Sequencing Behavior in an Intelligent Pro-active Co-Driver System

by

Shokoufeh Monjezi Kouchak

A Dissertation Presented in Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy

Approved April 2020 by the
Graduate Supervisory Committee:

Ashraf Gaffar, Chair
Hani Ben Amor
Adam Doupe
Loretta Cheeks

ARIZONA STATE UNIVERSITY

May 2020

ABSTRACT

Driving is the coordinated operation of mind and body for movement of a vehicle, such as a car, or a bus. Driving, being considered an everyday activity for many people, still has an issue of safety. Driver distraction is becoming a critical safety problem. Speed, drunk driving as well as distracted driving are the three leading factors in the fatal car crashes. Distraction, which is defined as an excessive workload and limited attention, is the main paradigm that guides this research area. Driver behavior analysis can be used to address the distraction problem and provide an intelligent adaptive agent to work closely with the driver, fay beyond traditional algorithmic computational models. A variety of machine learning approaches has been proposed to estimate or predict drivers' fatigue level using car data, driver status or a combination of them.

Three important features of intelligence and cognition are perception, attention and sensory memory. In this thesis, I focused on memory and attention as essential parts of highly intelligent systems. Without memory, systems will only show limited intelligence since their response would be exclusively based on spontaneous decision without considering the effect of previous events. I proposed a memory-based sequence to predict the driver behavior and distraction level using neural network. The work started with a large-scale experiment to collect data and make an artificial intelligence-friendly dataset. After that, the data was used to train a deep neural network to estimate the driver behavior. With a focus on memory by using Long Short Term Memory (LSTM) network to increase the level of intelligence in two dimensions: Forgiveness of minor glitches, and accumulation of anomalous behavior., I reduced the model error and computational expense by adding attention mechanism on the top of LSTM models. This system can be generalized to build and train highly intelligent agents in other domains.

ACKNOWLEDGMENTS

This work was done as part of the “Intelligent Co-driver” project in the “Human Car Interaction (HCaI)” lab under the supervision of Dr. Ashraf Gaffar.

We acknowledge the effort and participation of all ASU graduate and undergraduate students who participated in data collection and analysis of these experiments

s as well as working on other modules of the project as part of their graduate research work, course load, capstone projects, or i-projects. We also want to thank the hundreds of volunteers who enthusiastically participated in these experiments and data collection simulated drives related to this project.

TABLE OF CONTENTS

Page

Page

Page

LIST OF TABLES

LIST OF FIGURES

Figure Page

Figure Page

Figure Page

INTRODUCTION

Driver distraction research started around 50 years ago and has been intensifying over the last decade. Distraction, which is defined as an excessive workload and limited attention is the main paradigm that guides this research area [1]. This definition doesn’t explain the dynamics of distraction and doesn’t express how and under which conditions drivers engage in distracting activities. The dynamics of distraction deem failure in task prioritization and task switching as essential contributors to driver distraction. Disengagement from driving activities such as mind-wandering or divided attention which performing a secondary task and driving might exacerbate situation [2].

Driving, being considered an everyday activity for most people, still has an issue of driver safety. Over the 20 years from 1980 to 2000, the number of licensed drivers in the U.S. increased by 23.7% from about 154.0 million to 190.6 million. Total annual mileage traveled annually in the U.S. increased by 28.9% from 1990 to 2000 and reached 2,767 billion miles in 2000 [3]. Over the years, we have seen emerging technology for safer driving such as Lane Departure Warning [4], Collision Avoidance [5] and Pedestrian Detection [6] that all can help mitigate the threats to drivers. Modern cars with advanced infotainment systems often need more cognitive attention while interacting with the infotainment system consequently causing more distraction [7].

The primary task of driving can be challenging for drivers since it requires full attention and focus on the road, continuously watching other cars, traffic signs and pedestrians. Drivers also need to keep an eye on their own cars’ instrument panel as well as side and rear views [3]. Additional tasks include operating necessary car controls to turn on lights, windshield wipers or turn signals. All these activities make driving inherently distractive [3]. While not essential for proper car operation, driver entertainment was added early on to reduce the boredom of driving. As modern cars become predominantly digital in most of their systems, with a lot of information available, industry needed to communicate some of this information to the driver using an information system. Combining this “Information” system with a complex entertainment system and a significantly larger screen, it is now commonly known as the “Infotainment System”, occupying the mid console of many cars. The design of car infotainment systems moved from simple interface design to multi-modal interaction through big screens with many icons and texts on the screen similar to the computer and phone graphical design. While users can generally afford to have full focus on their smartphone screens, with few exceptions, drivers cannot [8].

Besides the essential driving tasks, some drivers execute several secondary tasks like texting, listening to music, talking to other occupants in the vehicle or over phone. These secondary tasks need cognitive attention while some of them also need visual or physical attention, requiring the driver to take his eyes off the road or his hands off the wheel, causing distraction. Modern cars with advanced infotainment systems often need more cognitive attention consequently causing more distraction. A growing trend in the research community is to find better ways to manage distraction by enhancing the interaction between the car and the driver using design principles from Natural User Interfaces (NUI) and Human-Computer Interaction (HCI) [7, 9].

The Nature of Distraction

Most studies focusing on driver distraction look at different reasons at the same level of necessity and importance, hence treat them equally. However, not all distractions are equal. Even for ideal driving behavior, drivers are expected to take their eyes off the road frequently to check their side and rearview mirrors. It would be considered unsafe driving practice or even unacceptable if the driver did not regularly check these mirrors. On the other hand, checking these mirrors takes the driver’s eyes and attention off the road, albeit for a short time. While dialing a phone number using a cell phone is a form of distraction, it is not the same practice, even if it took the same amount of distraction time as looking in the rearview mirror [10].

Driving can become more complex depending on the driving condition. While some situations require less attention and a short distraction can be allowed, other situations require full driver attention and a split-second decision. Therefore, the diver can get away with distracting activities in some situations, which can lead to a false sense of safety [10].

Driver Distraction Types

Driving-related tasks can be classified into three main groups:

1. Primary tasks of driving like maneuvering steering wheel, brake as well as paying attention to the road.

2. Secondary tasks involve operating utilities. For example, using headlights and windshield wipers.

3. Tertiary tasks are all other tasks related to comfort, information and entertainment functions.

Four types of driver distraction caused by cars' infotainment system were identified [11]:

1. Visual distraction: anything that causes the driver to take her eyes off the road.
2. Manual distraction: any task which causes the driver to take his hands off the steering wheel.
3. Audio distraction: any noise which obscures important voice in the car such as alarms or outside such as ambulance vehicle sirens.
4. Cognitive distraction: any task that causes cognitive load and reduces the attention of the driver to the primary task of driving [8].

While the first three types of distraction are easier to identify and measure, cognitive distraction remains the hardest and the most elusive type of distraction.

Motivation for Distraction

Drivers are not expected to fully and exclusively concentrate on the pure activity of driving. Three major forces can be identified:

1. The human ability to automate repetitive tasks: the human brain is well adapted to identify repetitive tasks and automate them, hence requiring less focus and concentration after an initial learning period. Driving can often follow the same

pattern of automation. A first-year driver definitely requires more focus than an established driver [12].

2. The multitasking nature of humans: by default, people tend to do multitasks to save time and optimize our activities. The human brain is well adapted to multi-tasking most of the time. Only highly cognitive tasks require our full attention. Based on filter theory that discussed in [12], the human brain receives sensory information from multiple sensors until some bottleneck is reached. Though multitasking is feasible for some tasks, the brain may focus on a secondary task and ignore the important information related to the primary task of driving.

3. The big commercial drive for new cars: today's car is loaded with many modern technologies that are becoming more affordable every year. Some new technologies can contribute to driver distraction, even when they look attractive. For example, recent studies found that speech recognition in cars can be as distracting as drunk driving [13].

Three traps can lead drivers to disengage the primary task of driving [10]:

1. Contingency traps are situations where road hazards and road demands are hard to perceive for the driver. Novice drivers are mainly prone to contingency traps.

2. Consequence traps allude to the situations where drivers perceive the driving demands, yet they choose to neglect them since the reward associated with the secondary task outweigh the costs of ignoring the roadway.

3. Conditioning traps allude to the situations that the driver neglects driving because they have experienced similar situations several times before and haven’t suffered any negative outcomes. Conditioning traps are more common between experienced drivers.

The Effects of Distraction on Driving Behavior

Driver distraction is becoming a critical safety problem claiming 3,166 lives in 2017 alone [14][15]. Speed, drunk driving as well as distracted driving are the three leading factors in fatal car crashes [16]. Vision is the most critical sense for safe driving. Drivers that use their cell phones while driving, even hands-free phones, experience inattention blindness which means they look at objects but fail to see up to half of the information in the driving environment. Therefore, they are unable to process all crucial information that they should know to identify potential hazards.

Driver distraction adversely affects some driving behaviors [17]:

1. It reduces lateral and longitudinal control. For example, the ability to maintain lane position, the appropriate speed and head distance.
2. It causes slower detection and reaction to roadway events especially those ones occurring in drivers' peripheral vision.
3. It increases the number of missed traffic signals and traffic signs.
4. It increases impaired visual scanning and search patterns.
5. It causes risky decision making.
6. It diminishes the driver’s awareness of surrounding traffic.
7. It increases the mental workload.

Distracted driving is becoming one of the United States' biggest health concerns [18]. A recent study [19] estimates the effect of distraction on the accidents among teenage drivers at a much higher rate of 58%. The study warns that "distracted driving is now an Epidemic in the U.S." [18]. The Centers for Disease Control and Prevention estimates that nine people are killed every day in the U.S. in car crashes involving a distracted driver. Even short time distraction can be deadly. For example, if you send a short text that takes 5 seconds while driving 55 MPH, you drive the length of a football field blindfold. Based on the Virginia Tech Transportation Institute you are three times more likely to have an accident when you are distracted by a cell phone [18].

One of the largest driver behavior studies, 100-Car Naturalistic Driving, collected a database containing many cases of driving performance including judgment error, severe fatigue, impairment, willingness to engage in secondary tasks as well as risk-taking, traffic violations aggressive driving. The database includes around 2,000,000 vehicle miles, 241 drivers, 43,000 hours of driving data and 12 to 13 months of data collection for each vehicle. Based on the collected data, 78% of the crashes and 65% of near-crashes had one form of inattention or distraction as a contributing factor [20].

"With more than 37,000 deaths on U.S. roads in 2016, we need to continue finding ways to limit driving distractions and improve traffic safety," said Dr. David Yang, executive director of the AAA Foundation for Traffic Safety [21]. "The Foundation's work offers insight on drivers' attitudes toward traffic safety and their behaviors, so we can better understand the issue and identify potential countermeasures to reduce crashes." Drivers that participated in AAA 2016 survey, believe that driver distraction problem increased in the last three years. “As the number of distractions behind the wheel increases from the latest phone apps to in-vehicle technology, it is important that we better educate drivers on the dangers of distraction,” said Jake Nelson, AAA director of traffic safety advocacy and research. “There is a disconnection between what drivers do and what they believe. While most recognize the dangers created by taking your eyes off the road, they engage in distracting behaviors anyway creating a ‘do as I say, not as I do’ culture on the roadway”. In 2016, 444 fatal crashes were reported that involved cell phone use as a distraction (14% of all fatal distraction-affected crashes) and a total of 486 people died in these accidents [22]. In 2017, the driver of a pickup truck collided with a church minibus in Texas while he was texting; 13 people were killed in that accident. Based on a witness claim, the truck was moving erratically prior to that collision for several minutes. In this example, continuous distracting activity led to a fatal accident [23].

Driver distraction is also influencing vulnerable road users: pedestrians, motorcyclists and cyclists. These road users suffer from the lack of protection and visibility issues [24]. In 2016, 562 non-occupants (pedestrians, bicyclists, and others) were killed in distraction-affected crashes. They are smaller and their presences are less mentally proceeding by drivers compared to vehicles. Besides, they don’t have protection in the event of a crash. These factors make them more vulnerable than distracted vehicle drivers [22].

Driver Distraction Statistics

1. NHTSA estimated distracted drivers cause 25% of traffic accidents in the United States of America [25].

2. Driver distraction is reported to be responsible for more than 58% of teen crashes [26].

3. Car crash is the number one killer of teenagers in the U.S. [26].

4. Over 80% of drivers admit having hazardous behavior while driving [27].

5. Around 60,000 traffic accidents happen every year because of sleep-related problems [28].

6. Around 13% of crashes involving heavy vehicles happen due to driver tiredness [29].

7. People are as impaired when talking on the phone while driving as they are when they drive intoxicated at the legal blood-alcohol limit of 0.08% [30].

Review Solutions

Several solutions have been suggested to reduce the risk of doing secondary tasks while driving including using speech recognition, Advanced Driver Assistant Systems (ADAS), Autonomous Vehicles (AV) and driver behavior analysis [31]. In this work, the focus is on driver behavior analysis using machine learning models.

Driver behavior analysis using machine learning and deep learning models is a suggested approach to address the driver distraction problem. Different factors influence the driver's behavior such as driving context, tiredness, distraction and driver experience. Human error which can increase due to mental workloads persuaded by distraction is one of the leading causes of car crashes. Mental workloads are hard to observe and quantify, so analyzing driver behavior to distinguish normal and aggressive driving behavior can be used as a suitable approach to detect driver distraction and alarm the driver. Driver state, driver performance and the combination of both are three types of data that are used in driver behavior analysis models [29, 32, 33].

The Potential of Machine Learning

Machine learning models including neural networks outperform rule-based models in driver behavior analysis. Driving is a continuous task and there is a dependency between driving data samples, so considering this dependency can enhance the performance of driver behavior analysis methods. Hidden Markov Model (HMM), Recurrent Neural Networks (RNN) and Long Short Term Memory (LSTM) networks are three types of machine learning methods that have memory [34-36]. Including memory in the behavior analysis could provide a higher level of understanding compared to the instantaneous response in models such as feedforward neural networks. Although neural networks, including LSTM networks, don't provide insight into the structure of approximated functions, they are suitable for learning complex functions such as driving since they don't need any information about the model in advance.

Three important features of intelligence and cognition are perception, attention and sensory memory. We started with memoryless machine learning and deep learning models to estimate driver behavior using driving data. In the next step, the training process and the model’s accuracy were enhanced using recurrent models. Then, attention was added to the model to improve the computational expense and the accuracy of the model. A combination of attention and memory enhanced the model’s performance, computational expense and reduced training time significantly. In the following chapters, driver behavior estimation using a feedforward neural network, a recurrent model and an attention model will be discussed.

Conclusion

In conclusion, in this chapter, we introduced driver distraction problem, adverse effects of it on drivers and other road users and mentioned some proposed solutions for this challenge including speech recognition, Advanced Driver Assistant Systems, Autonomous Vehicles and driver behavior analysis using machine learning models. In the next chapters, the existing machine learning and deep learning approaches, which were proposed for driver behavior analysis and pros and cons of these models are mentioned. Besides, we discussed the proposed driver behavior analysis model and experiments that were conducted to create the driving dataset.

Driver behavior analysis can be used to address the driver distraction problem. Distraction, drowsiness and tiredness are dominant reasons behind car accidents. There is considerable research to understand the level of fatigue, its symptoms and its effects. Human fatigue generation is a very intricate process, which is affected by different factors in a complex relationship. Common symptoms and signs of fatigue are poor decision-making, slow reactions, reduced vigilance and poor communication [37]. These symptoms have a negative effect on driving and can lead to an accident. There are three groups of fatigue detection methods:

1. Methods based on driver state: they use driver state, especially eye movement, to detect fatigue. PERCLOS is one of these methods. It is considered a reliable determination method of driver alertness level based on the proportion of time that the driver’s eyelid is open [38].
2. Methods based on driver performance: these methods mostly use lane tracking and measuring head distance. For example, [39] proposed a method that uses response time to detect the level of sleepiness.
3. Combination of the driver performance and state: for instance, a combination of the lane changing, eye tracking and head distance. These methods enhance the sensibility and accuracy of fatigue detection [40].

Recent investigations on distraction detection approaches use Artificial Intelligence methods such as Markov Model (MM), Neural Network (NN) and Decision

Tree (DT). Some of the suggested approaches use external devices, such as eye trackers and cameras, to estimate the driver's cognition level. These models often assume that the driver's body motions such as head movement or eye gaze are a good proxy of visual attention [33, 41]. In this line of research, however, cameras or wearable sensors are used to measure driver distraction, to be adopted as a new solution, these devices must be accepted by mainstream drivers, which raises many doubts about their intrusive nature. Furthermore, cameras and sensors don't work efficiently in adverse conditions such as night or when the driver uses sunglasses, adding another challenge to general use applicability.

[42] Discussed a cognitive distraction detection model that uses a Support Vector Machine (SVM) and the combination of driver performance data (e.g. steering wheel, lane changing from the vehicle data bus) and driver state data (e.g. eye movement) as inputs of the SVM model to predict driver distraction level. Data related to drivers' states were collected by a camera and an eye tracker. The proposed system could predict distraction with 81% accuracy in a special context, namely driving on a two-lane straight highway in ideal driving condition. The effect of adverse conditions such as fog and rain as well as the type of roads on the system's performance hasn't been considered. Besides, cameras and eye trackers were used to measure driver distraction. To be adopted as a new solution, these devices must be accepted by mainstream drivers, which raises many doubts about their intrusive nature. Furthermore, cameras and sensors don't work efficiently in adverse conditions such as night or when the driver uses sunglasses, adding another challenge to general use applicability.

[43] Proposed a system that identifies car maneuvers using a Hidden Markov Model (HMM). It uses speed and steering wheel as the system's inputs to detect and prevent abnormal maneuvers. This system predicts some simple maneuvers including left turn, right turn, U-turn, roundabout, emergency brake as well as reverse. Besides, it provides a quantitative measurement of the driver's performance including good, bad and inconsistent performance. The system predicts these maneuvers and their quality accurately only using car-related data. On the other hand, it doesn't consider the effects of different contexts of driving on the driver's performance. Moreover, it can't predict abnormal maneuvers continuously since the system only is able to predict six simple maneuvers.

