127936	33.780295	-84.413279	0
20	33.779398	-84.413279	0	50	0.0986775	7.10477997	141	33.7786991	-84.412598	340.5	95	0.18728425	7.09708736	33.7802854	-84.413279	1
21	33.779398	-84.413279	0	50	0.10054931	7.2395505	203	33.7785702	-84.413702	11.5	35	0.19612201	20.213692	33.7803022	-84.413279	0
22	33.779398	-84.413279	0	50	0.09904364	7.13114235	63	33.7798063	-84.412315	301.5	0	0.01400852	20.213692	33.7802887	-84.413279	0
23	33.779398	-84.413279	0	50	0.09904364	7.13114235	91	33.779383	-84.412197	135.5	35	0.10400852	20.213692			
24	33.779398	-84.413279	0	50	0.10023189	7.21669634	329	33.7801689	-84.413836	74.5	0	0.05350408	20.213692	33.7802994	-84.413279	0
25	33.779398	-84.413279	0	50	0.09883304	7.11587915	73	33.7796609	-84.412244	306.5	105	0.11827915	4.055285	33.7802868	-84.413279	0
26	33.779398	-84.413279	0	50	0.09841849	7.08613159	103	33.7791957	-84.412225	321.5	30	0.15529092	18.6349101	33.7802831	-84.413279	0
27	33.779398	-84.413279	0	50	0.10006604	7.20475491	343	33.780258	-84.413595	81.5	140	0.02956553	0.76025637	33.7802979	-84.413279	0

* Note: Empty Intersection latitude and longitude cells represent the fact that there was no intersection

IV. Implementation

A neural network is a way of making computations and predictions based on existing data. The neural network being utilized will be implemented using *Keras*, an API developed by Google to run on top of the machine learning dependency TensorFlow. In this example, the neural network built in Keras will help solve a multivariate regression problem, where the dependent variable y (which is whether the collision occurred or not), is dependent on the x variables (which are the velocities of and speed of the guest vehicle, the distance and time to intersection for both of the vehicles, the time to intersection, and the azimuth between the host vehicle and the guest vehicle). A neural network consists of input, output, and hidden layers. Input layers take input based on the generated data, output layers consist of the output that is derived from the input and the hidden layers, and hidden layers use backpropagation to optimize the weights of the input variables in order to improve the predictive analysis [9].



A simple neural network with 3 inputs, 2 hidden layers and one output

A linear activation function in the Keras library will help create the regression-based network. Next is to import all libraries and import the dataset. In order to properly feed the data to our neural network, the data previously generated must be preprocessed. This includes getting rid of any row where values are NaN. Additionally, a few more columns (Unnamed: 0, COLLISION, HOST_LAT, HOST_LON, GUEST_LAT, GUEST_LON, HOST_DIRECTION, HOST_SPEED, INTERSECTION_LAT, INTERSECTION_LON) are dropped as they stay mostly constant in the generation at this time and may have little to no effect on the network.

```
data = pd.read_csv("./Vehicle-Avoidance-AI/vehicle_collision_data.csv")
print(data.shape)
data.head()
```

```
(146556, 17)
```

	HOST_DIRECTION	HOST_SPEED	HOST_DIS	HOST_TIME	HOST_AZI_TO_GUEST	GUEST_LAT	GUEST_LON	GUEST_DIRECTI
0		50	0.100278	7.220009	326	33.780144	-84.413884	73.0
0		50	0.099724	7.180155	30	33.780177	-84.412738	285.0
0		50	0.098415	7.085850	120	33.778948	-84.412342	330.0
0		50	0.098465	7.089448	97	33.779288	-84.412205	318.5
0		50	0.100176	7.212654	333	33.780199	-84.413770	76.5

```
data = data.dropna()
print(data.shape)
data.head()
```

```
(131443, 17)
```

ST_LON	GUEST_DIRECTION	GUEST_SPEED	GUEST_DIS	GUEST_TIME	INTERSECTION_LAT	INTERSECTION_LON	COLLISION
3884	73.0	15	0.058550	14.052049	33.780300	-84.413279	0
2738	285.0	130	0.051699	1.431670	33.780295	-84.413279	0
2342	330.0	95	0.171837	6.511718	33.780283	-84.413279	0
2205	318.5	15	0.148649	35.675702	33.780283	-84.413279	0
3770	76.5	30	0.046724	5.606919	33.780299	-84.413279	0

