Formal Analysis of a Neural Network Predictor in Shared-Control Autonomous Driving

John Grese*
Carnegie Mellon University, CyLab

Corina Păsăreanu†
Carnegie Mellon University, CyLab and NASA Ames Research Center

Erfan Pakdamanian‡
University of Virginia, LinkLab

Autonomous driving systems may encounter scenarios where it is necessary to transfer control to the human driver, for instance when encountering unpredictable dangerous road conditions. To be able to do so safely, the autonomous system needs an estimate of how long it will take for the human driver to take control of the vehicle. Deep neural networks can be used for making such predictions, however proving that neural networks meet critical safety requirements presents a challenge. We present a formally verified neural network which predicts "Takeover-time" in a shared-control autonomous driving system. The network is trained on data collected from a (semi-)autonomous driving simulator. We use *Marabou* (a formal verification tool), to analyze the network's sensitivity, local robustness, contextual robustness, and to find adversarial inputs which produce unsafe outputs.

I. Objectives and Impacts

The work reported here is done within a project called *Safe-SCAD — Safety of shared control in autonomous driving** whose objective is to answer *how humans and machines can safely share control of an autonomous car*. Ensuring that drivers retain sufficient situational awareness to *be able to take control of the vehicle in an emergency* [1] is a challenging problem. This is due to the uncertainties associated with measuring the level of situational awareness of drivers while not in control of the vehicle, and with the mapping of such measures to takeover times. As emphasized in the US Department of Transportation strategic documents, there is an urgent need for solutions to *critical research questions regarding driver transitions between automated and manual driving modes* [2].

The Safe-SCAD project aims to extend, adapt and integrate the recent research and the latest advances from human behavior and cognitive modeling, verification of deep neural networks, and automated controller synthesis to tackle these challenges.

The project will make significant and generalizable *impacts* in the areas of:

- shared autonomy
- training and verification of machine learning
- monitoring of autonomous systems by human operators

The project's team has designed and conducted a human subject study on a driving simulator located at University of Virginia. The simulator is a PC-based system consisting primarily of three 30-inch monitors, a steering wheel, accelerator and brake pedals, and eye tracking glasses. In this study, 20 subjects (11 female, 9 male) aged 18-30 (mean= 23.5, SD= 3.1) were recruited. All participants were hired at University of Virginia and were required to have normal or corrected-to-normal vision, to not be susceptible to simulator sickness, and to have at least one year of driving experience to be eligible for participation in this study. Before the experiment, participants were questioned as to their age and driving experience. None of them had prior experience of interaction with autonomous vehicles. Three participants' data were later excluded from the analysis, due to biometric data loss and a large amount of missing values. Participants received $20 to compensate for the time they spent in this study.

*Carnegie Mellon University
†NASA
‡University of Virginia
*This is a joint project between University of Virginia, University of York and Carnegie Mellon University, funded by the Assuring Autonomy International Program from University of York, UK

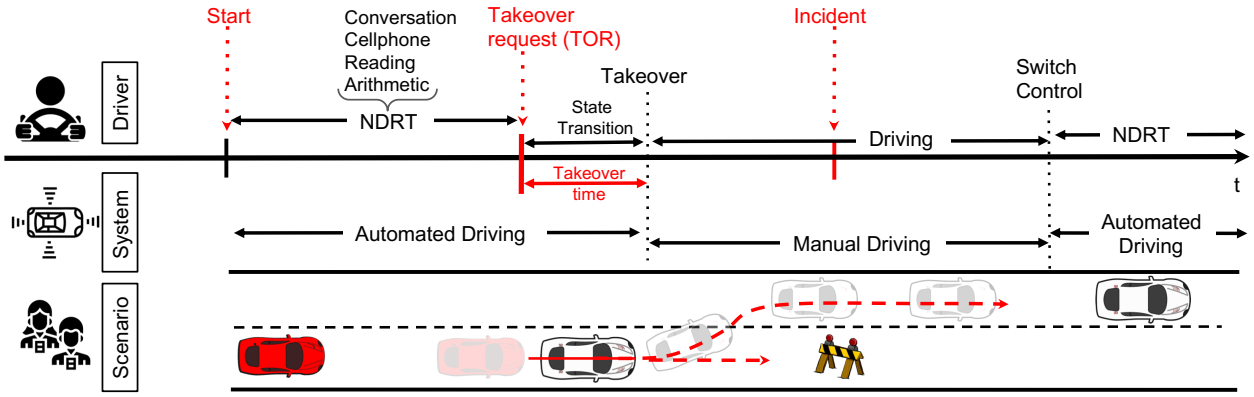

Fig. 1 A schematic view of an example of a takeover situation used in our study, consisting of: 1) takeover timeline associated with participants' course of action; 2) system status; and 3) takeover situation. The vehicle was driven in the automated mode to the point after the TOR initiation and transitioning preparation period. The ego-vehicle is shown in red and the lead car is white. When the ego-vehicle reaches its limits, the system may initiate (true alarm) or fail (no alarm) to initiate the TOR, and the driver takes the control back from the automated system.