[44] Discussed two driver behavior prediction approaches that use HMM and driver performance including steering wheel angle, speed and braking to predict three maneuvers including left turn, right turn as well as lane changing. The first approach has a bottom-up structure. It considered isolated maneuver recognition then connected them to make a rout. The second one is a top-down method. It considered the whole route as a phrase and recognized maneuvers in it. In both approaches, the baseline for non-distracted driving is defined as driving without performing any secondary task and the system detects distracted tasks comparing driving variables to this baseline. The accuracy of these approaches is 100% for a left turn and it is high for two other maneuvers. Although they predict all predefined maneuvers accurately without using any external devices, they unable to predict complex maneuvers. Also, they didn't consider the effect of different driving contexts on the accuracy of the system.

[45] Introduces a model of driver behavior analysis based on the pedal position. The model captured normal and distracted driving behavior and used these data to train a neural network to detect driver behavior. Then the paper discussed effects of performing secondary tasks on different drivers' performance. This research shows that chosen secondary tasks have three distinct effects on driving performance: major effect, moderate effect as well as minor effect. The severity of distraction for each secondary task is not the same for all drivers. Finally, it uses SVM to detect distraction condition for each driver.

[46] Discusses the context of driving as an important factor in driver's behavior prediction and proposed a model for driving context (driving events and distraction level) estimation using a cell phone equipped with some sensors. Several data sources including GPS, weather information and some mobile sensors have been used in the proposed model. Driving context and driver distraction were entered in the phone application manually. The proposed model was trained separately using 10 different machine learning algorithms using Weak's library of Android. Five methods including IB1, J48, LMT, Multilayer perceptron and Bayesian network classified driving events with 85% accuracy. Eight methods predicted the distraction level with more than 90% accuracy. Although the accuracy of distraction detection is high for all algorithms, the features of each distraction level are not clear. The researcher observed the driver's behavior and made the decision about the distraction level, so it depends on the researcher's opinion and he only can see the manual and visual distraction, therefore this application doesn't consider all types of distraction.

[47] Introduced a method that predicts future driver behaviors based on current behaviors and past observations using a dynamic Bayesian network and Junction tree algorithm for probabilistic reference. It used instances of driving performance in three levels of distraction including no distraction, low distraction and high distraction in the diversity of SVM algorithms and compared the accuracy of them in distraction detection. The mean accuracy of each of these SVM models was 74%, so driving performance can be used to detect cognitive distraction.

Some distraction detection methods focus on the driver’s body motions and use a variety of sensors and cameras in the car to capture drivers' face and body features. [48] Recognized driver distraction using computer vision tools and active sensors Kinect. Eye gaze, head position, arm position and face expressions are inputs of a Hidden Markov Model which has been trained to recognize driver distraction type. It can detect driver distraction with 90% accuracy and type of distraction with 85% accuracy. Though the system can detect distraction and its type accurately in ideal situations, the effects of adverse conditions and the context of driving on the system's performance haven't been considered. Besides, this system uses vision tools that don’t have the same performance under different conditions. For example, if the driver has sunglasses, the eye tracker doesn’t work accurately. Therefore, the performance of the system depends on the context of driving. [49] Used a vision-based system to detect eyes off the road time to detect visual distraction. It used a camera on the steering wheel to capture eye gaze, facial features as well as the head position and combine these variables to detect eyes off the road which is the visual distraction. The accuracy of this system was 90% based on an experiment that was conducted in a real car.

Situational Calculus

An action can be defined as a function that changes the situation of a dynamic system. The concept of action is defined in at least two subareas of artificial intelligence: natural language processing and problem-solving. Case grammar is the base of the most common action defining methods in the natural language system. Each action is a set of claims about the semantic role that a noun phrase does consider the action indicated by the verb. Problem-solving needs more intricate models of action and time compare to language processing. The most persuasive theory in this area is Situational Calculus. Situational calculus is a logic formalism that is designed for reasoning about dynamic systems. It uses a set of First-order logic formulas to represent the changing scenarios in dynamic environments. Three principal elements of situational calculus are:

- An action is a function that changes the situation of the agent using a set of prerequisites of the initial state and the effects that it has on the final state.
- Fluent describes the state of the dynamic system.
- Situations: a dynamic world is described as a set of situations and a situation shows a history of action occurrences [50].

When the system is in the situation S_i and a is an action the result will be a new situation S_j. Figure 1 shows an example of a situational calculus system with four states (start, state 1, state 2 and state3) and two actions (action1 and action2).

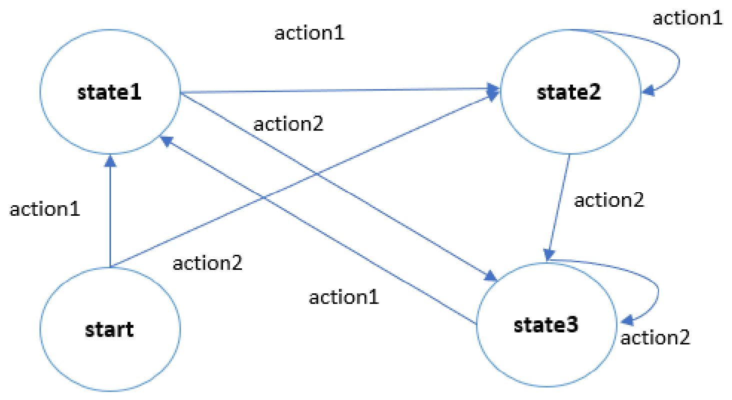


Figure 1. An Example of Situational Calculus

Following are some important features of this model:

1. While this model can describe the physical action in a static world with one agent, it doesn't work for simultaneous actions or events.
2. An action in situation calculus is defined as a change in a system, consequently, actions that involve no change and activity can't be modeled [51].
3. The main idea of situational calculus is that states are definable using actions that are required to reach them, and these states are called situations.
4. Multiple situations may refer to the same state in case several sequences of actions result in the same state.
5. Not all states have corresponding situations.
6. A state is reachable if there is a sequence of actions that can reach that state from the initial state. Unreachable states don't have a corresponding situation.

Situational calculus is an appropriate model for dynamic systems with a small set of states and actions. In systems with a large number of possible states, this model becomes computationally very expensive. Besides, in some dynamic systems, such as driving, all possible states cannot be defined in advance and the result of each action depends on many factors not only the current state. Situational Calculus formula can contain logical symbols and variables that express some general facts about the effects of actions. “Temporal Projection” is a reasoning method that is introduced by Hank and McDermott. It provides some description of the current situation, the possible actions in this situation and the effect of them, a sequence of actions that can be performed and predicting the properties of the world in the resulting situation. Describing the actions’ effects in situations calculus may raise some difficulties. One of these problems is the qualification problem. It means the successful performance of an action may depend on some qualifications. As a result, the description of actions and their effects need to include a long list of assumptions. It is better to describe each qualification by a separate axiom and consider it as a separate fact. Another problem is providing a large number of axioms listing the properties that are not affected by a specific action. For instance, when an object in state x does an action, the properties of the rest of the objects and unrelated states remain the same [52].

The foundational axioms for situations present a qualitative idea of time. It only describes the sequential action occurrence. For example, action x happens before action y in a situation. It can’t express that specific action happens at a particular time or if some actions happen simultaneously [53]. In summary, the limitations of situational calculus are:

• In situational calculus, we can’t talk about the exact time of action and how long each action takes

• There aren’t any hidden exogenous actions or unnamed events.

• Simultaneous actions can’t be modeled.

• Continuous actions can’t be described using situational calculus.

• It is only hypotheticals. It means we can’t say if an action has occurred or will occur.

• Only primitive actions can be modeled, but actions made up of other parts, such as conditionals or iterations, can’t be used [54].

Although situational calculus is a powerful model for many systems, it is not an appropriate choice for highly dynamic systems such as driving. Driver assistant systems, which control the car in specific situations, use rule-based models. They have some predefined situations and when the car is in one of these situations, they perform an action or an algorithm to move from these situations to a safe one. For example, when the distance between the car and the next car is less than a threshold, driver assistant system decreases the speed to increase the head distance. Even Driver Assistant Systems that work with a limited number of situations make wrong decisions in many cases since they don’t consider the context of driving and only follow some rules. For example, the driver may change the lane suddenly to avoid an accident, but the driver assistant system considers this action as dangerous and unappropriated action. In order to control a dynamic system like driving continuously, patterns need to be learned instead of situations.

Patterns

A pattern is a repeated design or recurring sequence in numbers, images, shape or behavior. It is an abstraction of details to help prevent the expensive process of analyzing a large number of situations. While we encounter many large and small situations everyday, we can’t process and comprehend all of them individually. Patterns are our way of processing situations because our brain has emerged to find and extract these similarities, extract patterns and use them for prediction and decision-making. We extract patterns from situations which allow us to reapply them in similar situations and avoid managing an enormous amount of data [55].

[56] Defines “Minimal Triangle” which is the representation of a pattern that has three elements including a problem, a context and a solution. He defines three groups of patterns: identical patterns, similar patterns, and equivalent patterns. Identical patterns are the same patterns mentioned in different collections. Similar patterns are patterns that look similar and act similarly. Equivalent patterns are patterns that look different but act similarly [56].

[57] Discusses the concept of informal patterns. Christopher Alexander, the author of the book, was one of the first persons that use patterns to manage complexity. His approach helps to manage complexity from an engineering perspective. While his approach wasn’t completely formal, he introduced the concept of pattern to manage complexity. In general many subjective aspects of behavior are governed by specific patterns of events that are repeated over and over again [57].

After years of driving, an experienced driver will have accumulated hundreds of driving hours and will have been through a very large number of almost unique driving situations. Although all these situations are unique, there is some similarity between them that is known as a pattern. Each situation has a large number of features. We need to choose some of these features that provide the most amount of information and abstract all situations that have close values based on these selected features to one pattern. For example, a driver may have driven in hundreds of different highways and made millions of situations, but all these situations can be converted to a small number of patterns and the driver behavior can be analyzed based on these patterns.

A pattern is opposite to chaos in predictability, especially at a lower number of occurrences. Pattern recognition, a branch of machine learning, focuses on recognition of regularization and patterns in data that is called formal patterns. Pattern recognition, machine learning, data mining and Knowledge Discovery (KDD) in databases are very close concepts and they largely overlap in their area. Machine learning and pattern recognition mostly focus on supervised learning. On the other hand, the focus of data mining and KDD is on unsupervised learning.

Machine learning focuses on increasing the recognition rate, but pattern recognition interest in formalizing, visualizing and explaining the patterns. Some common applications of pattern recognition are classification, object detection, speech recognition and computer vision segmentation. Pattern recognition algorithms goal is providing a logical answer for all possible inputs and finding the most likely match for each input, considering their statistical variation. Several pattern recognition algorithms are probabilistic. It means they determine the probability that input data is described with each label and the label with the highest probability is the most accurate answer. They associate a confidence level to their decisions also they work more efficiently in machine learning tasks with the large database since they partially or completely avoid error propagation problem [55].

Algorithmic and heuristic approaches use specific rules and calculate the exact output of the system using input data and these rules on a spontaneous basis or with simple queues, timestamps, and counters. This approach is used in driver assistant systems since the number of addressable situations and actions in each situation is low. On the other hand, if we want to monitor the driver behavior or control the situations for a long time the number of parameters and rules becomes computationally intractable, and it would not be possible to define a specific rule for each of them. In this case, we use pattern recognition and abstraction methods and estimate the label or output value. The goal of pattern recognition approaches is finding the closest pattern to the given input data.

The main concept of pattern recognition is uncertainty because of noise in data measurement and database. Statistics and probability theory are critical parts of pattern recognition. Probabilities are quantities that measure the likelihood of a given event. Probability is not related to the real world intuitively and straightforwardly like measurements such as length and weight, they only become meaningful at a large number of occurrences. There are four main definitions for probabilities:

1. Long-term frequency: the probability of an event is the long-term frequency of this event occurrence. There are two major differences between this definition and

classical definition. The probabilities can't be defined in advance by analyzing the sample space. Also, the probability can't calculate by a one-time occurrence of an event. There are no probabilities of rolling odd numbers in a single roll. The probabilities are calculated using long-term observation of the specific outcome of an event.

2. Physical tendencies of an event to happen under specific circumstances. For instance, fair dice lands with each side up with 1/6 probability. This definition considers the physical properties of an object in the long-term frequencies. It relies on the law of large numbers which links the frequency to the probabilities. Based on this rule if the same process is repeated for a long time the frequencies of the specific outcomes will get closer to the probability of this outcome. In this definition, you can talk about the probability of a single event, not just the long-term frequencies. Also, it explains the principle of indifference with the physical probabilities of an object. For instance, fair dice is a perfectly symmetric cube with uniformly distributed density.
3. Degree of belief can be a belief about the occurrence of an outcome, the truth of any random fact or truth of a hypothesis. It means how certain you the truth of statements. The probability of one shows the certain belief that a statement is true and the probability of zero shows the certain belief that something is wrong. Anything between these two tails shows a degree of uncertainty. In this definition, your belief is based on your experiences and knowledge. The principle of indifference is used to assign a probability to a specific event in which you don't have any knowledge about it, and you don't have any clue to believe that the

probability of this event is more than others. The advantage of this definition is that you can attach it to any statement and no repeatedly is required. The initial belief is your prior probability for an event, and it can be updated to the posterior probability using Bayes theory.

4. Degree of logical support is similar to the degree of belief, but it talks about the degree of logic instead of unbelief. Any probability between 0 and 1 shows a degree of uncertainty. This degree grows when the probability of getting closer to 0.5. In this definition, the probability is updated using Bayes theory but the initial probability assignment needs to be more logical compared to the degree of belief [58].

Bayes Theory

Pattern recognition methods use a combination of some statistics formula and a large number of data to detect existing patterns in the given dataset. Two parts of statistics are: calculating the probabilities and interpreting them. Frequentist interpretation determines the frequency of an occurrence. If the probability of an experiment's outcome is *p* and the experiment is repeated *n* times, we will observe this outcome *n*p* times. For frequencies, probabilities are essentially correlated to the frequency of events. The estimation of an outcome probability when the number of repetitions increases gets closer to real probability. In (1), *x* is the number of specific outcome occurrence, and *n* is the total number of event occurrence:

$$P(x) = Lim\ n \to \infty \quad x/n \ (1)$$

Although this probability definition can be useful in some cases, having an infinite trial for one event is impossible. As we increase the number of trials, the estimated gets closer to the real value but the number of trials always is finite, and we can't claim this approach determines exact probability. The alternative solution for probability calculation is using Bayes Theory. The Bayes theorem is an approach to calculate the conditional probability. It is the probability of an event occurring, given that it has some relationship to one or more other events. It computes the real probability of an event using information from some tests. Tests are flawed, it means they have some false-positive results and they are different from events. Rare events tend to have a high false-positive rate compared to more common ones.

Bayesian decision is one of the pattern recognition methods to estimate the probability of each output based on some prior assumptions. The Bayesian interpretation of probabilities is different. In Bayesian interpretation, probability is our belief in the likelihood of a specific outcome of an experiment. It uses prior knowledge in calculating the probability and measure a degree of belief instead of simple probability.

$$P(A|B) = P(B|A) * P(A)/P(B) \ (2)$$

In (2), *P(A)* is the initial degree of belief, *P(A|B)* is the posterior probability which is the degree of belief having an account for *B*. The probability of *B* is considered constant for all A_n in this equation. The posterior can be expressed with (3):

$$P(A_n \,|B) \alpha \, P(B|A_n\,) * P(A_n\,) \ (3)$$

The extended formula which is used for some portions of an event is (4):

$$P(A|B) = P(B|A_i) * P(A_j) / \sum P(B|A_j) * P(A_j) \quad (4)$$

Diagnose test is one practical application of Bayes theory. The probability of true diagnose and false classification can be computed using Bayes theory and feeding the test probabilities into the Bayes formula [59]. Systems that have a small number of possible situations and few pre-assumptions for each state can be modeled using Bayesian theory, but when there is a large number of situations this model is computationally expensive and very complicated. For example, one driving maneuver like left turn can be modeled using Bayesian theory, but it is not a suitable choice for predicting and estimating driver behavior continuously.

There are many applications in autonomous cars and modern driver cars that use classification approaches such as distraction detection, obstacle detection and object detection. The first pattern recognition classifier is linear discriminant which was introduced by Fisher. It was developed using frequentist tradition. Frequentist is based on the first definition of probabilities that says only repeatable random events have a probability. Frequentist doesn’t link the probability to any hypothesis or unknown value. In the frequentist approach, the model’s parameters are objective and unknown. They can be extracted from the collected data, and the probability of each class is estimated from these data. For instance, the mean vector and covariance matrix are the linear discriminate parameters that can be computed using the collected data. Both Bayesian and frequentist approach can be used in classification [60].

Linear Regression

Linear regression is a basic predictive analysis that predicts one dependent variable using one or multiple independent variables. A regression line has the general form as in (5). In this equation, α is the y-intercept, β is the slope of the line and ε is an error.

$$y = \alpha + \beta * x + \varepsilon \quad (5)$$

Regression might be used to estimate the strength of the independent variables' effect on the dependent variable. It also can be used to estimate the effects of one or more independent variables change on the dependent variable. Besides, it might be used for predicting trends and future values. Although linear regression works well in many applications, it has some limitations. It only looks at the linear relationship between dependent and independent variables. There are many applications that variables have a non-linear relationship. For instance, the relationship between income and age is a curve. Linear regression only considers the correlation between the mean of independent variables and the mean of a dependent variable. In some applications, extreme values are very important, and only the mean of a variable doesn't provide enough information about it.

In linear regression, all data must be independent. Spatial and temporal clustering are two examples that the data are dependent. Spatial clustering example can be clustered the grade of students from different classes, schools and restrictions. Students of each class and each school have some features in common, consequently, these variables are dependent. Temporal clustering is any clustering that considers the same object multiple times. For example, in the study, the weight of each person might be considered several times [61]. In many real-world problems, the boundaries of classes and clusters are not a straight line and linear approaches can't be used for pattern recognition or classification in these cases.