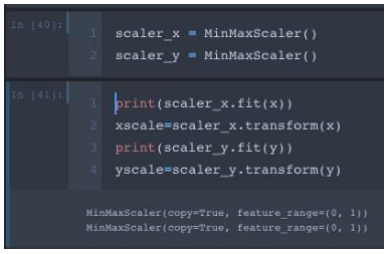
```
x = data.drop(['Unnamed: 0','COLLISION',"HOST_LAT","HOST_LON","GUEST_LAT","GUEST_LON","HOST_DIR
print(x.shape)
print(x)
```

```
(131443, 7)
[[1.00277909e-01 7.22009942e+00 3.26000000e+02 ... 1.50000000e+01
  5.85502859e-02 1.40520492e+01]
 [9.97243744e-02 7.18015493e+00 3.00000000e+01 ... 1.30000000e+02
  5.16931928e-02 1.43166995e+00]
 [9.84145788e-02 7.08584967e+00 1.20000000e+02 ... 9.50000000e+01
  1.71836998e-01 6.51171782e+00]
 ...
 [1.00187172e-01 7.21347635e+00 1.90000000e+02 ... 2.50000000e+01
  1.99424854e-01 2.87171789e+01]
 [9.87926387e-02 7.11314199e+00 7.50000000e+01 ... 5.50000000e+01
  1.21026618e-01 7.92174228e+01]
 [9.86969860e-02 7.10617579e+00 1.42000000e+02 ... 9.50000000e+01
  1.87874100e-01 7.11943956e+00]]
```

```
y = data['COLLISION'].to_numpy()
y = np.reshape(y,(-1,1))
print(y.shape)
print(y)
```

```
(131443, 1)
[[0]
 [0]
 [0]
 ...
 [0]
 [0]
 [1]]
```

Next, it is necessary to normalize the variables so that the neural network can interpret them properly, and this is possible with Keras MinMaxScalar() function[9].



```
scaler_x = MinMaxScaler()
scaler_y = MinMaxScaler()
```

```
print(scaler_x.fit(x))
xscale=scaler_x.transform(x)
print(scaler_y.fit(y))
yscale=scaler_y.transform(y)
```

```
MinMaxScaler(copy=True, feature_range=(0, 1))
MinMaxScaler(copy=True, feature_range=(0, 1))
```

After the variables are normalized the data is split into training and test data so that the neural network can be trained.

```
X_train, X_test, y_train, y_test = train_test_split(xscale, yscale)
```

There are 7 inputs that were previously defined as well as two hidden layers of 6 and 4 neurons, and then linear activation is used to generate the output. The mean-squared-error and mean-absolute-error are loss functions in that will measure the neural networks accuracy of predicting the test

data. 80 percent of the training data will be used for the training model whereas the other 20 percent will be used to test the training data. Generally, from the output, it is seen that the greater amount of runs accounts for the lower amount of MSE and MAE, which shows that the model is improving in regard to accuracy [10].

```
In [43]:
model = Sequential()
model.add(Dense(6, input_dim=7, kernel_initializer='normal', activation='relu'))
model.add(Dense(4, activation='relu'))
model.add(Dense(1, activation='linear'))
model.summary()

Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_9 (Dense)              (None, 6)                 48
_________________________________________________________________
dense_10 (Dense)             (None, 4)                 28
_________________________________________________________________
dense_11 (Dense)             (None, 1)                 5
=================================================================
Total params: 81
Trainable params: 81
Non-trainable params: 0
_________________________________________________________________

In [44]:
model.compile(loss='mse', optimizer='adam', metrics=['mse','mae'])
```

When the model is fit, the training and validation loses are deviated and shown. The chosen amount of trainings, or epochs, is 150. This means the model will be forward-propagated and back-propagated 150 times.

```
In [49]:
>>> history = model.fit(X_train, y_train, epochs=150, batch_size=50,  verbose=1, validation_sp
1578/1578 [==============================] - 6s 4ms/step - loss: 0.0082 - mse: 0.0082 - mae: 0.0179 - val_loss: 0.0078 - val_m
se: 0.0078 - val_mae: 0.0175 0.0083 - m - ETA: 1s - loss: 0.0082 - ms
Epoch 144/150
1578/1578 [==============================] - 2s 1ms/step - loss: 0.0082 - mse: 0.0082 - mae: 0.0177 - val_loss: 0.0078 - val_m
se: 0.0078 - val_mae: 0.0154
Epoch 145/150
1578/1578 [==============================] - 2s 1ms/step - loss: 0.0082 - mse: 0.0082 - mae: 0.0176 - val_loss: 0.0078 - val_m
se: 0.0078 - val_mae: 0.0160
Epoch 146/150

1578/1578 [==============================] - 2s 1ms/step - loss: 0.0082 - mse: 0.0082 - mae: 0.0178 - val_loss: 0.0078 - val_m
se: 0.0078 - val_mae: 0.0152
Epoch 147/150
1578/1578 [==============================] - 2s 1ms/step - loss: 0.0082 - mse: 0.0082 - mae: 0.0175 - val_loss: 0.0078 - val_m
se: 0.0078 - val_mae: 0.0173
Epoch 148/150
1578/1578 [==============================] - 3s 2ms/step - loss: 0.0082 - mse: 0.0082 - mae: 0.0179 - val_loss: 0.0078 - val_m
se: 0.0078 - val_mae: 0.0182
Epoch 149/150
1578/1578 [==============================] - 2s 2ms/step - loss: 0.0082 - mse: 0.0082 - mae: 0.0178 - val_loss: 0.0078 - val_m
se: 0.0078 - val_mae: 0.0161
Epoch 150/150
1578/1578 [==============================] - 3s 2ms/step - loss: 0.0082 - mse: 0.0082 - mae: 0.0177 - val_loss: 0.0078 - val_m
se: 0.0078 - val_mae: 0.0171
```