The preliminary data from this study was used to train a neural network that achieves 86% accuracy of driver takeover time prediction, with the takeover time organized into several categories, e.g. fast, med-fast, medium, med-slow, and slow. In this paper we report on the formal verification of the neural network using the Marabou [3] verification tool to analyze its robustness and sensitivity to input perturbations.

II. Takeover Time Network

A. User Study

The subjects were briefed about the semi-autonomous systems, the driving tasks and non-driving-tasks (NDRTs), they proceeded to the main driving scenario. The participants were instructed to follow the lead car, stay on the current route, and follow traffic rules as they normally do. *Figure 1* illustrates the scenario that the participants went through in this study. The participants were cautioned that they were responsible for the safety of the vehicle regardless of its mode (manual or automated). As the vehicle approaches the obstacle, it alerts the driver with an alarm requesting that the driver take control of the vehicle. "Takeover time" refers to the amount of time between when the "takeover request" alarm (*TOR*) is triggered and when the driver has taken control (*Takeover*). It is calculated by subtracting the timestamps of *TOR* and *Takeover*. Therefore, they were required to be attentive and to safely resume control of the vehicle in case of failures and takeover requests (TORs). The given instruction enabled the drivers to respond meticulously whenever it was required and to reinforce the idea that they were in charge of the safe operation of the vehicle. Due to the system's limitations, participants were told to maintain the speed within the acceptable range (< 47mph). The experiment was conducted utilizing scenarios consisting of sunny weather conditions without considering the ambient traffic. In addition, the order of NDRT engagement was balanced for participants (see Figure 1). The experiment consisted of three trials, each containing 15 TORs, followed by a 5-minute break between trials.

B. Data Preparation

The goal of this project is to provide a procedure to not only reliably predict drivers' takeover time before a TOR initiation (reported in [4]), but also formally verify safety properties of the model. Hence, the taken procedure for data preparation depends on the driving setting, collected data and the context. Herein, we incorporate data of drivers' physiological measurements, as well as vehicle dynamic data. We initially apply data preprocessing techniques including outlier elimination, missing value imputation using mean substitutions, and smoothing to reduce artifacts presented in raw data. It is worth mentioning that we exclude any data stream providing insights about the unknown future (e.g.,

label	lbound (ms)	ubound (ms)
fast	0	1612
med-fast	1612	2802
med	2802	5180
med-slow	5180	6370
slow	6370	+inf

Table 1 Takeover time categories

type of alarm) or containing more than 50% missing value. The preprocessed time series data are then segmented into 10-second fixed time windows *prior to the occurrences of TORs* with the offset sliding window of 1, experimentally [4]. For instance, if TOR happened at T, we only used data captured in the fixed time window of (T-10s, T) and did not include any data later than T. However, depending on specific applications and contextual requirements, the selected time window length could vary. Subsequently, the segmented windows from modalities are processed to extract meaningful features describing the attributes impacting takeover behavior.

For the eye movement, we acquire interpolated features extracted from raw data through iMotion software. The extracted eye movement attributes include gaze position, pupil diameters of each eye, time to first fixation, and fixation duration/sequence on the detected area of interest.

Finally, the generated features from each modality concatenated to create a rich vector representing driver takeover attributes. The joint representations of all feature vectors with the provision of their associated labels are eventually fed into neural network for training. The final processed dataset consists of 25 input features including: *FixationDuration, FixationSeq, FixationStart, FixationX, FixationY, GazeDirectionLeftZ, GazeDirectionRightZ, PupilLeft, PupilRight, InterpolatedGazeX, InterpolatedGazeY, AutoThrottle, AutoWheel, CurrentThrottle, CurrentWheel, Distance3D, MPH, ManualBrake, ManualThrottle, ManualWheel, RightLaneDist, RightLaneType, LeftLaneDist, LeftLaneType*. These features were labeled with 5 output targets of *fast, med-fast, med, med-slow, slow*.

C. Neural Network Architecture

These features are labeled into the five classes shown in Table 1 and one-hot encoded. Then, the minority classes are upsampled to ensure that all of the classes are represented equally. The target (y) columns are separated from the training data, and the input (x) values are scaled using standard scaling, which standardizes features by removing the mean and scaling to the unit variance. The dataset is then divided, using 80% for training, 10% for testing, and 10% for validation.

The neural network is a fully-connected feed-forward classifier with three hidden layers as shown in figure 2. The input layer has 25 nodes. The three hidden layers have 21, 18, and 11 nodes, and use ReLU as the activation function. The output layer has 5 nodes and uses Softmax. The input layer's 25 inputs (x_0 through x_{24}) map to the feature names listed in theData Preparation section. Inputs x_0 through x_{10} are provided by the simulator's eye tracking system, and describe the driver's fixation, gaze, and pupil diameter. Inputs x_{11} through x_{19} provide information related to both autonomous and manual steering, braking, throttle, and vehicle speed. Inputs x_{20} through x_{24} provide information around the vehicle such as lane type, position, and distance. The output layer's 5 nodes (y_0 through y_4) represent the takeover time class labels listed in table 1.