Non-linear Regression

When there is a linear correlation between independent inputs and the dependent output, the system can be modeled using linear discriminant approaches such as linear regression. In many cases, there isn't any prior knowledge about the type of correlation between inputs and outputs or the inputs are not 100% independent, so a suitable approach should be able to model both linear and non-linear correlation between data. An artificial neural network which is an extended version of linear regression is one of these approaches. Neural networks extended this model by defining some adjustable parameters and let them change during the training phase. Each basis function is a nonlinear function of a linear combination of inputs in this case. The basic neural network can be described as a series of functional transformations. In (6), which is a simple feed forward neural network, h is the activation function, w_{ji} shows the weights between layers and w_{j0} is the bias.

$$y_j = h(\sum_{n=1}^{\infty} w_{ji}(x_i) + w_{j0}) \quad (6)$$

The activation function (h) in equation 6 needs to be chosen based on the application of neural networks. In the case of regression problems, identity function can be used and in the multiclass problems, SoftMax might be used as an appropriate activation function. Binary classification problems use logistic sigmoid function [55]. The linear regression and classification models are based on a linear combination of some fixed nonlinear basis functions (g(x)). In (7), *f* is an identity function for regression, and it is a non-linear function for classification problems.

$$y(x, w) = \mathrm{f}(\sum_{j=1}^{M} w_j \ g_j(\mathrm{x})) \ (7)$$

Artificial Neural Networks

Artificial neural networks are inspired by biological neural networks. The artificial neural network learns tasks using a large number of examples without task-specific programming. An artificial neural network consists of four main parts: a processing unit, weighted interconnections between process units, learning rule that determines how to adjust the weights during the training process and activation function which works on a set of input signals to produce the output [49].

Artificial neural network (ANN) is equivalent to the human brain in several ways:

1. A neural network can obtain knowledge through the training phase
2. A neural Network can go into perpetual improvements over time, learn from its mistakes and continuously increase its own level of intelligence.
3. A neural network’s Knowledge and experience is stored in the weights of the network.

Types of Neural Networks

Feedforward Neural Network: it is a simple neural network that data travel in one direction from the input to the output. It can have one or more hidden layers and it only has the front propagate wave [62].

The Radial Basis Function Neural Network: it is a type of neural network that uses a radial basis function. The output of this network is a linear combination of the radial basis of neuron parameters and inputs. It has many applications such as time-series prediction and classification [63].

Kohonen Self Organizing Neural Network: it uses unsupervised learning to convert high dimensional input samples to low dimensional, discretize maps. It is used for producing a low dimensional view of a high dimensional data [64].

Convolutional Neural Network: it is a deep feedforward neural network that has been used successfully for image processing. In this type of neural network, input images need less preprocessing compared to other image processing algorithms since it can learn some filters which have to be done manually in other algorithms. Convolutional neural networks have several hidden layers consisting of convolutional layers, pooling layers, fully connected layers, and normalization layers [65].

Recurrent Neural Network: standard neural networks assume test and train data are independent, so when each input sample is processed the entire state of the network is lost. A recurrent neural network is a type of neural network which can send data selectively across sequence steps while processing the sequential data one sample at a time. They can model the systems with varying length sequential inputs and outputs [66].

Training Algorithms in Neural Network

Gradient descent algorithm uses the absolute error of the output, the difference between actual output and the network output, to determine the accuracy of the system. It is the simplest training algorithm in supervised learning models. The weights of the network are changed in such a manner to minimize the absolute error of the network. Equations 8 to 10 show the steps of updating the weights.

$$MSE(\theta) = 1/\mathrm{n}\sum_{i=1}^{n}\left(\mathrm{h}\theta(x^{(i)}) + y^{(i)}\right)^{2} \text{ (8)}$$

$$\partial MSE(\theta)/\partial\theta j = 2/\mathrm{n}\ \sum_{i=1}^{n}\left(\theta^{T}.x^{(i)} - y^{(i)}\right).xj^{(i)}\text{(9)}$$

$$\theta j \leftarrow \theta j - \alpha.\partial MSE/\partial\theta j\text{(10)}$$

Backpropagation algorithm is an extension of the gradient descent algorithm. It propagates the error of the system backward from the output to the input and adjusts the weights to minimize the network’s error.

There are four types of techniques which are used in different neural networks’ applications:

1. Classification: a neural network can be trained to classify the given labeled data to predefined classes. Feedforward neural networks are used for classification.
2. Clustering: a neural network can be trained to identify unique features of data and classify them using these features without any previous knowledge about them

(unlabeled data). Kohonen Self Organizing Map is one type of neural network which is used for clustering.

3. Prediction: a neural network can predict the expected output from the given input data. For example, stock prediction.
4. Association: a neural network can be trained to remember specific patterns and when a noisy pattern feeds to the network it can find the closest pattern considering that input. Image recognition is one example of this technique [67].

We conducted two experiments to produce driver-car interaction situations. In the first experiment, the effects of two different interfaces on the driver's performance were compared then the better interface was chosen based on the results of the first experiment and it was used as the second experiment's user interface. In the second experiment, distracted driving was simulated in three different driving contexts. Car and driver related data were collected in all those scenarios and collected situations were abstracted to some patterns. The patterns were used to train a neural network to predict driver behavior using car data.

Experiment 1

In this experiment, the effect of a crowded user interface on the driver distraction level was examined.

- Hypothesis: the number of icons on the user interface influences the driver response time.
- Null hypothesis: driver response time is independent of the number of icons on the user interface.

Two graphical user interfaces were designed. The first interface has six icons and each one shows a number between 1 and 6 and the second one has 30 icons and each one shows a number between 1 and 30. In case the user touches each icon, its color changes for a few seconds. An android application, which displays the user interface, was built. The application was hosted on the Android v4.4.2 based Samsung Galaxy Tab4 8.0 which was connected to the Hyper Drive simulator.

The experiment was conducted in HCaI lab using a Drive Safety Research simulator DS-600. In all, 42 volunteers participated in the experiment including undergrad and grad students of Arizona State University in the age range of 20 to 35 years old. The experiment has two scenarios: the first one is driving while using an interface with six icons and the second one is driving on the same road while using the crowded (30-icons) interface. Each volunteer was trained for 5 to 10 minutes, after that he’s asked to drive two times. During each trip, he’s asked to do four tasks, reach one icon on the interface is considered as a task. The response time was collected manually for each task.

The paired t-test was used to check the significant difference between the two groups of collected data. The two-tailed p-value for these two interfaces’ response time is 0.009 and this difference is statistically very significant. Based on the NHTSA report [68], the average time of any eye glances away from the road scene should be less than 2.0 seconds while the driver is conducting the secondary task. The average response time for the second interface was 4.42 times more than the first interface. Less than 8% of the first scenario’s response times were more than 2 seconds, safe eyes off the road time, and the maximum difference between response time and the safe response time in this scenario is

0.32 seconds. In summary, this interface is almost safe to use all the time. Around 70% of the response times in the second scenario are more than the safe time. Table 1 shows the details of the collected response times in this experiment. Adding more icons to the main screen of the car interface doesn't help the driver to find them faster. It is better to have less and bigger icons on each screen since the driver can find his desired icon in the range of the safe eyes off the road time.

Experiment 2

Traditional artificial intelligence focuses on well-defined processes with deterministic logic such as playing chess, problem solving and reasoning. This works well in a system with a relatively low number of states and possible actions, but it doesn't work well in dynamic systems with large number of situations such as driving. Driving is a dynamic and nondeterministic system in continuous environments and there is an infinite number of unique context-dependent situations in this system. Besides, each action can have varying results depending on many conditions such as the driver's skill, the context of driving and other drivers. Using Markov chain, HMM or situational calculus is not appropriate choices since this problem is not scalable to these approaches. By the mid-2000, artificial intelligence researchers noticed that the power of data in this field is more than theoretical models. The real world is dynamic and there aren't two identical samples in the dataset, which is fed to machine learning methods, so the methods choose the best fit for individual data which is fed to the trained system [36].

Table 1. Response Time Details for Experiment 1

	Tasks took <2 Seconds	Tasks took >2 seconds	Mean	Minimum response time	Max response time
6-icon	156	12	1.08	0.4	2.32
30-icon	51	117	4.78	1.54	22

An experiment was designed in HCaI lab using a driving simulator and a user interface to validate the hypothesis that driver related variables (Vh) can be predicted using car related variables (Vc). Distracted driving situations were collected in several driving contexts including normal, adverse and double adverse conditions. Two groups of data were defined:

1. Vc shows car features. It was collected from the car CAN-Bus.
2. Vh shows driver features while interacting with the car's infotainment system. It was collected manually.

The potential correlation between Vc and Vh and the possibility of accurately predict Vh from Vc were validated. The collected data preprocessed and abstracted to some patterns. Each pattern is a map between a Vc and a Vh. The neural network was chosen as an appropriate method to detect and learn distracted driving patterns since it could find a nonlinear and intricate correlation between input and output vectors besides it doesn't have any restriction on the data. A 3-layer neural network was built, and it was fed with collected data to find the correlation between Vc and Vh in each pattern. Using this trained network, driver pattern could be predicted using car related data (Vc) only which is collected automatically without using any intrusive external devices. Also, the effect of different driving contexts on the driving patterns was analyzed.

Behavior Triangulation Method

A "Behavior Triangulation Method" was defined to validate this hypothesis. This method has three parts: internal data (Vc), external data (Vh) and a neural network. Each part can be predicted using two other ones. Internal data are collected from the car system's data buses, like CAN bus. External data are collected by observing the drivers' behavior while asking them to perform distractive tasks during driving. After collecting both internal and external data, a neural network was trained to find the correlation between these two sets of variables.

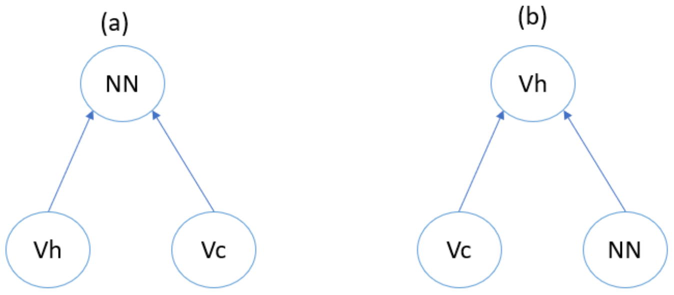


Figure 2. Behavioral Triangular Method Training Phase (b) Behavioral Triangular Method Predicting Phase

Figure 2(a) shows the training phase. In this phase, we fed both internal data (Vc) as the input of the neural network and external data (Vh) as the output and trained the network in order to find the correlation between these two groups of data. Figure 2(b) shows the predicting phase. In this phase, unseen internal data (Vc) were fed to the trained neural network and the network estimates the external data (Vc). The driver's behavior was predicted using the car related data and the trained network.

Participants and Equipment

The experiment was conducted in HCaI lab using a Drive Safety Research simulator DS-600. In this experiment, the day, night and fog modes were used. [8] Introduced a design approach called “Minimalist Design” to minimize navigation clutter, and enhance driver interaction with the infotainment system, the same design was used in this experiment. In this design, the user interface has six main categories: Car Systems, Media, Phone, Navigation, A/C, and Settings. Besides, the maximum number of icons on each screen is six. Car Systems and Navigation were simplified into Car and GPS due to the User Interface (UI) screen space of one-word constraint. The next level in each category was designed to provide the most essential features, ensuring the user will be able to reach the most common functionalities in two steps. The third, and in a few cases fourth level, is needed for interaction. This design objective is to allow the driver to navigate to each of the applications in the least number of steps possible (the menu depth). Figure 4 shows two levels of the navigation model.

Thirty volunteers with different levels of driving skills in the range of 20 to 35 years old participated in this experiment. Each participant was trained for five to ten minutes before starting the experiment. They were asked to drive in three different modes. The first mode, “ideal mode”, was in the urban area with a 45 miles-per-hour speed limit, low traffic and daylight. Figure 3(b) shows the designed urban road.

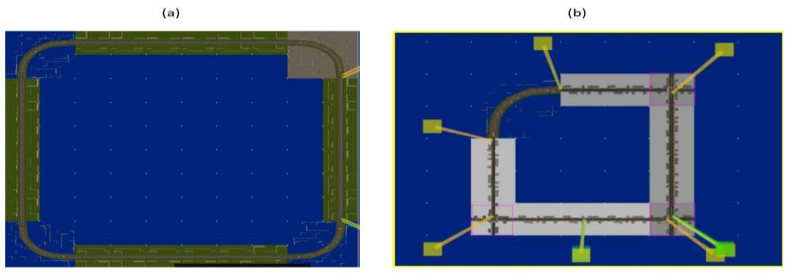


Figure 3. (a) The Designed Highway (b) The Designed Urban Area

The yellow arrow on the right down corner shows the start point. Volunteers started their trip from the start point, turn left on all intersections and finally they finish the trip and come back to the start point. The small narrow green part of the road is a highway, but the major part of the designed road is an urban area. The second mode, double adverse, was in the highway with a 65 mile per hour speed limit, night mode and low traffic.

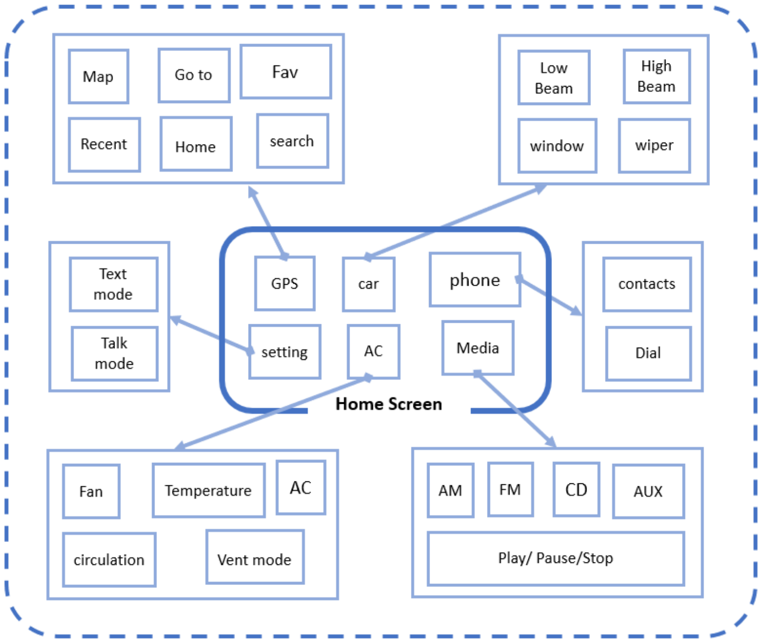


Figure 4. The Navigation Model

Figure 3 (a) illustrates the designed highway for this mode. The top right corner is the start-point of this road and the right down corner is the endpoint. The third mode, “adverse mode”, was in the urban area, low traffic and foggy weather. The designed road for ideal mode was used for this mode and the foggy weather was added to it.

In each mode, volunteers were asked to execute six tasks on the user interface. The tasks were classified into three groups consisting of two, three and four steps respectively, to complete the desired command. Each trip time was divided into six equal sections. In each section, participants were asked to execute one task. Four participants couldn’t complete all six tasks and they completed four or five tasks in one or all of their three trips since they had slow interaction with the interface and had a high number of errors during the interaction. For each task response time and the number of errors was recorded manually. The response time was defined as the total time taken for a person to manipulate the system and reach a command. Error in this system is any wrong touch while navigating or not following the traffic rules.

Data

In this experiment, each participant drove three times and each trip took three to five minutes. The simulator sampling rate was one sample per second and the average number of snapshots, which were collected by the simulator, was 250 per trip. In sum, volunteers drove 360.533 minutes in 90 trips and 18,392 car related (Vc) vectors were collected by the simulator. In each trip, the driver was supposed to do six distracting tasks while driving but five drivers couldn't finish all six tasks. In sum, 525 tasks have been done by participants during 90 trips.

Features Sensitivity

For each trip, a dataset including 53 features was collected by the simulator automatically. Nine of these features including velocity, long acceleration, headway time, headway distance, speed, lane position, steering, accelerating and brake were chosen for two reasons:

1. They were estimated to be more relevant to this problem.
2. They were validated using a t-test and they show significant differences between the three modes of driving.

The t-test compares the mean of two groups of samples and determines if the difference is significant. The larger t-test score shows more significantly different. P-value which comes with a t-test shows the probability that the sample data occurred by chance and it is between 0% and 100%. Table 2 shows the results of the t-test.

Table 2. T-Test Results

attribute	ideal vs adverse	ideal vs double-adverse	adverse vs double adverse
velocity	not significant	extremely significant	extremely significant
long acceleration	not significant	significant	significant
headway time	not significant	not quite significant	not quite significant
Headway distance	not significant	very significant	not quite significant
speed	not significant	extremely significant	extremely significant
lane position	significant	extremely significant	extremely significant
steering	not significant	extremely significant	extremely significant
accelerating	very significant	extremely significant	extremely significant
Brake	very significant	extremely significant	extremely significant

Mean and standard deviation of both populations are used in the t-test formula. Equation 11 is used to calculate the t value. Na and Nb are the numbers of samples group1 and group2 of samples. Sa^2 and Sb^2 are the variances of group1 and group2.

$$t = \frac{\text{meana} - \text{meanb}}{\sqrt{\frac{Sa^2}{Na} + \frac{Sb^2}{Nb}}} \quad (11)$$

Some of these features, such as accelerating and braking, are more relevant since the t-test shows significant differences between these values in all pairs of modes. In sum, the combination of these features shows significant differences between the three modes of driving. These results show that the context of driving has a significant effect on the driver’s performance. In this experiment, the response time and the number of errors during each task were collected manually. They were analyzed to find the effect of adverse conditions and the length of tasks on the driver’s performance while interacting with the car interface.