Once the model is fit, predictive analysis can take place.

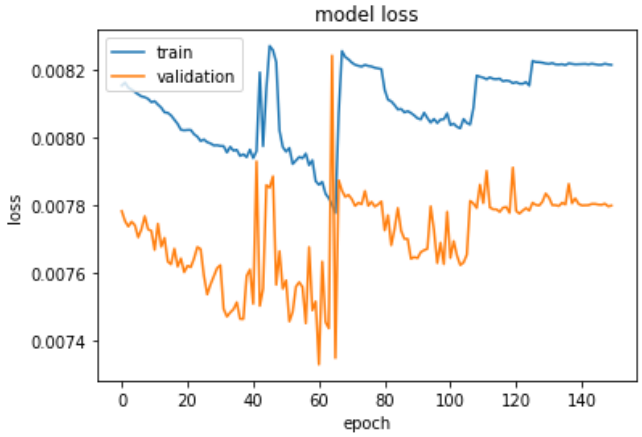
V. Simulation Results and Performance Analysis

Plugging in values for the inputs defined will calculate and result in a predicted number representing approximately 0 if the cars did not collide or approximately a 1 if the cars do indeed collide into one another.

```
In [55]:
Xnew = np.array([[0.100278,7.220009,326,80,15,0.058550,7.22]])
Xnew= scaler_x.transform(Xnew)
ynew= model.predict(Xnew)
#invert normalize
ynew = scaler_y.inverse_transform(ynew)
Xnew = scaler_x.inverse_transform(Xnew)
print("X=%s, Predicted=%s" % (Xnew[0], ynew[0]))

X=[1.002780e-01 7.220009e+00 3.260000e+02 8.000000e+01 1.500000e+01
 5.855000e-02 7.220000e+00], Predicted=[0.00118777]
```

While the implementation of regression based neural networks have potential can provide accuracy, this is only possible when being fine-tuned to fit one's data. Although the predictions itself worked, the output for the training and validation loss was not as expected possibly for two reasons: Not optimizing the neural network in terms of batch size, epochs, input and hidden layer number, etc., but is likely due to overfitting the data. One could see that the loss starts off minimal and barely decreases:



The data generation made it so that there was always an intersection, but because of the randomized speed of vehicles, it did not guarantee that the vehicles would collide. When generating the randomness, it was kept simple: The host vehicle started in the same location every time. This was on purpose because when looking on a global scale, it won't matter what the coordinates are but rather the speeds and directions of each vehicle relative to each other towards an intersection. However, if one point is defined for the vehicles whether it is the host coordinates, guest coordinates, or the intersection coordinates, all other points can be found.

VI. Concerns

Potential problems that may arise with this project include properly generating data as well as properly training the data as well. It took more time to figure out how to generate the data that it did to implement the neural network, which ultimately led to falling behind in regard to completion. However, when trying to create a neural network from scratch, it did not want to work with the data given due to NaN loss [10]. That is what eventually cause a switch to implement the neural network using Keras and TensorFlow. Additionally, once the neural network was up and running, it was difficult to fine tune it to make the learning and subsequently the graph more gradient. However, it did seem to predict whether a crash would or would not occur based on the inputs.

VII. Conclusion

It seems as if predicting vehicle collisions based upon locations and velocities may not be as difficult as it may show at first. However, it is crucial that the correct independent variables are used as input for the neural network: this would have been entirely possible if not as much time was spent getting the coordinate system working or if more time was permitted. It would probably make more sense to also give more randomization in terms of the host vehicles location, speed, and azimuth, that way the neural network has more to data to normalize and derive.