We utilize Softmax cross-entropy loss with an Adam optimizer and a learning rate of 0.001 to update the parameters and train the network. In each iteration, we randomly sample a batch of data in order to compute the gradients with a batch size of 16. Once the gradients are computed, the initiated parameters are updated. The early stopping method set to 50 epochs prevents overfitting. The resulting network has an accuracy of ~86%.

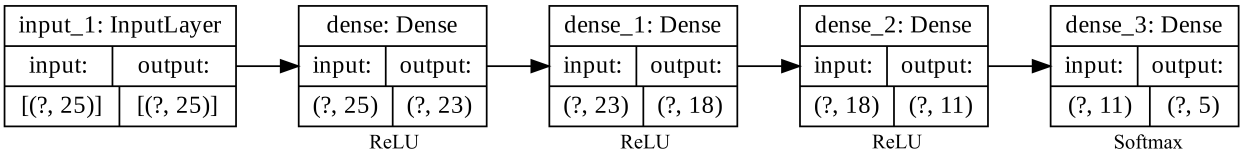

Fig. 2 Neural Network

III. Verification with Marabou

Deep neural networks (DNNs) are increasingly used in safety-critical applications such as autonomous transport, raising serious safety and security concerns. To address these concerns, DNNs need to be validated to ensure that they meet important safety requirements. However, validation of DNNs is challenging due to the nature of data-driven learning and lack of meaningful specifications. Evaluating sensitivity to input perturbations and robustness against adversarial attacks is also challenging due to the huge input space and unclear boundaries.

In this paper we report our investigation of techniques that provide assurance guarantees for neural networks. The problem is difficult as it is known that neural networks are unstable with respect to so called *adversarial perturbations* [5, 6], which are (minimal) changes that cause the network to misclassify an input. They can be devised without access to the training set [7] and are transferable [8] in the sense that an example misclassified by one network is also misclassified by a network with a different architecture, even if it is trained on different data. Existing testing and approximation techniques [6, 8–11] that can be used for the analysis of neural networks are inherently limited as they can not provide guarantees.

To address this limitation, recent work has employed formal verification methods based on Satisfiability Modulo Theory (SMT) that provide sound assurances that no adversarial examples exist within a given neighborhood of an input. Marabou [3] is an SMT-based neural network verification framework which can be used to provide formal guarantees about a neural network. Marabou works by accepting queries about the network's properties and transforming these queries into SMT constraint satisfaction problems. It is capable of accommodating networks with different activation functions and topologies, and performs high-level reasoning on the network to curtail the search space and improve performance. Common Marabou queries are expressed in terms of upper and lower bounds on the network's inputs, and an expected output. After solving a query, Marabou either returns "UNSAT", or "SAT" with a counterexample if one was found.

Using Marabou, we performed local robustness checks, targeted robustness checks, a sensitivity analysis, and data-driven verification on the takeover-time network. All verification was performed on an Amazon Web Services t2.x-large EC2 instance with 16GB of RAM and a 64bit Intel(R) Xeon(R) E5-2686 v4 CPU @ 2.30GHz with 4 cores.

A. Local Robustness

Local robustness can be defined as the minimum perturbation $\pm\delta$ applied uniformly to all features ($x_0 ... x_{24}$) from an single input x, which causes a change in the network's prediction. We find the minimum $\pm\delta$ using Marabou by evaluating a series of input queries with the lower bounds set to $x - \delta$, and the upper bounds set to $x + \delta$ over a range of values for δ until finding the minimum $\pm\delta$ that causes the predicted label to change. Specifically, we perform something similar to a binary search over the range of possible values for δ. For each value of δ, we perform a query for every label other than the expected label for x to ensure that no other labels exist between $\|x - \delta\|$ and $\|x + \delta\|$. Using the value of δ we find from local robustness for a given input x, we are given the guarantee described by equation 1, which states that for any input $x\prime$ such that the distance from x is smaller than δ, the network will make the same prediction.

$$\forall x\prime \; s.t. \; \|x\prime - x\| < \delta \Rightarrow f(x\prime) = f(x). \tag{1}$$

We checked the network's local robustness on a set of ~2500 inputs from the dataset. Table 2 shows the minimum and average values of $\pm\delta$ grouped by label. With respect to the inputs we tested, the minimum observed δ was 0.00011, which belonged to the *med* class. The highest observed δ was 0.65801, which belonged to the *med-slow* class. The class with the lowest mean δ was *med-slow*, and the class with the highest mean δ was *fast*. We can see from figure 3 that the δ for the majority of inputs fell within the range 0.001 to 0.05. The runtime per input of the local robustness checks took between ~1.2 seconds and ~61 minutes, with an average of ~4.9 minutes.