The average number of errors in the double adverse mode was around 6% more than the ideal mode. On the other hand, in all three modes, the average of errors and response time for 2-step tasks were almost constant. It shows that adverse conditions don’t have a significant negative effect on the accuracy of performing short tasks, but it adversely affects the driver's performance for long tasks (3-step and 4-step tasks). For instance, comparing the driver car interaction while conducting 4-step tasks in the ideal mode and double adverse mode shows the significant negative effect of adverse conditions on the accuracy of driver interaction. The average number of errors for 4-step tasks in the double adverse mode was 30% more than the ideal mode. In general, participants had at least one error in 28% of the tasks which they have done in ideal mode and it increased to 37% for the adverse and the double adverse modes.

Based on NHTSA report, drivers should perform any task on the car interface in the maximum of 6 steps and each step must be less than 2 seconds to have a safe driver car's interface interaction since more than 2 seconds of eyes off the road can lead to a fatal accident [69]. In this experiment, the average response time of 2-step tasks was four seconds. Also, it was less than six seconds for 3-step tasks and less than eight seconds for 4-step tasks besides all tasks performed in less than six steps. Although it doesn't show that all steps took less than 2 seconds, the response time of our interface is not very far from the safe range.

The collected data by the simulator were preprocessed. For each trip, a large set of Vc vectors generated, namely more than 200 vectors of 53 numbers each (an excel sheet of 53 columns). For the same trip, a small set of Vh was collected, made of six consecutive sections. Each section corresponds to one distracting task given to the driver. Eight columns from the 53 columns in the master Vc including velocity, brake, accelerating, headway distance, long accelerating, headway time, lane position and steering were chosen. Then the rows of the master Vc were grouped into six equal sections. Since each trip was divided into six parts and the driver performed one task in each part, the average of each section of the Vc has been linked to the corresponding section of the Vh.

In this experiment, a neural network classifier was used for context detection using only driving data, and function prediction to find the correlation between the driver and car variables. If driver behavior can be estimated using only car variables, the distraction would be predicted using a non-intrusive approach.

Driver Behavior Prediction

A 3-layer neural network was made from scratch with MATLAB and selected five driver-related variables (Vh) as the output of the network includes the number of errors, response time, driving mode, speed and number of steps. Also, 5 car-related variables

(Vc) were chosen as the input of this network including braking, accelerating, head distance, steering wheel, and lane changing. This network has a hidden layer with 10 neurons. The dataset was divided into two parts and used 80% of it as the training set and 20% for testing. Besides, the backpropagation was chosen as the learning algorithm. The forecasted output of each layer calculates using (12). In this equation, *w* is the weights between every two layers and *x* is the input of the network.

$$output = \sum_{i=1}^{n} \mathrm{x(i)} * \mathrm{W(i,j)} + bias \quad (12)$$

The backpropagation was used for adjusting the weights, in each iteration, the system updates the weights using the error of the output and the learning rate (13).

$$Wi\_new = Wi\_old + \partial[d(p) - y(p)] * Xi(p)(13)$$

In (13), *p* is the pattern that is fed to the network. A low learning rate makes the training more reliable, but it reduces the speed of training. On the other hand, when the learning rate is large training may not converge since changes can be so big that pass the optimal point which minimizes the loss function. To find the perfect learning rate, a balance between speed and accuracy is needed to be find. The sigmoid function was chosen out of existing activation functions (14) due to superior performance. It works well in approximating nonlinear functions.

$$g(z) = 1/(1 + \exp(-z)) \quad (14)$$

The Mean Absolute Error Percentage (MAEP) was chosen as the lost function (15). Weights were initialized with random numbers between 0 and 1, and multiply by four different multipliers (0.01,0.1,1,10). The initial random weights were multiplied by some numbers since the initial weights have a notable effect on the final accuracy, so we tried both small, medium and large initial weights and chose the one that resulted in the best performance. The best training result was achieved by 0.1 multiplier. Moreover, several learning rates were tested and 0.5 made the best result with the Z-scored dataset.

$$MAPE = 100 * 1/n(\sum_{i=1}^{n}(|Actual - forecast|/|Actual|)) \quad (15)$$

Context Detection

Based on the collected data in three modes of driving, the driving context has a significant effect on driver performance. Consequently, context detection using driving data (Vc) produces valuable information about driver performance. A classifier was built to detect the context of driving and using driving data as the input of this classifier. We chose “DNNClassifier” from TenserFlow’s estimators that encapsulated the training, evaluation and prediction phases of Neural Network as the classifier.

A 3-layer and a 4-layer neural network were built using DNNClassifier. More hidden layers didn’t have any positive effect on the accuracy of the network. The mode of driving was chosen as the target of the classifier and seven features from driving data as inputs of the network including velocity, brake, accelerating, headway distance, long accelerating, lane position and steering. Besides, 80% of data was chosen for training and 20% for testing. Adam was chosen as the network’s optimizer and ReLU as the activation function. Adam is an extension of stochastic gradient descent that computes an individual adaptive learning rate for different parameters instead of using a fixed one. ReLU is an activation function that returns the maximum of the input and zero. It is computationally less expensive than sigmoid and tanh. Besides, it had better performance compared to these activation functions.

We started with a 4-layer network and tried three learning rates including 0.01, 0.001, 0.0001 for this neural network. Besides, different numbers of neurons in hidden layers were tried ranging from 10 to 1000 for each of these learning rates. Using less than 50 neurons in each layer resulted in under-fitting and more than 500 neurons caused overfitting. The 3-layer network was trained with learning rate 0.001 that is a default learning rate for Adam optimizer and had the best performance in the 4-layer network. Also, a variety number of hidden neurons were tried in the range of 100 to 1000.

In conclusion, these hyperparameters were tried:

1. Neural networks with one and two hidden layers.
2. Three learning rates including 0.01,0.001 and 0.0001.
3. Different numbers of hidden neurons in each hidden layer ranging from 10 to 1000.
4. 50,100,150 and 200 as the number of neurons in each hidden layer.
5. Adam optimizer as an appropriate optimizer function.
6. ReLU as the activation function.

Results

Behavior Detection Results

To find the correlation between Vc and Vh the neural network was trained with both raw and normalized data (z-scored) and z-scoring showed a significant positive effect on the training accuracy. The neural network achieved an average error percentage of 11.2% with raw data. After Z-scoring, the average percent error sank to 2.6%. Also, the gap between training error and testing error decreased by 5.4%.

The weight multiplier had a drastic impact on the training error of raw data. The higher the weight multiplier, the greater the variance of percent error. Figure 5 (a) shows the relationship between some iterations in the training phase (Test Number) and errors (%Error) for 3 different weight multipliers for raw data. Very low weight multiplier results in almost constant test error and increasing training iteration didn't change the accuracy of the system. Increasing the weight multiplier gradually increased the effect of it in the system accuracy, so 1.0 was chosen as the weight multiplier.

After data normalization, the effect of different weight multipliers on the z-scored data were tested. A combination of z-scoring and the use of a multiplier for weight initialization had considerable effects on improving error percentage. The test findings for the Z-scored data showed that the best performance and the most consistent one was 0.001 as weight multiplier, which had an error of around 2.46 %. It also shows that the lowest testing percentage error was a weight of 0.1 that resulted in the lowest percentage error of 2.03%. Using 0.01 as the weight multiplier resulted in a somewhat consistent percentage error, but it had some flexibility and wasn't as consistent as the 0.001 weight. The 0.01 weight ended up being more consistent than the 0.001 weight multiplier when the testing was done with the z-scored data. Finally, 0.1 was chosen since using this weight multipliers resulted in the lowest testing error.

Learning Rate Effect

Another variable that influenced the error percentage was the learning rate. We started with a very low learning rate and increased it gradually. When 0.2 was tried as the learning rate the results stayed mostly stagnant. At a 0.5 learning rate, the results became more variant. This trend continued as the learning rate increased, as evidenced by the 0.7 and 0.9 learning rates having the greatest variance. The effect of using a low learning rate and low weight multiplier was almost the same. Figure 5(b) shows the effect of different learning rates on the error in the first 3 iterations (Test1, Test2 and Test3) when Z-scored data were used. A learning rate of 0.5 seemed to produce the most consistently low results. In summary, the combination of Z-scored data, learning rate 0.5 and weight multiplier 0.1 produced the minimum training error equal to 1.38 and testing error 4.25. These results validate the hypothesis that driver variables can be predicted using car variables with high accuracy.

Context Detection Results

Two classifiers were built. The first one has two hidden layers with a low number of neurons, and three different learning rates were tried for this neural network including 0.01, 0.001 and 0.0001. The results show the lowest learning rate (0.01) resulted in the biggest gap between the training accuracy and testing accuracy compared to other learning rates which means using a large learning rate for this network caused overfitting. Although the gap between the train and test result is almost the same for two other learning rates (0.001 and 0.0001), the learning rate 0.001 produced better training and test results compared to the learning rate 0.0001. A variety number of neurons were tried for hidden layers from 10 to 1000. The best results were achieved when the number of hidden layers' neurons was between 50 and 200 and the neural network with 100 neurons in each hidden layer produced the best training and testing results. Table 3 shows the results of the classification with a 4-layer neural network. The second column shows the number of neurons in hidden layers and the last two columns show the percentage of training and testing accuracy. The neural network with learning rate 0.001, two hidden layers and 100 neurons in each layer had 96.42% training and 93.33 testing accuracy which is the best-achieved result.

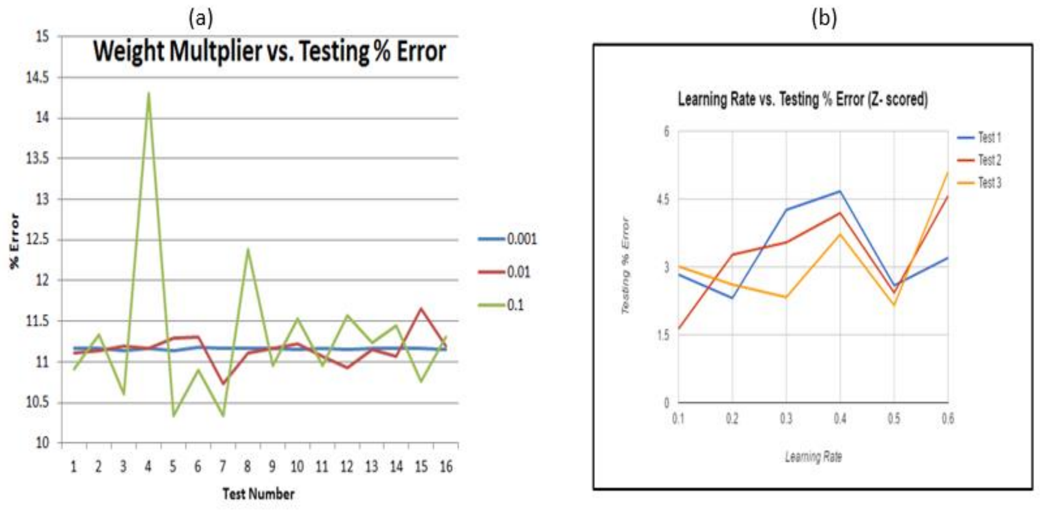


Figure 5. (a) The Effect of Weight Multiplier on the Error (b) The Effect of Learning Rate on the Error

The second classifier was built with one hidden layer and tried different numbers of hidden neurons between 50 and 1000 for this layer. Besides, 0.001 was chosen as the learning rate of this 3-layer network since it is recommended as the best learning rate for the estimator that was used in the classifier (Adam estimator). Also, using this learning rate in the first classifier resulted in better accuracy compared to other learning rates.

Table 3. Results of the 4-layer Neural Network Classifier

Learning rate	Hidden layers	Train accuracy	Test accuracy
0.0001	(50,50)	85.68	80.95
0.001	(50,50)	94.27	93.33
0.01	(50,50)	92.8	84
0.0001	(100,100)	87.35	85.71
0.001	(100,100)	96.42	93.33
0.01	(100,100)	89	77
0.0001	(150,150)	86.63	82.85
0.001	(150,150)	94.98	86.6
0.01	(150,150)	90.69	82.85
0.0001	(200,200)	92.38	90.69
0.001	(200,200)	91.88	83.8
0.01	(200,200)	91.4	82.85

A low number of neurons in the hidden layer leads to a big gap between training and testing accuracy. As the hidden neurons were increased, this gap decreased. The Minimum gap was for 700 as well as 800 neurons. Also, the maximum accuracy was achieved by the classifier with 800 neurons in the hidden layer. The training and testing accuracy of this network was 98.8 and 95.23 percent. Table 4 shows the results of the second classifier. The first column shows the number of neurons in the hidden layer and the last two columns show the percentage of training and testing accuracy.

Table 4. The Results of the 3-layer Classifier

hidden neurons	Training accuracy	Testing accuracy
200	95.46	84.76
500	95.47	88.57
700	97.37	94.28
800	98.8	95.23

In summary, a two-part experiment was conducted to induce varying levels of distractive task sets to drivers and collected corresponding data patterns, then used both data sets to train the network. First, with the triangulation method, the trained network was reused to predict distraction patterns when fed with the data patterns from part 1. With the neural network in reverse mode, it was able to accurately predict driver behavior when fed with system variables alone. After that, two classifiers were built using TenserFlow and Vc vectors were fed as the input of them in order to classify them to three contexts of driving (driving modes) since driving context has a significant effect on driving data (Vc), so if the driving context can be detected, it provides valuable information about driver behavior. Also, the experiment was conducted in three different modes of driving to validate the hypothesis about the effect of adverse conditions on the driver's performance.

The desire to entertain drivers often leads to designing a very complex and highly interactive infotainment system. Complex infotainment system can adversely affect driving and cause driver distraction. Most of the available works of literature and proposed solutions focus on two main areas: Advanced Driver Assistant Systems (ADAS) and Autonomous Vehicles (AV). ADAS provides modern cars with many useful sensors that collect a large number of driving data. Thousands of events and actions are captured and transmitted across complex computer-like architecture inside the car that enables full digital control of the vehicle such as drive-by-wire, steering and brake. Autonomous vehicles utilize complex algorithms to automate the driving process. They benefit from the fully digitized car by controlling all car systems automatically. While they are deployed on the road, they will not be for all of us any time soon since they have limitations and challenges yet to be solved.

ADAS systems work well in predefined situations that were defined and developed by system developers. They use a variety of sensors to collect driving data and monitor the environment according to specific algorithms. In case the car is in one of those predefined dangerous situations, ADAS alarms the driver or takes control of the car for a short period. Although these systems work efficiently in some cases, they don’t consider the context of driving in their dangerous situations detection. Besides, they can’t detect all the things that can go wrong in traffic. Moreover, they rely on sensors, so if the sensors don’t work perfectly due to some adverse conditions, their accuracy decreases significantly.

Autonomous cars use many sensors such as cameras, Lidars as well as radars to observe the environment and collect driving data to extract and learn driving patterns from the collected data. They have full control of the car except in some emergency situations. Using Autonomous Vehicles (AV) is another solution to refine car safety. Based on the NHTSA report, 94% of serious car crashes are due to human error [70]. AVs have the potential to inflate car safety and lessen fatal car crashes by removing human error. They face some challenges such as dealing with driver cars, road unsuitable infrastructures and cybersecurity. Similar to ADAS, autonomous cars rely on the performance of their sensors, so in some adverse conditions, such as rainy days, their performance may degrade. Moreover, before they reach the fifth level of automation, drivers need to be ready to take the control of the car in emergency situations but sometimes the driver is distracted, and the take overtime is not fast enough to avoid an accident. In the long run, when these challenges will be solved, autonomous cars boost car safety and decrease commuting time and cost [71].

There is a gap between ADAS and autonomous cars since we moved from a rule-based system that mostly just monitors selected attributes in the driver behavior to an automated system that controls the car continuously most of the time. In this work, an intelligent co-driver system was built that works between ADAS and autonomous vehicles. The data collected with existing ADAS is used to capture and recognize risky driving behavior and patterns using artificial intelligence. Unlike autonomous vehicles, intelligent vehicles utilize the existing sensors and will not require any of the advanced hardware used in autonomous cars.

Deep Learning

One key concept in the pattern recognition field is uncertainty which arises because of raw data sets and noise on measurements. Probability theory provides a framework for quantification of uncertainty and makes one of the foundations of the pattern recognition field. A combination of decision making and probability theory allows optimal decision making using available data. These classification and regression models such as Decision Tree (DT) work efficiently for low dimensions datasets. Regular machine learning algorithms don't process all raw data since they can have many features with no valuable information, so for decades area expertise and engineers needed to design a feature extractor to convert raw data to suitable feature vectors or internal representations. Machine learning algorithms could use these preprocessed data for pattern recognition or classification.

Defining some basic functions and using parameters that change during the training phase of a machine learning algorithm is an appropriate approach for high dimensions and large datasets. Feedforward neural networks are one of the most successful models of this type in the field of pattern recognition. Since they are non-parametric, and they don't have any restrictions on the data type, data distribution and the number of the system's inputs and outputs. Besides, they can learn complex nonlinear systems.

Deep learning methods are representation learning methods. Representation learning is a collection of methods that lets a machine use raw data as input and automatically extracts the representations that are needed for pattern recognition and classification. Deep learning is a computational model with multiple hidden processing layers to learn the complex patterns of data using several levels of abstraction. They can extract intricated structures inside a huge data set using the backpropagation method which specifies how the internal parameters should be changed [34]. Deep networks break down tasks in a way that makes machine assistants, which was impossible with other methods, possible [36].

Gradient Descent

To build a supervised learning system, a large set of labeled data needs to be collected. In the training phase, the system produces an output vector which shows the score for each category. The goal of this phase is maximizing the score of the desired category compared to other categories. Learning algorithms' goal is finding the global minima of the cost function. Figure 6 (a) shows the steps of finding the cost function's global minima.

The gradient descent process is formulaic. The entire dataset is used in each step forward and it computes the cost into account in total. After that, a wholesale propagation of errors goes backward from the output layer to the input layer. Gradient descent is susceptible to a local minimum because the whole dataset is used in the weight adjustment in each iteration. Converging to a local minima does not allow for the more exploration of the function when it reaches a local minimum to move beyond it. Figure 6(b) shows a function that gradient descent may be stuck in its local minima.