VIII. Future Work

To continue this research, it is necessary to generate more data for consumption, and then optimize the learning

algorithm to minimize loss without overfitting. Firstly, this model fits a spherical earth, where the calculations give about a 0.3% error, so it would make sense to update the coordinate system to fit an ellipsoidal earth. Secondly, it would be important to establish a form of communication between vehicles and allow the simulation for vehicles to actually move. However, in the real world, concerns arise in terms of communicating between vehicles. It would be difficult to create technology for IoT data streams, as well as networking and processing that data in addition to ensuring that data being networked is secure. One must intelligently select needed streams and integrate them for useful content [11]. This could also be possible by implementing some sort of blockchain, which is a decentralized database that holds records, usually transactions between entities.

Some pseudocode is provided:

```
while(vehicleIsInMotion()):
    loc HOST0 = location.update()
    signal.transmit(A0, radar.getNearNeighbor)
    loc GUEST0 = signal.request()
    blockchain.send(create(New block B0))
    vehicle.validate(block.transaction)
    vehicle.request(PoF)
    blockchain.update(block)
    loc HOST1 = location.update()
    signal.transmit(A1, radar.getNearNeighbor)
    loc GUEST1 = signal.request()
    blockchain.send(create(New block))
    vehicle.validate(block.transaction)
    vehicle.request(PoF)
    Feed NN with information regarding polar
  locations and velocities
    Use data model to predict if collision will occur or
  not and decide what action to take next
    blockchain.update(block)
```

However, a potential false proof of location could cause major calibration issues when deciding how to move a vehicle [12]. In terms of an intelligent agent, some factors may limit how much it can actually do. For example, the amount of demonstrated data may not be enough to represent the behavior of the system, and we need as much information as possible to minimize error and reach our goal [13]. Lastly because of the many functions needed as well as real time processing from one vehicle to another, these ideas would need to be implemented with parallel techniques and architecture to ensure and maximize efficiency in real time.

REFERENCES

[1] "New WHO report highlights insufficient progress to tackle lack of safety on the world's roads." Koninklijke Brill NV, doi: 10.1163/2210-7975_HRD-9841-20180026.

[2] "National Motor Vehicle Crash Causation Survey: Report to Congress," p. 47.

[3] "Understanding Latitude and Longitude." https://journeynorth.org/tm/LongitudeIntro.html (accessed Dec. 14, 2020).

[4] "Calculate distance and bearing between two Latitude/Longitude points using haversine formula in JavaScript." https://www.movable-type.co.uk/scripts/latlong.html (accessed Dec. 14, 2020).

[5] "python - Get lat/long given current point, distance and bearing," *Stack Overflow*. https://stackoverflow.com/questions/7222382/get-lat-long-given-current-point-distance-and-bearing (accessed Dec. 14, 2020).

[6] E. Spinielli, "Understanding Great Circle Arcs Intersection Algorithm," *Enrico's blog*, Oct. 19, 2014. https://enrico.spinielli.net/2014/10/19/understanding-great-circle-arcs_57/ (accessed Dec. 14, 2020).

[7] D. Bertels, "Intersection of Great Circles," p. 37.

[8] "Finding The Intersection Of Two Arcs That Lie On A Sphere." https://blog.mbedded.ninja/mathematics/geometry/spherical-geometry/finding-the-intersection-of-two-arcs-that-lie-on-a-sphere/ (accessed Dec. 14, 2020).

[9] "Keras: Regression-based neural networks," *DataScience+*. https://datascienceplus.com/keras-regression-based-neural-networks/ (accessed Dec. 14, 2020).

[10] "Debugging a Machine Learning model written in TensorFlow and Keras | by Lak Lakshmanan | Towards Data Science." https://towardsdatascience.com/debugging-a-machine-learning-model-written-in-tensorflow-and-keras-f514008ce736 (accessed Dec. 14, 2020).

[11] K. Yasumoto, H. Yamaguchi, and H. Shigeno, "Survey of Real-time Processing Technologies of IoT Data Streams," Journal of Information Processing, vol. 24, no. 2, pp. 195–202, 2016, doi: 10.2197/ipsjjip.24.195.

[12] G. Brambilla, M. Amoretti, F. Medioli, and F. Zanichelli, "Blockchain-based Proof of Location," 2018 IEEE International Conference on Software Quality, Reliability and Security Companion (QRS-C), pp. 146–153, Jul. 2018, doi: 10.1109/QRS-C.2018.00038.

[13] C. You, J. Lu, D. Filev, and P. Tsiotras, "Advanced planning for autonomous vehicles using reinforcement learning and deep inverse reinforcement learning," Robotics and Autonomous Systems, vol. 114, pp. 1–18, Apr. 2019, doi: 10.1016/j.robot.2019.01.003.