label	mean δ	min δ	max δ
fast	0.028733	0.00030	0.26471
med-fast	0.026576	0.00015	0.16012
med	0.027524	0.00011	0.13579
med-slow	0.023497	0.00040	0.65801
slow	0.026293	0.00033	0.19719
overall	0.026513	0.000110	0.658010

Table 2 Local robustness results

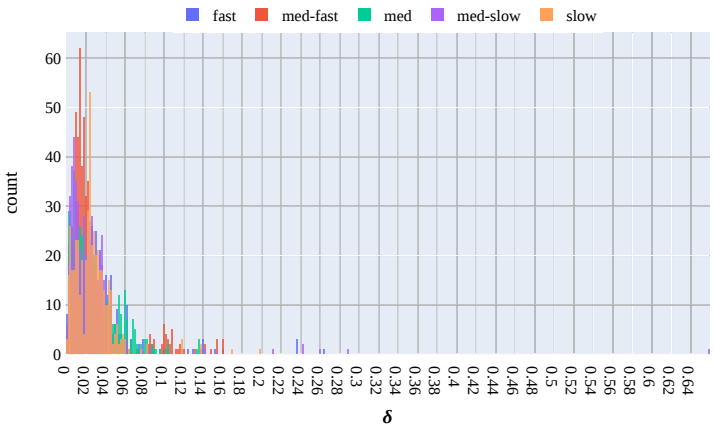

Fig. 3 Local robustness distribution

B. Targeted Robustness

We define targeted robustness as the minimum perturbation $\pm\delta$ applied uniformly to all features $(x_0...x_{24})$ of an input x which produces a *specific* change in the network's predicted label. Focusing on specific changes in the network's prediction provides more specific robustness guarantees than local robustness, and can also be helpful in finding targeted adversarial inputs. The process for targeted robustness checks is similar to local robustness, however instead of proving that only one label exists between $x+\delta$ and $x-\delta$, we prove that the targeted label *does not* exist. The guarantee provided by targeted robustness checks for a given x is described in equation 2, which states that for any input $x\prime$ such that the distance from x is smaller than δ, the network will not predict the target label.

$$\forall x\prime \; s.t. \; \|x\prime - x\| < \delta \Rightarrow f(x\prime) \neq target. \tag{2}$$

Because safety is our primary goal, we targeted the most unsafe change in classification — *slow* inputs which change to *fast* with a perturbation of δ. The results of the targeted robustness checks on a set of 500 *slow* inputs can be seen in Table 3. For the *slow – fast* target, the minimum δ was 0.0013, the mean δ was 0.073864, and the maximum δ was 0.749950. We can see from the distribution in figure 4 that the δ for the majority of inputs fell within the range 0.008 - 0.06.

The targeted robustness checks for the *slow-fast* target took longer to complete on average than the local robustness checks, which is a little counter-intuitive because fewer queries are performed in the targeted robustness. The reason for this is that the *slow-fast* target requires Marabou to work harder to find a counterexample. The runtime per input for our targeted robustness checks took between ~0.7 seconds and ~78 minutes, with an average of ~10.7 minutes.

target	mean δ	min δ	max δ
slow – fast	0.073864	0.00130	0.749950

Table 3 Targeted robustness results

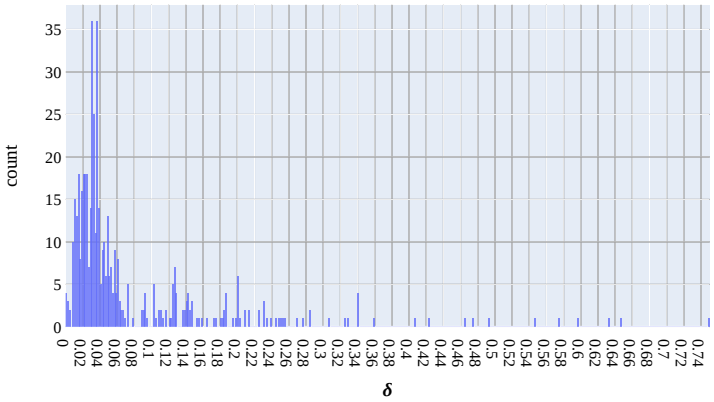

Fig. 4 Targeted robustness distribution

C. Sensitivity Analysis

To learn more about the model's sensitivity to perturbations of individual features, we performed a sensitivity analysis using Marabou. Sensitivity can be defined as the minimum perturbation δ applied to an individual feature x_i of an input x which causes a change in the network's prediction. Perturbing each feature individually also provides some visibility into feature importance. In addition, using a formal verification tool such as Marabou for this type of analysis also gives strong formal guarantees. We took two different approaches to analyzing the network's sensitivity — *symmetric* and *asymmetric*. The symmetric approach considers a single value of δ for the negative and positive perturbations to each feature ($x_0 ... x_{24}$), which extends the space around x_i symmetrically. The asymmetric approach considers individual values of δ for the negative and positive perturbations ($l\delta$ and $u\delta$) to each feature. These two different approaches to sensitivity analysis yield slightly different pictures of the network's sensitivity. In the following two sections, we discuss the symmetric and asymmetric sensitivity analyses in more detail.