Stochastic gradient descent is a proposed solution to this problem. It adjusts the weights after every input data. After each data instance forward pass the weights are updated. It makes the gradient descent more volatile, but it can escape the local minima and find the global minima of the cost function. Mini-batch gradient descent was proposed as a solution between these two methods. It is faster than stochastic gradient descent, and it still allows for more fluctuation to pass the local minima. This method considers a small batch of data error in each iteration and updates the weights using this error [72]. Mini-batch gradient descent was chosen for all deep learning models.

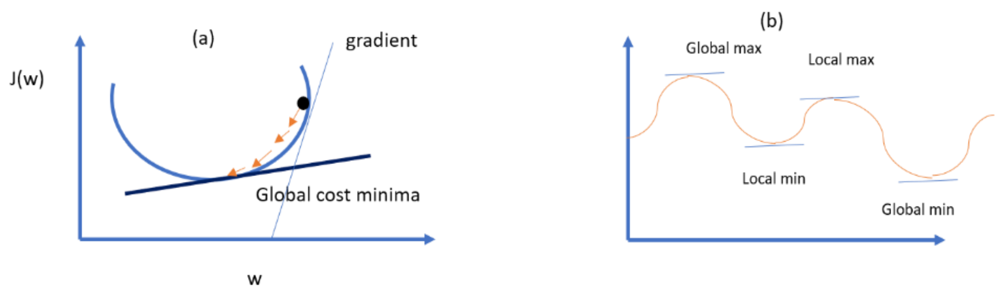


Figure 6. Cost Function

There are different cost functions that can be used in calculating the error of a system. Equations 16 to 20 show the steps of gradient descent using mean square error cost function for a 3-layer neural network.

$$out_{hi} = activationfunction\left(\left(\sum_{n=1}^{k} Wn * xn\right) + b\right)\ (calculate\ the\ \ output\ of\ hidden\ layer)\ (16)$$

$$out_{Oi} = activationfunction\left(\left(\sum_{n=1}^{k} Wn * out_{hi}\, n\right) + b\right)\ (calculate\ the\ \ output\ )(17)$$

$$Error_{total} = \sum_{n=1}^{k} \frac{1}{2} * (target - output)^2\ \ (calculate\ the\ total\ error)(18)$$

The backward pass:

$$\frac{ðError_{total}}{ðW_i} = \frac{ðError_{total}}{ðout_i} * \frac{ðout_i}{ðnet_i} * \frac{ðnet_i}{ðW_i} (find\ the\ derivative)\ (19)$$

$$newW_i = W_i - \alpha \frac{ðError_{total}}{ðW_i} (\ updating\ the\ weights)\ (20)$$

Activation Function

Although gradient descent in shallow neural networks works great, it doesn’t work efficiently in deep networks since the gradient often gets smaller as the backpropagation algorithms go toward lower layers. One proposed solution is using non-saturating activation functions. A sigmoid function was used as the activation function of the neural network in chapter 2. Sigmoid is the most popular activation function in shallow neural networks but it saturates fast in deep networks since when the neuron's activation saturates at either tail, 0 or 1, the gradient is almost zero and no updates will happen in the network’s weights. Besides, a sigmoid function is non-zero centered, so the gradient in each step is all positive or all negative and it can cause zig-zagging dynamic in updating the weights [73].

ReLU is an activation function that is commonly used in deep learning (Figure 7). The output of this function is x for x > 0 and zero for the negative inputs. It doesn’t have the saturating problem of sigmoid, but it has another problem called dying. When the output of the network is biased to the negative values, the ReLU output is zero, so no error backpropagates through the network and the ReLU died.

Leaky ReLU is a type of ReLU which is non-saturated, and it was suggested for deep networks. The output of this function is always non-zero and it doesn’t have the dying problem [74]. Randomized Leaky ReLU which is proposed in Kaggle NDSB competition is a randomized version of Leaky ReLU. The distinguishable feature of this function is that the number which multiplies to the negative inputs is a random value from the uniform distribution [75]. ReLU was used as the activation function of the deep neural networks and LSTM networks since it has a better performance compared to other functions.

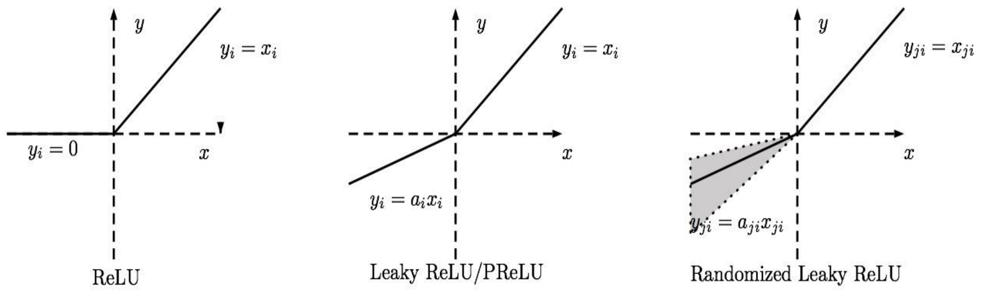


Figure 7. Different Types of ReLU

Learning Rate

The learning rate in the gradient descent process determines the length of steps toward the minimum point of the cost function. There are several suggested approaches for learning rate initialization. One approach is trying different learning rates and comparing the speed and accuracy of the training phase then choosing the best one. Adapting the learning rate for gradient descent optimization procedure may increase performance and decrease training time. This is called adaptive learning rates or learning rate annealing. This method is a balance between high speed of large learning rate and good performance of low learning rate. Two easy ways to do learning rate decay is:

1. Decrease the learning rate gradually based on the epoch. An epoch is one complete presentation of data set to be learned in machine learning algorithms.

2. Decrease the learning rate using punctuated large drops at particular epochs [76, 77].

[78] Discusses an optimizer method with an adaptive learning rate called Adam optimizer. It computes adaptive learning rates for different parameters individually using estimates of the gradient's first and second moments. This method combined the advantages of two optimizers: **AdaGrad** and **RMSProp**. It stores an exponentially decaying average of past squared gradients like **Adadelta** and **RMSprop** methods. Besides, it keeps an exponentially decaying average of past gradients like momentum. This optimizer was used in deep networks and recurrent networks since the adaptive learning rate improved the accuracy of the networks compared to optimizers with fix learning rate.

Intelligent Co-Driver System

Advancements in many research areas have been made possible by deep learning. The most remarkable feature of deep neural networks or in general neural network is that they can learn very complex systems which are computationally extremely expensive or even impossible for other machine learning approaches and rule-based systems. The key concept in neural networks is patterns. They are fed with a large number of collected situations then they extract the existing complex patterns from these situations.

The HCaI lab focuses on bridging the gap between autonomous cars and ADAS, with the "Intelligent Proactive Co-driver System". It is a multi-disciplinary methodology that spans engineering and computer science, as well as cognitive science and automotive engineering. The goal is to create a user-centered engineering design for future cars grounded in the study of how drivers behave in modern cars. The intelligent Co-driver system can be designed for both autonomous and traditional vehicles using machine learning and deep learning approaches. Deep neural networks have cumulative learning capacities to manage complex real-life situations. The Intelligent Co-driver system is designed to selectively feed data from onboard system to a deep neural network to learn situational driving patterns and recognize driver distraction with high confidence.

Experiment 3 was conducted to collect a large number of driving situations in different driving contexts and made a master dataset using these collected situations. After that, the master dataset was preprocessed and the existing features in this dataset were validated. The most significant ones that provide the most information about each situation was chosen. The pure collected situations were abstracted to a dataset of driving patterns. These patterns are fed to deep networks to detect the context of driving and predict the driver behavior using driving data, which were produced by the car simulator.

Related Works

Deep learning is a branch of machine learning that has great flexibility to learn intricate patterns in real-world data by nesting hierarchy of concepts each of which is related to simpler concepts. In deep learning more abstract representation computes in terms of less abstract ones. Deep neural networks learn categories incrementally using their hidden layers and they are powered by an enormous amount of data. Using deep networks decreases the need for domain expertise and dedicated feature selection since the model learns complex patterns in an incremental approach by itself [36, 65]. Different types of deep networks are used in autonomous vehicles, driver distraction detection and driver behavior recognition applications such as lane detection, pedestrian detection and driver distraction detection, and they outperform other machine learning methods [79-81]. The following are some of these car safety applications.

[81] Introduces a new framework for driver distraction detection. The proposed framework uses a dataset of images with a diversity of driver’s ethnicity and gender, different camera position and illumination conditions and a pre-trained convolutional neural network VGG-19. The developed system includes three parts. The first part is a convolutional deep neural network that is used for high abstracted feature extraction. The second part is a max-pooling layer that decreases the dimension of features, and the last part includes six fully connected layers and a SoftMax layer. The best test accuracy was 90% and the average accuracy was 80% per class. This system uses a multi-class classification process to map the input observation to the driver’s state.

[79] Discusses a lane detection method using deep neural networks. Two of the most successful deep learning applications are: recognizing object classes and learning representative image features. Most existing neural networks use low and middle-level indication without considering any spatial structures, but for some application visual cues which spatially are distributed over an image are important. This research discusses two types of neural networks including a multitask deep convolutional network, that detects the target considering the region of interest, and a recurrent neural network which is adapted to be used as structured visual detection. The first network provides auxiliary geometric information to help model the lane structures, and the recurrent neural network

is used to detect the lane boundaries without any prior knowledge. Both proposed networks outperform conventional detectors in practical traffic scenes.

[80] Talks about a new deep learning technique for categorizing the driver’s gaze. Sequences of the driver’s gaze zone show the driver's behavior. Drowsiness, distraction, and focusing can be detected by analyzing the driver’s eye gaze. Nine gaze zones were defined, and a convolutional neural network was used to categorize the driver’s eye gaze from given face detected images. The images are captured using MOSSE tracker with a single camera and fed to the convolutional neural network. The accuracy of this approach is 95% on average.

[82] Introduces an object detection approach that exploits context, segmentation, and location before performing 3D object detection. It generates some candidate class-specific object proposals. Then fed them into a standard convolutional neural network to gain high-quality object detection. They proposed an approach that places the objects in 3D using this fact that the objects are on the ground plane. It scores each candidate box using contextual information, some intuitive encoding semantic segmentation, size, and location priors and typical object shape. They reached 80.6% accuracy using 200 proposals.

[83] Talks about a new driving decision approach called direct perception approach. There are two main paradigms for vision-based driverless cars’ systems. The first one is the mediated perception approach which parses the entire scene to make a driving decision. The second one is the behavior reflex approach which maps an input image to a driving action using regressor. The proposed approach in this paper is in between these

two major paradigms. In this approach, the input image was mapped to a set of a compact description of the scene, and it enables a simple controller to drive autonomously. A convolutional neural network was trained using 12 hours of human driving recording in a video game and the results show that the system can work well in a diversity of virtual environments.

Despite the significant progress of autonomous cars driven by deep neural networks, like other software, deep networks sometimes show unexpected or incorrect behavior which can lead to fatal collisions. [84] Talks about a new tool called a deep test. It automatically detects the erroneous behavior of deep neural networks driven cars. This new tool is designed to generate test cases that show the changes in real-world driving conditions such as rain and fog. The Deep test tool found many erroneous driver behaviors in different real-world driving conditions.

Experiment 3

This experiment aims to conduct an empirical study to observe and model drivers in HCaI lab using a simulated modern car environment provided by ASU to collect a large body of driving patterns under different conditions; also, to construct a large database of composite driving contexts to be used for training of artificial intelligence systems. Furthermore, long-term memory was added to it using LSTM networks, which will greatly increase the system intelligence. Using deep learning methods, the new AI approach will allow future cars to learn the driver’s behavior, cognitive abilities and limitations as well as their risk-awareness while driving.

Participants and Equipment

The experiment was conducted in HCaI lab using a Drive Safety Research simulator DS-600. 132 volunteers participated in the simulator experiment by taking a 45-minute simulated drive that resembles two groups of driving: distracted and non-distracted. Each group consists of four separate drive sessions: ideal daylight drive, adverse condition 1 (night drive), adverse condition 2 (Fog), and dual adverse condition (night and Fog) (Table 5). Volunteers were chosen from undergraduate and graduate students of Arizona State University, in the range of 20 to 35 years old with at least 2 years of driving experience.

The total of these eight sessions were used to generate a large number of different drive situations that were recorded and analyzed using deep neural networks, which is already trained to recognize distraction on a medium set of simulated driving conditions. Moving to a large set of data increased the network's intelligence level. We also expanded it to a recurrent neural network (RNN) to provide a long-term memory, hence have a higher level of intelligence in recognizing complex driving patterns and extracting better anomalies.

Table 5. Eight Driving Contexts

Mode	Distracted	Non-distracted
Day	Day Distracted	Day Non-distracted
Night	Night Distracted	Night Non-distracted
Fog	Fog Distracted	Fog Non-distracted
Fog &night	Fog & night Distracted	Fog & night Non-distracted

An urban road was designed with three left turns and one curve highway part, so both highway and urban road driving data collected during the experiment. Each volunteer drove eight trips with different scenarios and each trip took 4 to 5 minutes. The maximum data sample rate was set at 60 samples per second, and the drivers were asked to follow traffic rules and drive as realistic as possible. All scenarios were driven on the same road. In order to add unpredictability and reduce any learning effect, several unpredictable events were added to the trip design and the order of scenarios were chosen randomly. In distracted scenarios, drivers were asked to execute some tasks on the infotainment system during each trip, while in non-distracted scenarios, drivers were asked to pay full attention to the road and it was used as the baseline since it shows safe and focused driving behavior.

In distracted scenarios, drivers were asked to execute some tasks on the interface continuously. A task was defined as reaching a specific application on the interface. The interface's applications divided into three groups based on the number of navigation steps that are needed to reach them from the main screen and they are called 2-step, 3-step and 4-step applications. We tried to put an equal time interval between every two tasks. It means that when the driver finished a task, we waited for a few seconds before asking him/her to do the next task, hence eliminating any cross-task dependencies. The driver's behavior while interacting with the car infotainment system was observed and analyzed. Four features were defined for each interaction including the number of errors that the driver did during the task, response time which shows the time range from the moment the driver was asked to do a task and the time that he/she finished it, the mode of driving which is the name of current driving scenario and the number of navigation steps that the driver needs to pass on the interface to complete the task. The goal is detecting if the driver is in a distracted mode or not, and if he/she is distracted, what are the features of the interaction that caused the distraction. In distracted modes, these four features were collected for each command for 35 volunteers and only the number of errors in each trip for the rest of the volunteers (97 volunteers).

In each trip, distracting tasks were chosen randomly from these three groups, and drivers had different sequences of distracting tasks on each trip. An equal number of tasks were chosen from each group. Furthermore, different orders of scenarios were chosen for each driver. For example, one driver started with the night distracted mode and another one started with the day non-distracted mode. If all the drivers follow the same sequence, after one or two trips they will be more familiar with the road and driving in the simulator, so they drive better. This learning effect was distributed among all scenarios by using different sequences of trips in order to minimize the effect of learning on the final data. One of the goals is detecting the effect of the context of driving on driver performance when all other factors are similar between driving scenarios.

Collected Dataset

On average 19000 snapshots were collected by the simulator on each trip. Each snapshot includes 53 features. For 35 volunteers, both car-related data and driver-related data were collected. Figure 8 shows the details of the collected data by the simulator and manually for these drivers. Only the simulator data were collected for 97 volunteers. Figure 9 shows the details of the collected data by the simulator for these volunteers.

Although the master data set was built with all 53 collected features and the maximum possible sample rate, all these data samples won’t be used since there isn’t considerable change in 1/60 second for this work also some of the features have a large number of missing value or they don’t show significant differences between different conducted scenarios. The master data were preprocessed, and every 20 rows were compressed to one row. Also, ten features were chosen from 53 existing features in the master data. After preprocessing, around 850,000 patterns were extracted in two datasets including 35 volunteers’ data and 97 volunteers’ data. These extracted patterns were used to train the intelligent Co-driver system.

While deep learning and neural networks rely heavily on the availability of training data, very few databases suitable for AI systems are available. The main goal of this experiment is creating an appropriate master dataset of driving patterns that can be used in a variety of deep learning and machine learning algorithms. This master dataset can be preprocessed based on the application and algorithms that we are going to train using these data. Existing driving-related datasets has been searched and most of them were related to accidents and they don’t provide any information about normal driving or distracted driving that didn’t lead to accidents. We need a dataset that covers both distracted and non-distracted data in different driving contexts, so a dataset was created using the car simulator in HCaI lab. Following is some existing data set related to driving patterns that can be used in artificial intelligence systems.

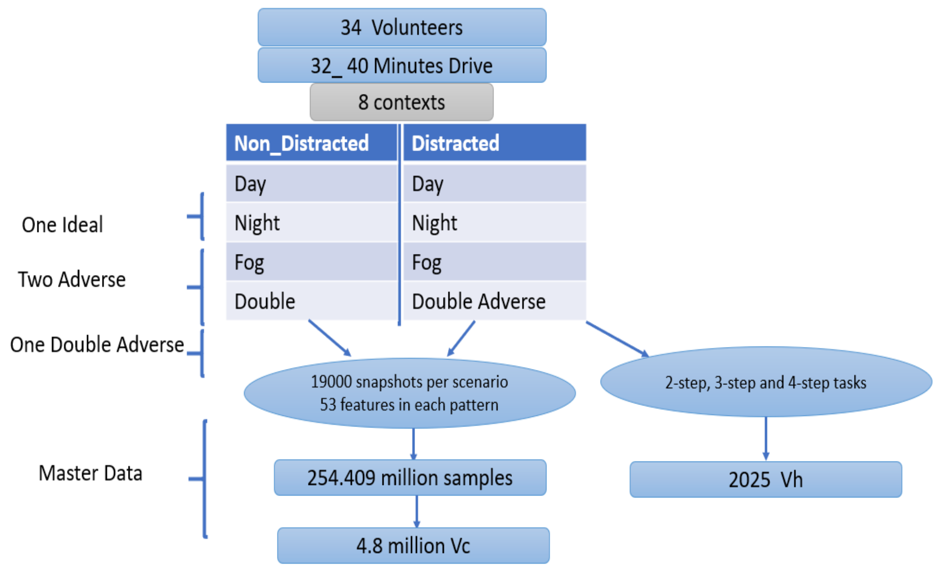


Figure 8. Vc and Vh for 35 Volunteers

NHTSA (National Highway Traffic Safety Administration) provided a survey about vehicle crashes causation. It made a dataset consisting of on-scene, in-depth multidisciplinary investigations of 6,949 crashes that occurred between 2005 and 2007. This dataset provides more details compared to the police report about the driver, car and traffic characteristics associated with driver distracted related accidents. According to this dataset and survey, internal distraction was the main reason for 11 % of crashes that were studied [85].

Department for transport under OGL (Open Government License) in the United Kingdom provided a road safety dataset about the circumstances of road accident injuries from 1979, the types of cars (make and model) involved in the accident and the consequential casualties. The statistics include personal injury accidents on public roads which reported to the police and recorded, using the STATS19 accident reporting form.

This dataset updates annually and available in the CSV format for each year separately [86].

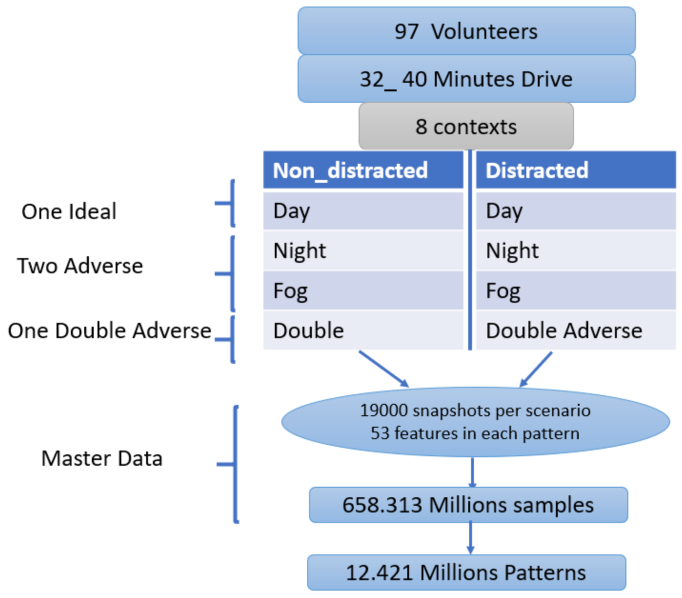


Figure 9. Vc Vectors for 97 Volunteers

The National Highway Traffic Safety Administration (NHTSA) has researched driver distraction considering both behavioral and vehicle safety countermeasures to understand and mitigate vehicle crashes caused by driver distraction. They have examined the driver distraction factors using NHTSA datasets. The databases include codes that accommodate participation in secondary tasks and cognitive distraction. FARS collects “Driver-Related Factors” which are coded in a way that can capture distractions situations like the use of cell phones and navigation systems. This variable also captures careless and inattentive driving from cognitive distractions such as daydreaming [87].

FARS, which is used in NHTSA research, is the Fatality Analysis Reporting System. The program collects data for analysis of traffic safety crashes. The goal of the program is to identify problems and evaluating countermeasures to reduce injuries and property damage to car crashes. The FARS dataset contains descriptions, in a standard format, of each reported fatal crash. To qualify for inclusion, a crash must involve a motor vehicle traveling a traffic-way customarily open to the public and resulting in the death of a person (occupant of a vehicle or a non-motorist) within 30 days of the crash. Each crash has more than 100 coded data elements that characterize the crash, vehicles, and people involved. The specific data elements may be changed each year in order to adapt to the user needs, vehicle characteristics and highway safety emphasis areas [88].

The Nation lost 35,092 people in crashes on U.S. roadways during 2015, an increase from 32,744 in 2014. The 7.2% increase is the largest percentage increase in nearly 50 years. NHTSA research provides a brief overview of the 2015 fatal crash picture using data from the National Highway Traffic Safety Administration’s Fatality Analysis Reporting System (FARS), a census of motor vehicle fatal traffic crashes in the 50 States, the District of Columbia, and Puerto Rico, and the National Automotive Sampling System General Estimates System (NASS GES), a nationally representative sample of police reported motor vehicle crashes. Information is presented in the following sections.

■ Overall Trends ■ Fatality and Injury Rates ■ Police-Reported Crashes ■ Change in Fatality Composition ■ Fatality and Injury Changes by Person Type ■ Inside Versus Outside the Vehicle ■ Fatal Crash Types ■ Human Choices ■ Alcohol-Impaired-Driving

Fatalities and Drivers ■ Restraint Use and Time of Day ■ Economic and Other Indicators [89].

Data Analysis

Driving is a dynamic and context-dependent activity. A variety of factors such as weather, time of day, the type of road and the driver’s skill can influence the driver's performance. The hypothesis is that driving in adverse conditions increases the number of driver’s errors consequently it increases driver distraction. In Experiment 3, driving data were collected in eight scenarios including four distracted and four non-distracted ones. For 97 volunteers, the driver’s errors in each scenario were collected. Considering all contexts of driving including “Day”, “Night”, “Fog” as well as “Fog and Night” on average 79.72% of drivers had more errors in a distracted mode compared to a non-distracted mode. Besides, on average 11.58% of drivers had the same number of errors in both distracted and non-distracted modes of the same context of driving. Based on our observation during the experiment, adverse conditions had less effect on the drivers with high driving skills, and they could control both primary tasks of driving and the secondary task that they were asked to perform. These drivers gave the primary task the higher priority and perform the navigation steps of each task with more delay between every two steps.

It was hard for most drivers to handle these situations and interact with the car interface continuously. They were asked to put the primary task of driving as the first priority and put a delay between every two navigation steps of each task but interacting with the car interface adversely affected around 80% of drivers.

Drivers' Errors

Based on the results of the experiment, it was found that the context of driving and driver distraction adversely affect driver performance. On average, 7.5 percent of drivers had fewer errors in a distracted scenario compared to a non-distracted one. These drivers were considered as outliers in the dataset. Besides, the drivers drove eight times on the same road in different driving contexts, so gradually they became more familiar with the road and the errors that were related to unfamiliarity with the road decreased. Table 6 shows a comparison between driving errors in distracted scenarios and non-distracted ones.

Table 6. Compare Distracted and Non-Distracted Modes' Errors

Distracted Vs Non-distracted	Day	Night	Fog	Fog & Night
Less Errors	8.69%	11.6%	5.8%	4.3%
Equal Errors	11.59%	13.04%	14.5%	7.2%
More Errors	79.72%	75.36%	79.7%	88.5%

Table 7 shows the effect of driving context on the performance of a non-distracted driver. Around 52.33 percent of drivers made a less or equal number of errors in the day mode, which was the baseline, compared to adverse modes (Fog, Night) and double adverse mode (Fog and night). Double adverse mode influenced the driver more adversely compared to adverse modes since 58% of drivers had more mistakes compared to the day mode but in adverse modes, on average 50% of drivers had more mistakes than the day mode. In the fog non-distracted scenario 56% of drivers made fewer errors than day non-distracted scenarios. One assumption is the driver has more confidence in the day mode, and he pays less attention to the road and driving task, but in the foggy mode the driver sight distance is less than other modes, so he needs to drive more carefully with less speed. Moreover, the learning effect might influence the number of errors of drivers in adverse and double adverse modes. In non-distracted scenarios, drivers were asked to only focus on the road and driving, so they had a chance to learn the details of the road faster than distracted scenarios consequently make fewer mistakes in their last trips compared to earlier ones.

Table 7. Compare Errors in Non-Distracted Modes

Non-Distracted	Day vs night	Day vs Fog	Day vs Fog Night
Equal errors	21%	13%	21%
Day less errors	34%	31%	37%
Day more errors	45%	56%	42%

Table 8 shows the effect of the context of driving on the driver's performance in distracted scenarios. Around 52% of drivers made a less or equal number of errors in the day mode, compared to adverse and double adverse modes. The double adverse mode had more adverse effects compared to adverse modes since 54.5% of drivers had more mistakes than day mode but in adverse modes, on average 50.73% of drivers had more mistakes compared to day mode. In adverse modes, night mode had more effect on the driver's performance than the foggy mode in distracted scenarios like the non-distracted ones.

Table 8. Compare Errors in Distracted Modes

Distracted modes	Day vs night	Day vs Fog	Day vs Fog Night
Equal errors	17.4%	20.28%	14.5%
Day less errors	34.78%	29%	50%
Day more errors	47.82%	43%	25.5%

Data Preparation

The simulator generated 14,914,947 data vectors during 776 trips that have been driven by the first 97 volunteers. Twenty six undergrad and grad students in 5 teams helped in data collection and processing over a one-year span under the supervision of Dr. Gaffar. The dataset was cleaned and preprocessed. After that, the dataset was compressed and every 20 vectors were replaced with the average of them since the maximum sampling rate was set and using all collected data was computationally expensive besides it doesn't provide much more information compared to a compressed dataset. The final dataset has 621,088 data vectors and each vector has 53 features. More significant features that provide more information about each driving context were chosen by using feature validation methods.

A correlation heat map was used to show the correlation between some of these features. A heat map (or heatmap) is a graphical representation of data where the individual values contained in a matrix are represented as colors and a correlation heat map is a matrix that is used to find the dependence between some variables at the same time. Values in the range (-0.3, -1) and (1,0.3) show weak correlation, values in the range of (-0.5, -0.3) and (0.3, 0.5) show a moderate correlation and values in the range (-1,1) show no correlation [90]. We can combine or delete one of two features that are strongly connected to improve the performance of the system. Figure 10 shows the correlation heat map for 10 features in the preprocessed dataset.

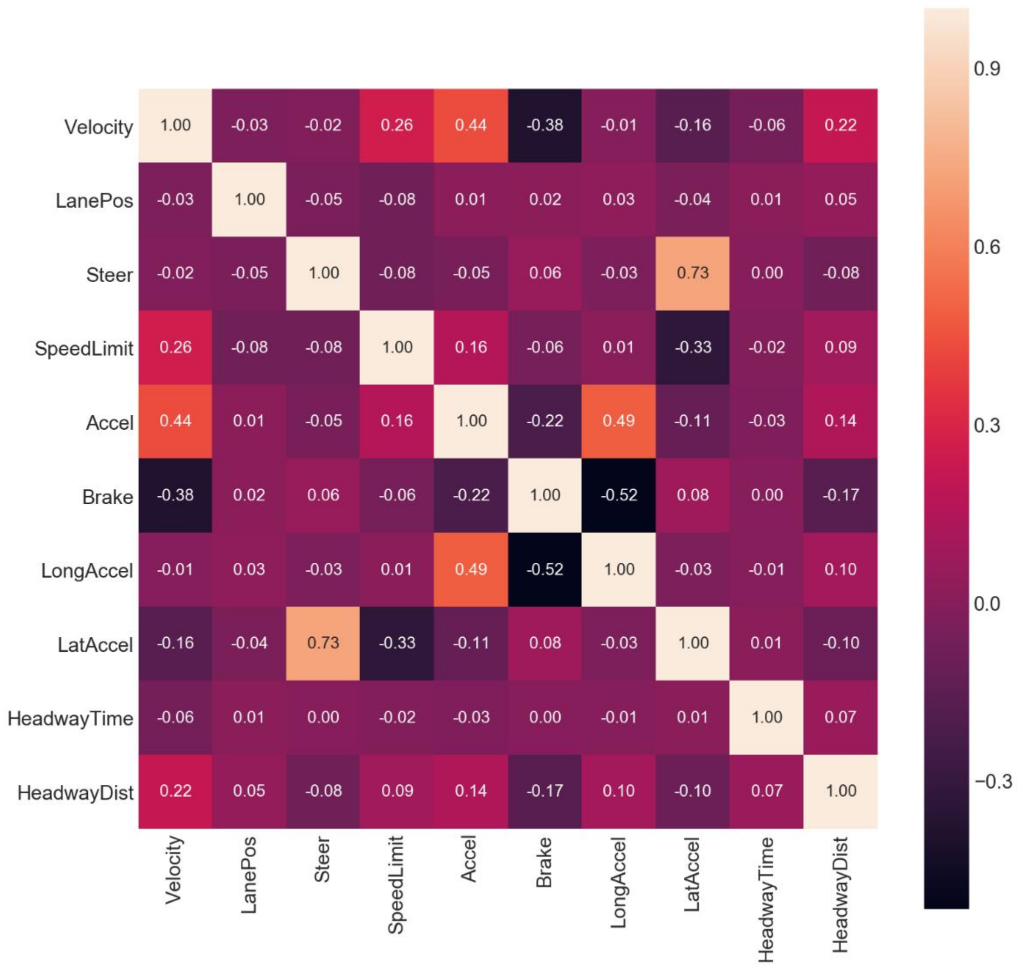


Figure 10. Selected Features' Heatmap

There are 10 features and 45 unique pairs of features in this map (Figure 10), and 69% of them show no correlation, 17.7% of them show weak correlation, 13.3% of them show moderate correlation. Heatmap was used to find a set of independent features from 53 features of our dataset. We use the set of independent features as the inputs of our intelligent system.

T-Test

A paired t-test was used to validate the features of the collected dataset. Table 9 shows the results of a paired t-test for non-distracted scenarios. The day non-distracted scenario has been considered as the baseline and other non-distracted scenarios have been compared with this scenario. Accelerating and braking show extremely significant difference between all non-distracted scenarios, but Speed doesn't show any significant difference between them. Other features show significant or extremely significant difference between two or three scenarios.

A p-paired t-test was conducted for distracted modes considering the distracted day scenario as the baseline and Table10 shows the results. Velocity, braking, headway time and headway distance show an extremely significant difference between distracted scenarios. Longitude accelerating shows non-significant difference between these scenarios and the rest of the selected features show significant or extremely significant differences among 2 or 3 distracted scenarios. Based on Tables 9 and 10, the combination of ten selected features is an appropriate choice for a machine learning method since each of them shows significant difference between at least two scenarios.

A paired t-test was conducted between distracted and non-distracted scenarios of each context of driving. Table 11 shows the results of t-test. Lane position and accelerating show an extremely significant difference between distracted and non-distracted driving in all contexts. Besides, other features show a significant difference between distracted and non-distracted driving in at least one context of driving.

Table 9. Paired T-Test for Non-distracted Modes

	Day non-distracted vs. night non-distracted	Day non-distracted vs. fog non-distracted	Day non-distracted vs. Fognight non-distracted
velocity	extremely significant	extremely significant	not significant
lane position	not significant	not significant	extremely significant
steering	significant	not significant	very significant
speed limit	not significant	not significant	not significant
accelerating	extremely significant	extremely significant	extremely significant
brake	extremely significant	extremely significant	extremely significant
Longitude accel	extremely significant	not significant	not significant
Lateral accel	significant	not significant	not significant
headway time	significant	not significant	not significant
Headway distance	extremely significant	extremely significant	not significant

Table 10. Paired T-Test for Distracted Modes

	Day distracted vs. night distracted	Day distracted vs. fog distracted	Day distracted vs. Fognight distracted
velocity	extremely significant	extremely significant	extremely significant
lane position	not significant	extremely significant	extremely significant
steering	extremely significant	extremely significant	not significant
speed limit	not significant	not significant	not significant
accelerating	extremely significant	extremely significant	not significant
brake	extremely significant	extremely significant	extremely significant
Longitude accel	not significant	not significant	not significant
Lateral accel	very significant	not significant	not significant
headway time	extremely significant	extremely significant	extremely significant
Headway dist	extremely significant	extremely significant	extremely significant

In conclusion, based on the heatmap results and t-tests, the features that were chosen can be used as inputs of a machine learning classification or regression method to distinguish these eight driving scenarios. The selected features were used as inputs of a neural network classifier in order to detect the driver status (distracted, non-distracted) and driving context (ideal, adverse and double adverse) using only driving related data, which were collected by the car simulator. A deep neural network was built, and was fed with collected data in Experiment 3 to detect the context of driving using driving data.

The first master dataset was made with 12.5 million data vectors that were collected during 776 trips of 97 volunteers. It includes the simulator data and the mode of driving. This dataset was used for detecting the context of driving using only driving data (Figure 9). The second master dataset was built using collected Vc and Vh vectors during 280 trips of 35 volunteers. Figure 8 shows the details of Vc and Vh data related to these volunteers. In this part, 5.3 million Vc vectors and 2025 Vh vectors were collected during 280 trips. The First dataset uses for context detection and the second one uses for recurrent neural network.

Context Detection Using Deep Neural Network

Ten features of the master dataset were selected using t-tests and they were used as inputs of our deep network. The target of this network is the context of driving. A data set of driving data vectors was built with 10 features and called it Vc. A Multilayer neural network (MLP) was built to detect the context of driving using driving-related data.

Table 11. Paired T-Test Between Distracted and Non-Distracted Modes

	Day distract vs. day non-distract	Night distract vs. night non-distract	Fog distract vs. fog non-distract	Fog night distract vs. Fognight non-distract
velocity	not significant	extremely significant	significant	very significant
lane position	extremely significant	extremely significant	extremely significant	extremely significant
steering	extremely significant	very significant	not significant	extremely significant
speed limit	not significant	not significant	significant	not significant
accelerate	extremely significant	extremely significant	extremely significant	extremely significant
brake	very significant	not significant	not significant	extremely significant
Longitude accel	not significant	extremely significant	not significant	not significant
Lateral accel	not significant	not significant	not significant	not significant
headway time	extremely significant	not significant	significant	not significant
Headway dist	extremely significant	extremely significant	extremely significant	not significant

A small network was used to investigate the effect of different learning rates on the system's accuracy. A deep neural network with four hidden layers and 50 neurons in each layer was built and variety of learning rates were tried in the range of (0.0001, 0.01). 0.0001 was chosen as the most suitable learning rate. After that, three activation functions including sigmoid, tanh and ReLU were tested. The network with ReLU as the activation function reached the best accuracy besides, this function is recommended as an appropriate activation function for deep networks since it can prevent the gradient vanishing and exploding problem. The Multilayers classifier of TensorFlow was used to build the deep classifier and chose Adam optimizer that uses adaptive learning rate as the network's optimizer since it is computationally efficient, it well suited for problems with a large dataset and it achieves good results fast in the field of deep learning. Figure 11 shows the classifier.