1. Symmetric Sensitivity Analysis

The symmetric variant of the sensitivity analysis searches the input space around each feature x_i of an input x to find the minimum value $\pm\delta$ which causes a change in prediction. The process is similar to local robustness, however we only perturb a single feature at a time. Using Marabou, we evaluate different values of $\pm\delta$ by generating input queries for each one with the lower bound of x_i set to $x_i - \delta$ and the upper bound set to $x_i + \delta$ until discovering the minimum δ that causes a change in the predicted label. For this type of sensitivity analysis, Marabou provides the guarantee that any perturbation smaller than $\pm\delta$ applied to a single feature x_i will not change the network's prediction. Equation 3 describes this guarantee provided by Marabou for a feature x_i of an input x.

$$\forall x_i\prime \text{ s.t. } \|x_i\prime - x_i\| < \delta,\ f([x_0 ... x_i\prime ... x_n]) = f([x_0 ... x_i ... x_n]) \tag{3}$$

Using this method, we analyzed the sensitivity of all 25 features on ~2500 inputs from the dataset. Figure 5 shows the results from the symmetric sensitivity analysis. The results show that the model is most sensitive to changes of *ManualWheel* (x_{19}) and least sensitive to *FixationX* (x_3). These results make sense in the context of predicting takeover time because changes in manual wheel indicate that the driver's hands are on the wheel, and the way that the simulation was designed did not necessarily require the human to move their fixation to the left or right. Analyzing a single feature x_i took a minimum of ~0.6 seconds, a maximum of ~4.5 minutes, with an average of ~36.4 seconds, which equates to an average of ~15.7 minutes per input x.

2. Asymmetric Sensitivity Analysis

The asymmetric variant of our sensitivity analysis operates similarly to the symmetric variant, but adjusts the lower and upper bounds of x_i independently. We refer to the perturbation to the lower bound as $l\delta$ and the perturbation to the upper bound as $u\delta$. For each feature x_i of input x, we use Marabou to evaluate separate queries for $l\delta$ and $u\delta$ over a range of values until discovering the minimum $l\delta$ and $u\delta$ required to cause a change in the predicted label. The queries on the lower bound perturb feature x_i by $x_i - l\delta$, and the queries on the upper bound perturb the feature x_i by $x_i + u\delta$.

For the lower bound of the asymmetric sensitivity analysis, Marabou provides the guarantee that any perturbation smaller than $l\delta$ *subtracted* from feature x_i of an input x, the network will predict the same label. For the upper bound, it is guaranteed that the network will predict the same label when any perturbation smaller than $u\delta$ is *added* to feature x_i. The guarantee for the lower bound's perturbation $l\delta$ is described by equation 4, and the upper bound's perturbation $u\delta$ is described by equation 5.

$$\forall x_i\prime \text{ s.t. } \|x_i\prime - x_i\| < l\delta,\ f([x_0 ... x_i\prime ... x_n]) = f([x_0 ... x_i ... x_n]) \tag{4}$$

$$\forall x_i\prime \text{ s.t. } \|x_i\prime - x_i\| < u\delta,\ f([x_0 ... x_i\prime ... x_n]) = f([x_0 ... x_i ... x_n]) \tag{5}$$

We used the asymmetric sensitivity analysis to evaluate ~2500k inputs from the dataset. Figure 6 shows the model's mean sensitivity by feature. Again, we can see that the model is sensitive to changes in the *ManualWheel* feature. Another interesting observation is that on average, the model is significantly more sensitive to negative perturbations of *CurrentWheel*, indicating that the model is more sensitive when the vehicle maneuvers to the left. This is likely due to the fact that the simulation always had the obstacle on the right side of the road, so the vehicle always maneuvered to the left during the takeover.

The asymmetric approach takes approximately twice as long as the symmetric approach to compute due to the fact that the $l\delta$ and $u\delta$ must be discovered in isolation. However, even though it takes more time, it yields more information with respect to how sensitive the model is to negative vs positive perturbations, and thus has the possibility of discovering bias in the model.

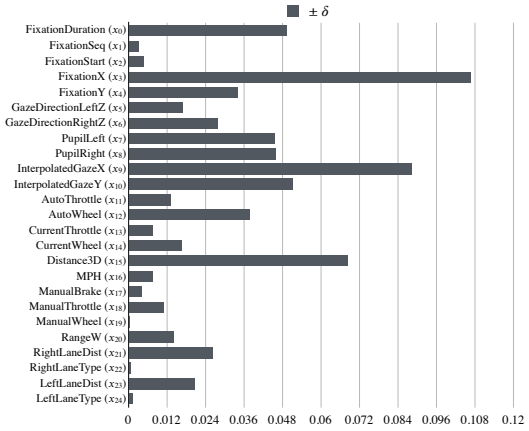

Fig. 5 Symmetric sensitivity results

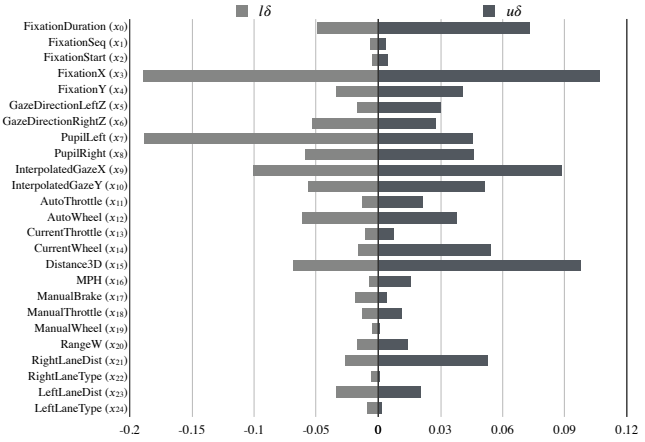

Fig. 6 Asymmetric sensitivity results

D. Clustering for Data-driven Verification

The data-driven verification technique tests the robustness of targeted regions from the input space that contain a dense population of points of a single label. This approach, which is adopted from the *Deep Safe*[12] technique, allows us to target the most relevant (densely populated) regions of the input space for verification. We start by running a modified K-Means clustering algorithm on the dataset to produce regions which contain points of a single label. Each region consists of a centroid, a radius, and a label. Then, we verify these regions with Marabou, using the centroids as inputs. The result of this verification technique is a set of targeted, "safe regions" which have been proven to contain points of a single label. Another benefit of this approach is that by targeting relevant regions of the input space, we can verify larger regions using fewer inputs, thus covering more of the input space with fewer queries. Furthermore, the results from this data-driven approach can also be useful to provide a measure of confidence about the network's predictions. The following sections describe the clustering algorithm, verification technique, results, and the proposed run-time usage of the verification results.

1. Label-guided K-Means Clustering

To identify regions of points of a the same label, we use a modified K-Means clustering algorithm called *label-guided K-Means* [12]. The regular K-Means algorithm is an unsupervised approach which does not provide any guarantees that the clusters will contain points of the same label, which means that the clusters may not be useful for verification. Label-guided K-Means clustering solves this problem by using the inputs' labels to help guide the K-Means algorithm to produce regions containing points of the same label. This is accomplished by applying K-Means with n set to the number of unique labels, checking the number of labels in each cluster, and then repeating the process iteratively to divide the K-Means clusters into regions which contain points of a single label. The output of the algorithm is a set of regions, each one consisting of a centroid, a radius, and a label. The $L2$ (euclidean) distance metric is used to measure distance between the points and compute the radius. Fig 7 shows Label-Guided K-means applied to a simplified example. The first step in the figure shows the initial K-Means clustering, the second shows the final iteration of the algorithm, and the third shows the resulting regions' centroids and radii. We also compute the density of each region as $r \div n$ where r is the region's radius, and n is the number of points in the region. Code listing 1 shows a python implementation of label-guided K-Means.

When applied to the takeover time dataset, the algorithm produced a 6138 regions with 10 or more points, however 484 of those regions had centroids which were incorrectly predicted by the network, so they were discarded, leaving us with a total of 5654 regions containing 10 or more inputs which were used for verification. These 5654 regions effectively cover ~88% of the points from the dataset.