Training was started with four hidden layers network and 50 neurons in each hidden layer. The system stopped learning after 200 epochs since the change in the network's loss was less than 0.0001 for two consecutive epochs and the final accuracy of the system was 9%. Each time 100 neurons were added to each layer and it was trained with the same learning rate and activation function. Gradually the number of epochs, that the system could continue learning, increased consequently the system's performance improved. Networks with 500 to 1000 neurons in each layer had the best performance. Besides, 1,000 neurons were used for each hidden layer.

The initial value of parameters is another factor that has a significant effect on the network's performance. If the parameters of the network are saved after training and are used as initial values, the network learns patterns faster and more accurately. In the next section, the results of the different networks, that were built and trained, are discussed.

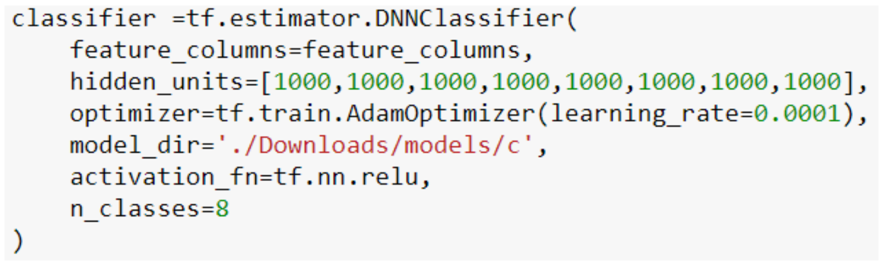

```
classifier =tf.estimator.DNNClassifier(
    feature_columns=feature_columns,
    hidden_units=[1000,1000,1000,1000,1000,1000,1000,1000],
    optimizer=tf.train.AdamOptimizer(learning_rate=0.0001),
    model_dir='./Downloads/models/c',
    activation_fn=tf.nn.relu,
    n_classes=8
)
```

Figure 11. TensorFlow DNN Classifier

Results

Learning Rate Effect

The first variable that was tuned is the learning rate. The learning rate is one of the most important parameters of deep neural networks since it has a significant effect on the speed of training and the accuracy of the network. A large learning rate increases the training speed, but it may cause the network to pass the optimum point. On the other hand, a small learning rate makes the training slow but increases the chance of finding the optimum point [36].

A network was built with five hidden layers and 50 neurons in each hidden layer and tried 0.01, 0.001 and 0.0001 as the network's learning rate while keeping all other parameters constant. Table 12 shows the accuracy of these networks after 10,000, 30,000 and 100,000 epochs. The next four figures compare the effect of different learning rates on the average loss of the network. Using 0.01 as the learning rate resulted in a low accurate network. The train and test accuracy are very close. Increasing the number of epochs didn't improve the system accuracy, so the system is underfitted. Figure 12 shows the average loss and time per step for this network. The average loss didn’t decrease, and it just fluctuated in a specific range. This learning rate is not an appropriate choice for this network.

Figure 13 shows the loss of the network with the learning rate of 0.001. The loss of the network decreased gradually as the number of epochs were increased. The system’s accuracy increased from 25.77% in 10,000 epochs to 49.37% after 100,000 epochs. Learning rate 0.0001 had a similar effect on the performance of the network. The network continued learning since the accuracy increased gradually and the gap between the train and test accuracy doesn’t show overfitting.

Table 12 . Network with 5 Hidden Layers and 50 Hidden Neurons in Each Layer

Epochs	Learning rate	Train accuracy	Test accuracy
10,000	0.01	12. 67	12.52
10,000	0.001	25.77	24.97
10,000	0.0001	22.15	21.3
30,000	0.01	13.13	13.27
30,000	0.001	35.81	32.39
30,000	0.0001	28.77	27.18
100,000	0.01	13	12.91
100,000	0.001	49.37	39.92
100,000	0.0001	40.55	34.96

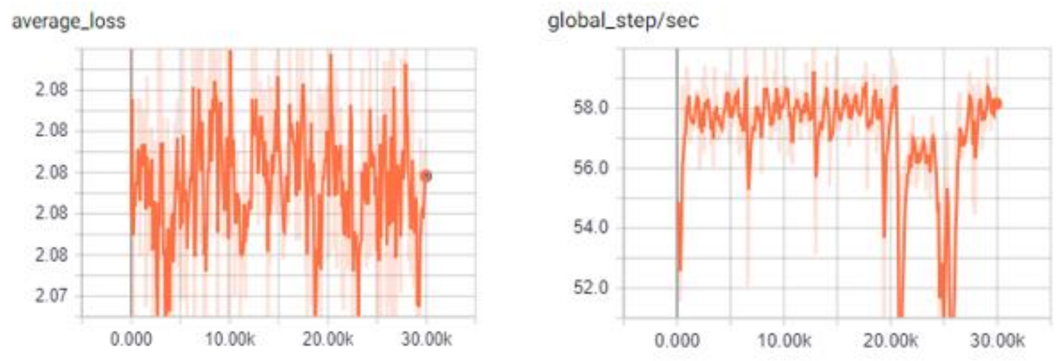


Figure 12. Network with 5 Hidden Layers, 50 Hidden Neurons and Learning Rate 0.01

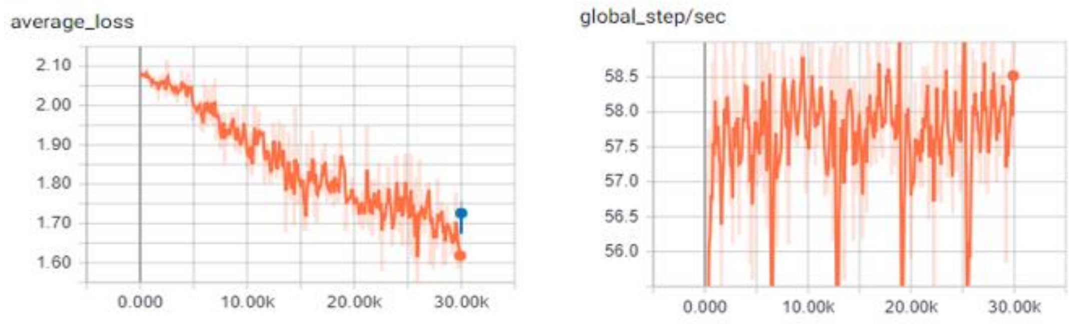


Figure 13. Network with 5 Hidden Layers, 50 Hidden Neurons and Learning Rate 0.001

Figure 14 shows the loss and time of each step in the training phase for the network with the learning rate 0.0001. Although this network continued learning with these two learning rates. The gap between the train and test error increased after 30,000 epochs, so the system will be overfitted if we increase the number of epochs. With learning rate 0.001, the gap was 1% after 10,000 epochs which increased to 10% after 100,000 epochs. It means on average every 10,000 epochs increased this gap by 1%. The network with 0.0001 learning rate trained a little slower than the previous one and the gap after increased from 1% for training with 10,000 epochs to 6% after 100,000 epochs, but this network will be overfitted before reaching an acceptable accuracy similar to the previous one.

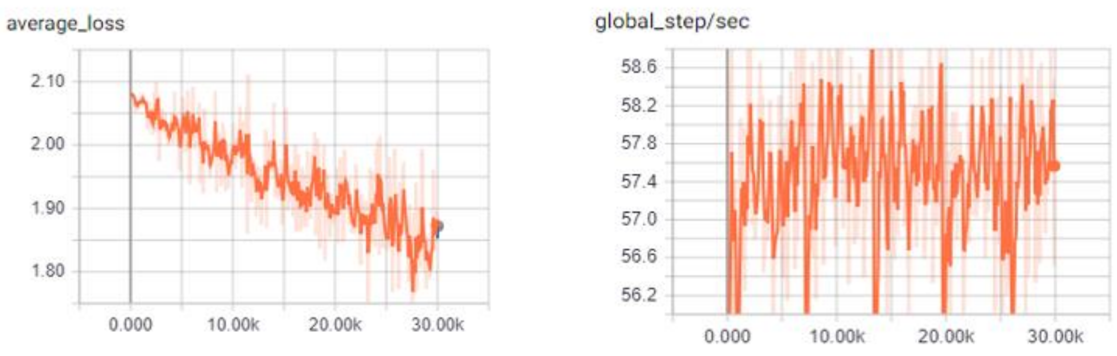


Figure 14. Network with 5 hidden layers, 50 hidden neurons and learning rate 0.0001

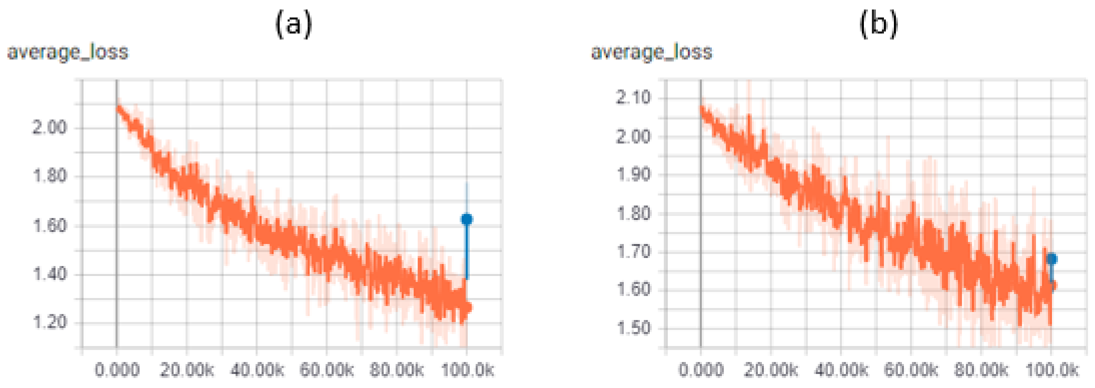


Figure 15. (a) Loss for Learning Rate 0.00, (b) Loss for Learning Rate 0.0001

In sum, the networks with learning range of 0.001 and 0.0001 have the same range of global-step time, but this variable has more variance in the network with 0.01 learning rate. Figure 15 (a) shows the loss of the network with the learning rate 0.001 and Figure 15(b) shows the loss of the network with learning rate 0.0001 after 100,000 epochs. They show an almost similar pattern of loss change. Based on the achieved results, 0.001 and 0.0001 were chosen as appropriate learning rates for the network.

The Number of Hidden Neurons

The number of hidden neurons is one of the parameters of neural networks. In theory, a neural network with one hidden layer can learn any function but learning complex functions with shallow networks are very slow and inefficient. Increasing the number of hidden layers and the hidden neurons in each layer increase the training speed and final accuracy of the network [67]. The low number of neurons in hidden layers made the training very slow and the network couldn't reach good accuracy even after 100,000 epochs. The same network was trained with 5000 hidden neurons instead of 50 neurons and chose the learning rate 0.0001. The accuracy of this network was compared to the previous one.

After 30,000 epochs training the training accuracy was 42.71% and the test accuracy was 36.56% in the network with 5000 neurons. Compared to the similar network with 50 hidden neurons it trained faster since it reached 42.71 % train accuracy and 36.56% test accuracy after 30,000 epochs, but the previous network reached 28.47% train and 34% test accuracy after 100,000 epochs. The network with 5000 hidden neurons had better performance in much fewer epochs. Figure 16 compared the average-loss of these networks, Figure 16 (a) is the loss of network with 5000 hidden neurons and Figure 16 (b) is the loss of network with 50 hidden neurons.

The accuracy of the network with a large number of hidden neurons is 19% higher than the same network with a low number of hidden neurons after 30,000 epochs. In summary, although the networks with a low number of neurons are computationally less expensive, they need more training epochs to reach a specific accuracy compared to networks with a large number of neurons. A balance between these two factors need to be found. We started from 100 neurons and added between 100 and 200 neurons to each layer then trained the network for some epochs. Between 500 and 1000 neurons in each layer resulted in the best performance.

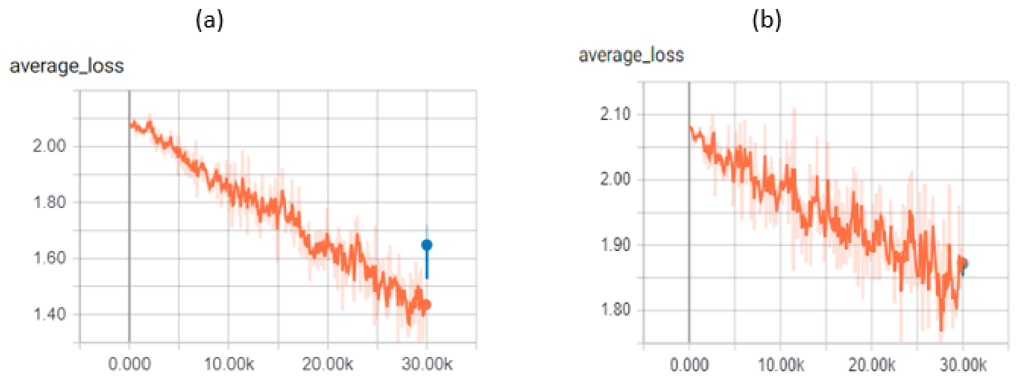


Figure 16. (a) The Average Loss for 5000 Hidden Neurons (b) The Average Loss for 50 Hidden Neurons

The Number of Hidden Layers

Deep networks can learn complex patterns faster and more accurately than shallow networks. The number of hidden layers is one of the hyperparameters of the neural network that need to be tuned based on the application and the number of input samples [65]. The previous network was used, and the number of hidden layers was increased to six. The new network was trained for 30,000 epochs to test the effect of the number of layers on the accuracy of the network. The training accuracy was 44.05% and the test accuracy was 37.39%. The training accuracy increased by 1.34% and the test accuracy increased by 0.83% compared to the network with five hidden layers. The number of connections in the network with six hidden layers is 25,000,000 more than the network with five hidden layers, and the difference between the accuracy of these networks can be because of different initial weights, so there wasn't significant improvement by adding one hidden layer. More hidden layers needed to be added each time to find significant improvement. Figure 17 compares these two networks.

Weight Initialization

Initial weights influence the final accuracy of the neural networks. If two neural networks were built and trained with similar parameters, they could have different accuracy since the weights of each network initialize randomly and might one of them has more appropriate initial weights and trains more accurately. In chapter 2, a neural network with one hidden layer was built and different initial weights were tried for that network [91]. The results show the significant effect of appropriate initial weights on the accuracy of the system. In deep networks, suitable initial parameters can make the training faster and more accurate [92].

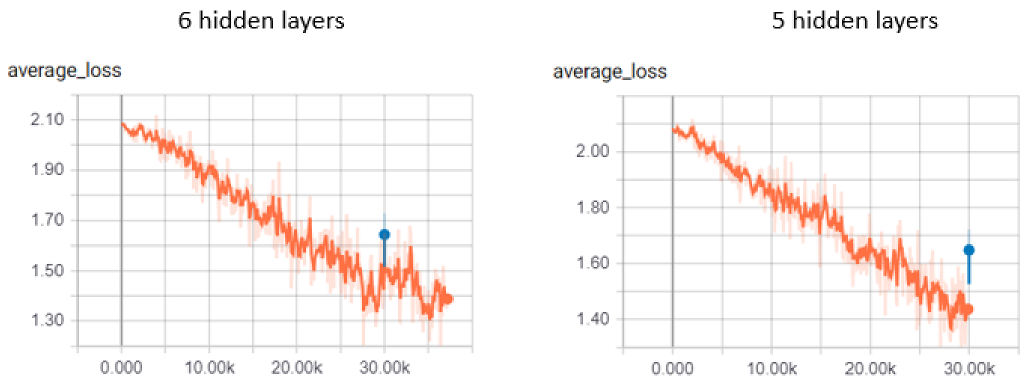


Figure 17. Average Loss of Networks with 5 and 6 Hidden Layers and 5000 Hidden Neurons

Random initial weights were tried in training deep networks with a variety of combinations of hyperparameters. Some of them overfitted fast and some of them trained very slow and stuck in the local minima. The models were trained, and their parameters were saved. The parameters of the pre-trained network were used as the initial values of the network.

Using Pretrained Networks

Some networks were trained with a variety of hidden layers and the best achieved value for each parameter were chosen. For each hidden layer, 1000 hidden neurons were chosen, and 0.0001 was used as the learning rate of them. These networks have the same hyperparameters and only the number of hidden layers is different between them. Table 13 shows a summary of the best results that were achieved.

The 13-layer network with 1,000 neurons reached the best result and other networks' results are very close to each other. A 10-layer network was chosen and fewer and more neurons for this network were tried to investigate the effect of the size of layers on the accuracy of the system. The results show a 10-layers network with 1000 neurons has a better performance compared to a network with a larger hidden layer and it is computationally less expensive. Besides, when the number of hidden neurons was reduced to half the training accuracy reduced 8% but the gap between train and test accuracy didn't change. In the next section, the details of the training process for each network were explained. Table 13 shows the details of these networks and the final accuracy of them.

Pretrained Networks Results

In the previous section, the effect of the number of hidden layers and using pre-trained networks on the accuracy of the network were discussed. In this section, the accuracy and training process of the networks of the previous section are discussed.