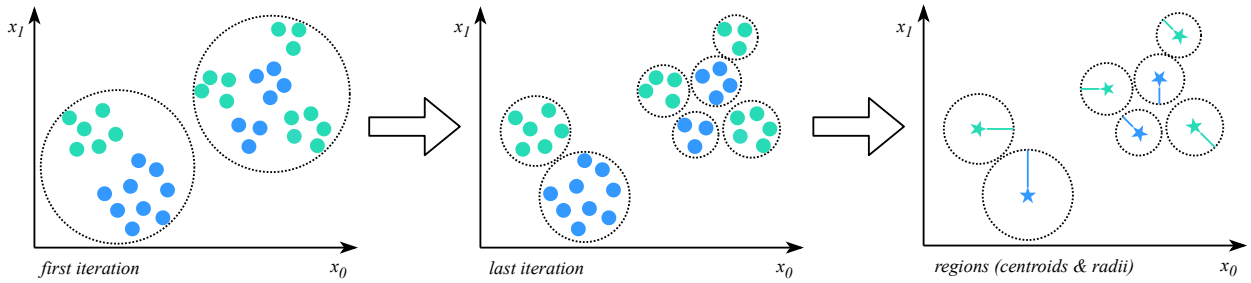

Fig. 7 Label-guided K-means

```python
import numpy as np
from sklearn.cluster import KMeans
from scipy.spatial import distance

# X:np.ndarray of inputs, Y:np.ndarray of labels
def label_guided_kmeans(X, Y):
    regions = [] # list of completed regions
    remaining = [(X, Y)] # stack of remaining inputs/labels
    while len(remaining) > 0:
        X, Y = remaining.pop(0) # pop inputs/labels to cluster from stack
        Yuniq = np.unique(Y, axis=0) # unique labels in Y
        n = Yuniq.shape[0] # number of unique labels in Y
        # initial centroids for KMeans (mean of inputs from each label)
        init = np.array([X[np.where(Y==y)[0]].mean(axis=0) for y in Yuniq])
        model = KMeans(n_clusters=n, init=init).fit(X) # run kmeans
        Yhat = model.predict(X)
        for c in np.unique(Yhat, axis=0):
            idxs = np.where(Yhat == c)[0] # indexes of inputs in cluster
            Xc, Yc = X[idxs], Y[idxs] # get inputs/labels in the cluster
            if np.unique(Yc, axis=0).shape[0] == 1:
                # cluster contained a single label; save as a region
                centroid = model.cluster_centers_[c]
                radius = max([distance.euclidean(x, centroid) for x in X])
                region = dict(centroid=centroid, radius=radius, label=Yc[0],
                    n=Xc.shape[0], density=radius/Xc.shape[0])
                regions.append(region)
            else:
                # cluster contained multiple labels; repeat KMeans on cluster
                remaining.append((Xc, Yc))
    return regions
```

Listing 1 Label-guided K-Means Python code

2. Region Verification

The regions discovered by the modified K-Means algorithm were verified using a technique similar to local robustness. For each region, we performed a robustness test with Marabou, using the region's centroids as the input. We set the queries' upper bounds to $centroid + \delta$, and the lower bounds to $centroid - \delta$, and iterated over a range of values for δ until discovering the minimum $\pm\delta$ which caused a change in the predicted label for the region. Using the δ we discovered for a given region, we computed the verified radius of the region as $\|(centroid + \delta) - centroid\|_{L2}$. The guarantee provided the verification of a given region with a radius r and *label* can be described by equation 6, which states that for all x

within the verified radius r of a safe region R, the predicted label will be *label*.

$$[h] \forall x \; s.t. \; \|x - centroid\|_{L2} \leq r \Rightarrow f(x) = label. \tag{6}$$

The results from the data-driven verification results can be seen in Table 4. In the results, we compare the verified radius with the original radius. We can see from the results that for the *med-fast* regions, the average verified radius was larger than original radius. For all other labels, the average verified radius was roughly half of the original. These results show that we were able to verify a larger portion of the *med-fast* regions than the regions from other labels. Even though we were not able to verify the entire region for the other labels, the data-driven approach still allowed us to cover a large portion of the dataset while testing only 5654 individual points. The data-driven verification tests per region took a minimum of ~0.558 seconds and a maximum of ~149.55 minutes, with an average of ~3.38 minutes per region. In general, the regions with higher densities took longer to verify than the regions with lower densities.

category	**verified radius**			**original radius**		
	mean	**min**	**max**	**mean**	**min**	**max**
fast	0.81743	0.00250	8.88250	1.67943	0.11222	8.68825
med_fast	2.07260	0.01250	15.51750	1.98323	0.17113	10.43312
med	0.74406	0.00250	3.62750	1.46866	0.12483	6.87752
med_slow	0.58589	0.00250	6.04750	1.36602	0.04696	13.84459
slow	0.82720	0.00250	4.49500	1.34232	0.11944	5.79814
all	0.84691	0.00250	15.51750	1.50298	0.04696	13.84459

Table 4 Verified regions comparison

3. Run-time Usage

One of the goals of the Safe-SCAD project involves designing a safety-controller which was designed by the University of York. Among other things, the proposed safety-controller has the job of ensuring that the takeover action is handled safely. When the takeover alarm is triggered, the safety-controller will consume the neural network's reaction time prediction to help make a decision on the safest action to take. However, neural networks are probabilistic in nature and may make incorrect predictions under certain conditions. So, to provide a measure of confidence about the neural network's prediction, we have devised a way to use the results from the data-driven verification within the safety controller. To accomplish this, the verified "safe regions" (each consisting of a centroid, a radius, and a label) are provided to the safety-controller in a searchable format, and then searched at runtime to discover whether or not a given input resides within one of the regions and has a matching label. Code listing 2 shows a simplified python implementation of a search function that the safety-controller could call to determine whether or not a given input resides within one of the safe regions. If the input resides within a safe region with a matching label, the safety-controller would have a strong degree of confidence that the prediction is correct. Otherwise, the safety-controller knows that it has a lower degree of confidence about the prediction because it is either from part of the input space that was not formally verified, or exists in a verified region but has a label mismatch.

```
from scipy.spatial import distance

# regions:list of safe regions, x:the input, y:predicted label for x
def exists_in_safe_region(regions, x, y):
    for r in regions:
        label_match = r['label'] == y
        in_region = distance.euclidean(r['centroid'], x) <= r['radius']
        if label_match and in_region:
            return True
    return False
```

Listing 2 Searching safe regions

IV. Conclusion and Next Steps

We presented the formal verification of a neural network that predicts takeover-time in a shared-control semi-autonomous driving system. We analyzed its sensitivity and robustness with several techniques using the Marabou verification tool. The sensitivity analysis provides insight into the importance of input features, in addition to providing formal guarantees with respect to the regions in the input space where the network behaves as expected. The asymmetric sensitivity analysis has the added benefit of providing the opportunity to discover biases with respect to negative and positive perturbations to individual features. We also evaluated the network's robustness using local robustness, targeted robustness, and a data-driven verification approach. Compared to targeted and local robustness, the data-driven approach has the benefit of verifying the robustness of larger regions of the input space while testing fewer inputs. We have also shown an example of how these results can be leveraged and included in a "safety controller" to provide confidence about the network's predictions.

For next steps, we plan to integrate the results of the neural network and verification results into the safety controller, which will be then evaluated using the simulator. We also plan to try to improve the network's robustness by experimenting with different adversarial training techniques. One idea is to use the counterexamples discovered by Marabou during re-training, and another is to use an adversarial training framework such as "Clever Hans". Another idea for future work is to experiment with additional clustering algorithms to try to consolidate some of the regions. We also plan to try to reduce the number of features in the model by using the results from our sensitivity analysis along with other feature importance analysis techniques to discover features that may be able to be dropped from the model.

References

[1] KPMG International, "Autonomous Vehicles Readiness Index," , 2018. URL https://home.kpmg.com/content/dam/kpmg/xx/pdf/2018/01/avri.pdf.

[2] US Department of Transportation – Intelligent Transportation System Joint Program Office, "Automation Research at USDOT," , 2018. URL https://www.its.dot.gov/automated_vehicle/avr_plan.htm.

[3] Katz, G., and et al., "The Marabou Framework for Verification and Analysis of Deep Neural Networks," *CAV*, 2019, pp. 443–452.

[4] Pakdamanian, E., Sheng, S., Baee, S., Heo, S., Kraus, S., and Feng, L., "DeepTake: Prediction of Driver Takeover Behavior using Multimodal Data," *arXiv preprint arXiv:2012.15441*, 2020.

[5] Roli, F., Biggio, B., and Fumera, G., "Pattern Recognition Systems under Attack," *CIARP (1)*, Lecture Notes in Computer Science, Vol. 8258, Springer, 2013, pp. 1–8.

[6] Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I., and Fergus, R., "Intriguing Properties of Neural Networks," , 2013. Technical Report. http://arxiv.org/abs/1312.6199.

[7] Papernot, N., McDaniel, P. D., Goodfellow, I. J., Jha, S., Celik, Z. B., and Swami, A., "Practical Black-Box Attacks against Machine Learning," *AsiaCCS*, ACM, 2017, pp. 506–519.

[8] Goodfellow, I. J., Shlens, J., and Szegedy, C., "Explaining and Harnessing Adversarial Examples," , 2014. Technical Report. http://arxiv.org/abs/1412.6572.

[9] Feinman, R., Curtin, R. R., Shintre, S., and Gardner, A. B., "Detecting Adversarial Samples from Artifacts," , 2017. Technical Report. http://arxiv.org/abs/1703.00410.

[10] Carlini, N., and Wagner, D., "Towards evaluating the robustness of neural networks," *Proc. 38th IEEE Symposium on Security and Privacy*, 2017.

[11] Chang, K., Parvez, M. R., Chakraborty, S., and Ray, B., "Building Language Models for Text with Named Entities," *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics, ACL 2018, Melbourne, Australia, July 15-20, 2018, Volume 1: Long Papers*, 2018, pp. 2373–2383. URL https://aclanthology.info/papers/P18-1221/p18-1221.

[12] Gopinath, D., Katz, G., Pasareanu, C. S., and Barrett, C. W., "DeepSafe: A Data-Driven Approach for Assessing Robustness of Neural Networks," *Automated Technology for Verification and Analysis - 16th International Symposium, ATVA 2018, Los Angeles, CA, USA, October 7-10, 2018, Proceedings*, Lecture Notes in Computer Science, Vol. 11138, edited by S. K. Lahiri and C. Wang, Springer, 2018, pp. 3–19. https://doi.org/10.1007/978-3-030-01090-4_1, URL https://doi.org/10.1007/978-3-030-01090-4_1.