Table 13. Deep Neural Networks Results

hidden layers	Number connections	Number neurons	Train accuracy	Test accuracy	Learning rate
7	6,011,000	1000	87.52	82.42	0.0001
9	8,011,000	1000	88.08	86.48	0.0001
10	9,011,000	1000	88.21	86.63	0.0001
11	10,011,000	1000	85.34	83.58	0.0001
13	12,011,000	1000	94.06	93.1	0.0001
14	13,011,000	1000	81.79	79.83	0.0001
10	2,255,500	500	80.78	78.13	0.0001
10	36,022,000	2000	86.01	84.98	0.0001

Network with 10 Hidden Layers

The training was started with 10 hidden layers network, 1000 neurons in each hidden layer and learning rate 0.0001. Figure 18 and Figure 19 show the accuracy and the average loss of this network over 400,000 epochs. In this network, the accuracy increased gradually in the first 300,000 epochs. After that, for 50,000 the graph doesn't show any significant change and in the last 50,000 epochs, the accuracy fluctuated in the range (80, 90).

The loss graph (Figure 19) has two curves, the red one is the training loss and the blue one is the validation lost. The maximum gap between the average train and test loss is 0.7 which is achieved in the 100,000th epoch. This gap decreased gradually and reached 0.2

in the 300,000th epochs. It is the point that the maximum accuracy was achieved. During the last 50,000 epochs this gap is less than 0.1, so both accuracy and the gap between test and train error are minimized as the result the system is well trained. The number of hidden layers was reduced, and the results of these networks were compared. In sum, 10 hidden layers network was considered as the baseline. After that, the number of hidden layers was increased and decreased to find the most appropriate number of hidden layers.

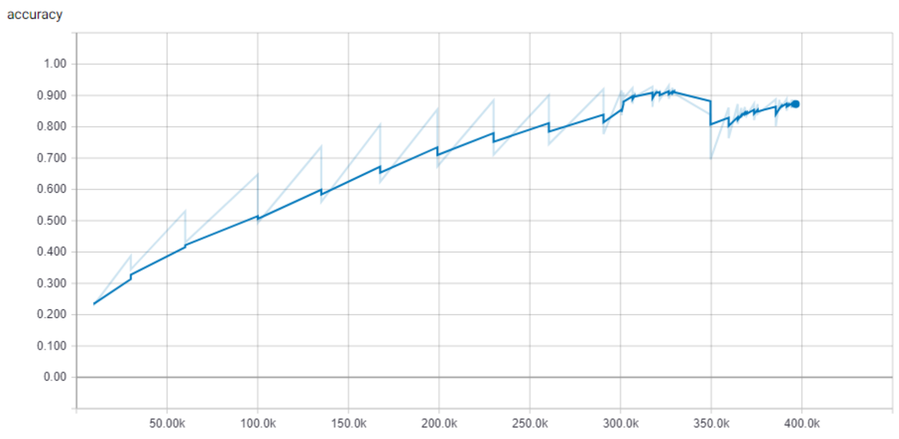


Figure 18. Accuracy of the Network with 10 Hidden Layers

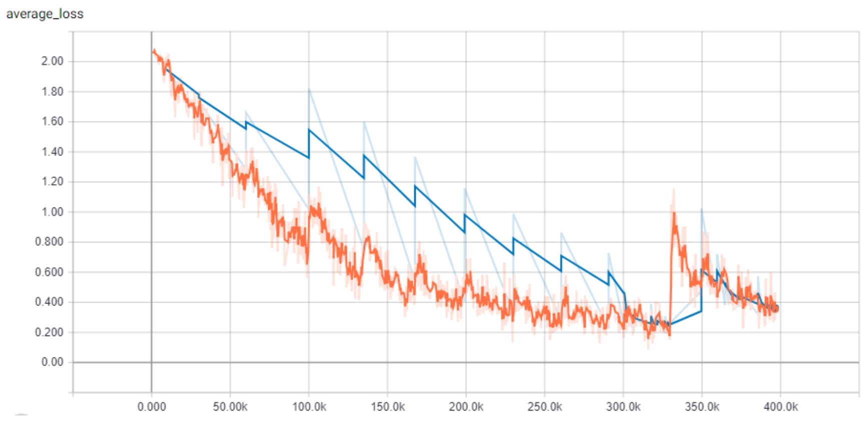


Figure 19. Average Loss of Train and Test for 10 Hidden Layers Network

Network with 9 Hidden Layers

Each hidden layer in 10 hidden layers network adds 1,000,000 connections so deleting one hidden layer can decrease each epoch runtime significantly. A network with 9 hidden layers was tested. The model has 1000 hidden neurons in each layer and the learning rate is 0.0001 similar to the previous network. Figure 20 shows the accuracy of this network. The accuracy graph is like the 10 hidden layers network in the first 300,000 epochs. In both networks, the accuracy increased except for few epochs that the accuracy decreased compared to the previous epoch. In the previous network, the accuracy changed from 20% to 90% and in this network, it changed from 30% to 80% in the first 300,000 epochs. In the last 100,000 epochs, the accuracy of the 9 hidden layers network hasn't changed for a longer period and showed less change compared to the 10 hidden layers network but the final results of them are similar.

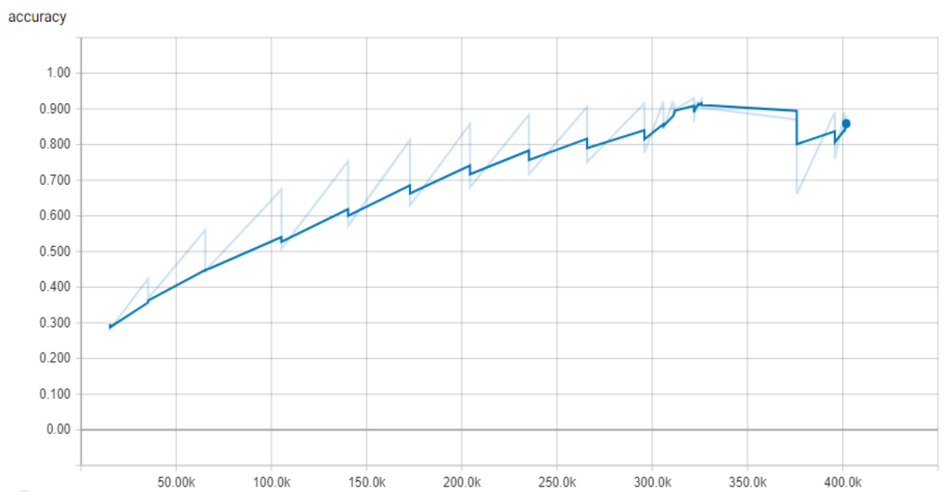


Figure 20. Accuracy of 9 Hidden Layers Network

Figure 21 shows the average train and validation loss of this network. The gap between train and test loss in the first 300,000 epochs is less than 10 hidden layers network in most of the epochs, but in the last 50,000 epochs, the 10 hidden layers network shows more stable behavior and has the minimum gap between test and train error. In conclusion, the results of the networks with 10 and 9 hidden layers are very close and we didn’t find significant differences in their performance, so the number of hidden layers was reduced to seven and a network with 7 hidden layers, 1000 hidden neurons and learning rate 0.0001 was built to compare the result of a network with much fewer connections to the 10 hidden layers network performance.

Network with 7 Hidden Layers

Figure 22 shows the accuracy of the 7 hidden layers network. The increasing pattern of accuracy plot in the first 300,000 epochs is similar to 10 hidden layers network but the accuracy decreased 20% suddenly in one epoch then in the last 100,000 epochs it increased gradually around 10% and reached 82% accuracy. Decreasing the number of hidden layers adversely affected the system accuracy and the gap between test and train accuracy. This gap in 10 hidden layers network was 1.58%, it increased to 2.4 for 9 hidden layers network and it reached 5.15 in 7 hidden layers network. Figure 23 shows the average loss of 7 hidden layers network. In the last 100,000 epochs, it shows less sudden changes compared to two other networks and the changes are smoother. On the other hand, in these epochs, the gap between test and train error is more than other networks.

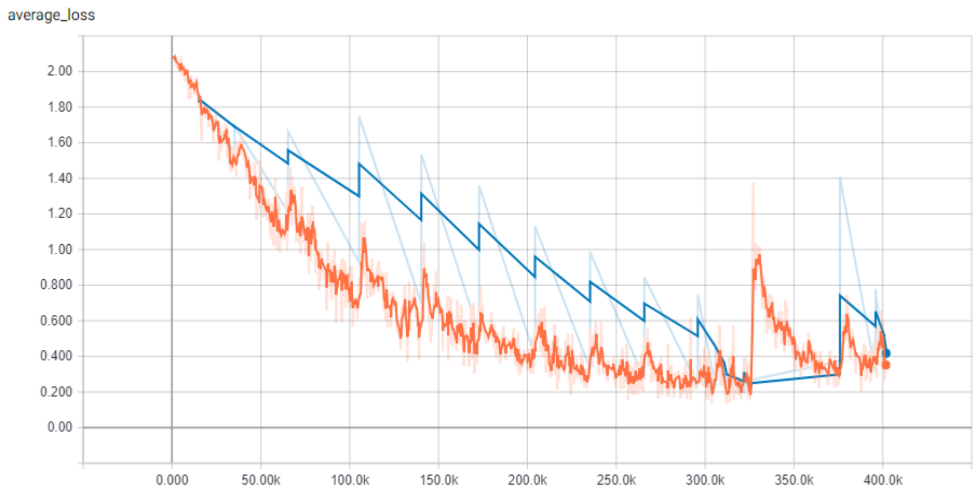


Figure 21. Average Loss of 9 Hidden Layers Network

In summary, although reducing the number of hidden neurons decreases the computational cost, it has a negative effect on the accuracy. In this case, the shallower networks went toward overfitting faster than deeper ones. The effect of adding more hidden layers on the final network performance was examined.

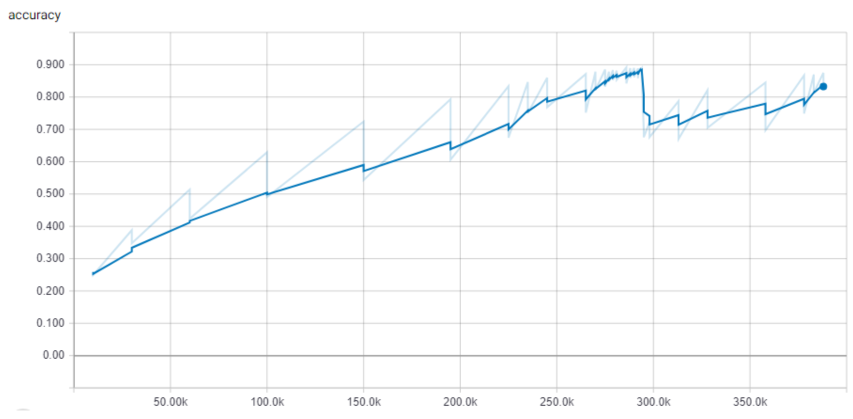


Figure 22. The 7 Hidden Layers Network Accuracy

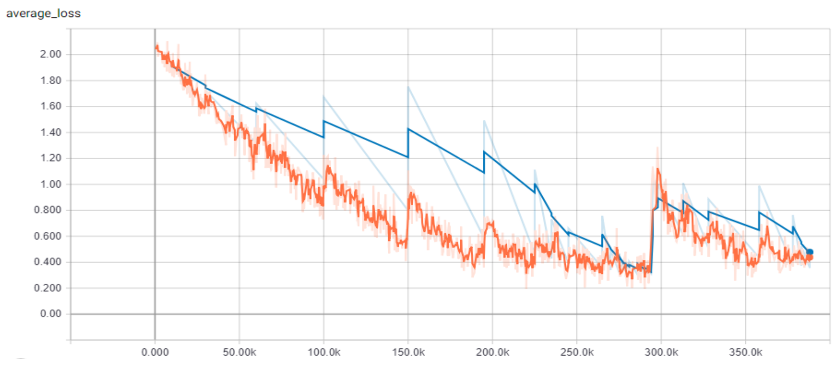


Figure 23. Average Loss of 7 Hidden Layers Network

Network with 11 Hidden Layers

In this section, the number of hidden layers of the baseline network (network with 10 hidden layers) was increased to test if it can improve the accuracy of the network. A network with 11 hidden layers, 1000 hidden neurons and learning rate 0.0001 was built. Figure 24 shows the accuracy of this network. The plot of accuracy is similar to the 10 hidden layers network, but the network reached 90% accuracy faster, around 50,000 epochs earlier. In this case, adding one hidden layer increased the learning speed. On the other hand, the final test and train accuracy of this network decreased by 3%. Since two networks show the same learning pattern and the 11 hidden layers network only trained faster, this difference can be due to different initial weights. Besides, the networks might find a local minimum which resulted in a less accurate network.

Figure 25 shows the average loss of 11 hidden layers network. The loss plot of this network is similar to 10 hidden layers network, but it has a bigger gap between train and test error in almost all epochs especially in the last 50,000 epochs. The loss plot of this network is like the loss plot of 9 hidden layers networks and shows the same gap in almost all epochs. In conclusion, removing one hidden layer didn’t have much effect on the training speed, but a network with one more hidden layer needed fewer epochs to reach the same result.

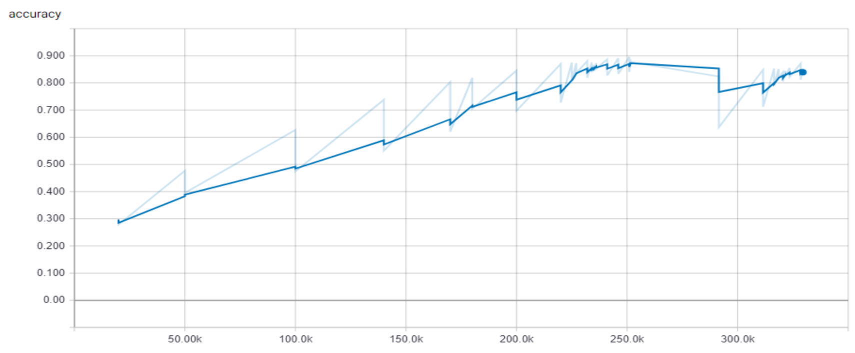


Figure 24. Accuracy of 11 Hidden Layers Network

Network with 13 Hidden Layers

Two more hidden layers were added to this network and a network with 13 hidden layers, 1000 neurons in each hidden layer and 0.0001 learning rate was built. Figure 26 shows the 13 hidden layers network accuracy plot. This network shows the best results between all networks that were trained. The accuracy increased gradually for 500,000 epochs with no sudden changes that was observed in the previous networks, so the network continued learning and reached 94% training and 93% testing accuracy which is the best-achieved result besides, it has the minimum gap between train and test error. Figure 27 shows the average loss of this network. Both train and test curves show a smooth decrease of the network error and in the last 50,000 epochs, the gap between these two curves is close to zero. In conclusion, this network has the best accuracy, the least gap between test and train accuracy and no sudden change in the accuracy of the network.

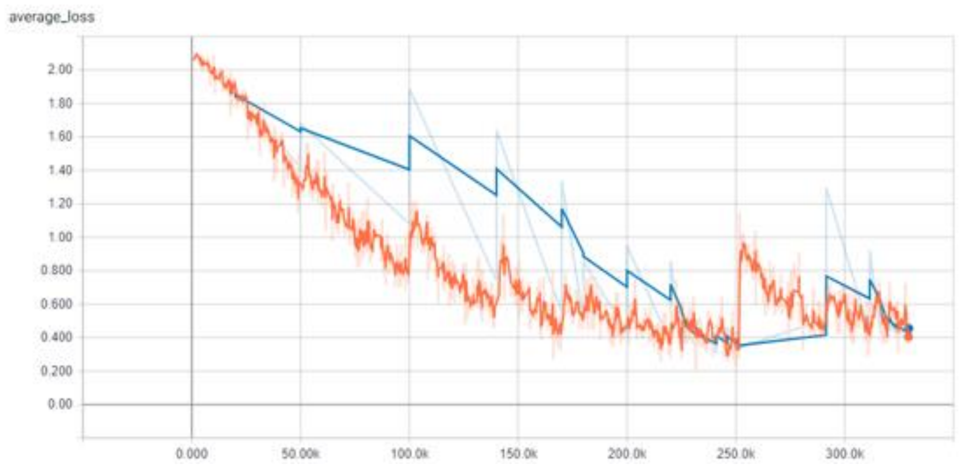


Figure 25. Average Loss of 11 Hidden Layers Network

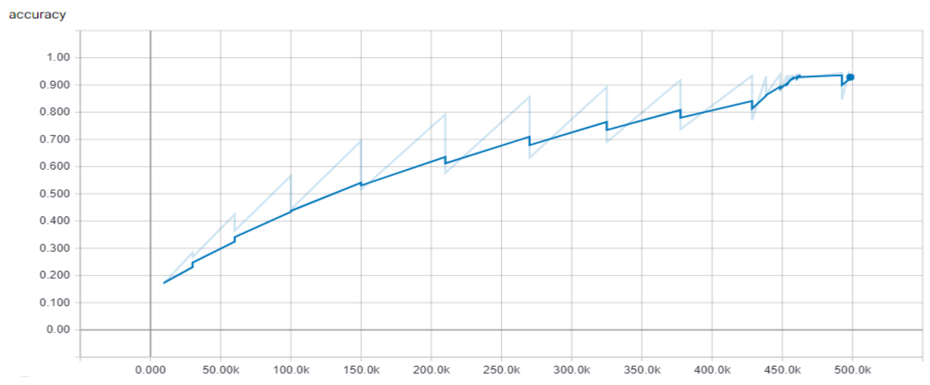


Figure 26. Accuracy of 13 Hidden Layers Network

Network with 14 Hidden Layers

If a network with 10 hidden layers is considered as the baseline removing one hidden layer didn’t make a significant change in the result but removing 3 hidden layers decreased the test accuracy by 4% and increased the gap between train and test accuracy. Adding one hidden layer decreased the accuracy but resulted in the same gap and adding three hidden layers resulted in the best result. The network’s performance was tried to enhance by adding one more hidden layer and built a network with 14 hidden layers, 1000 hidden neurons and learning rate 0.0001.

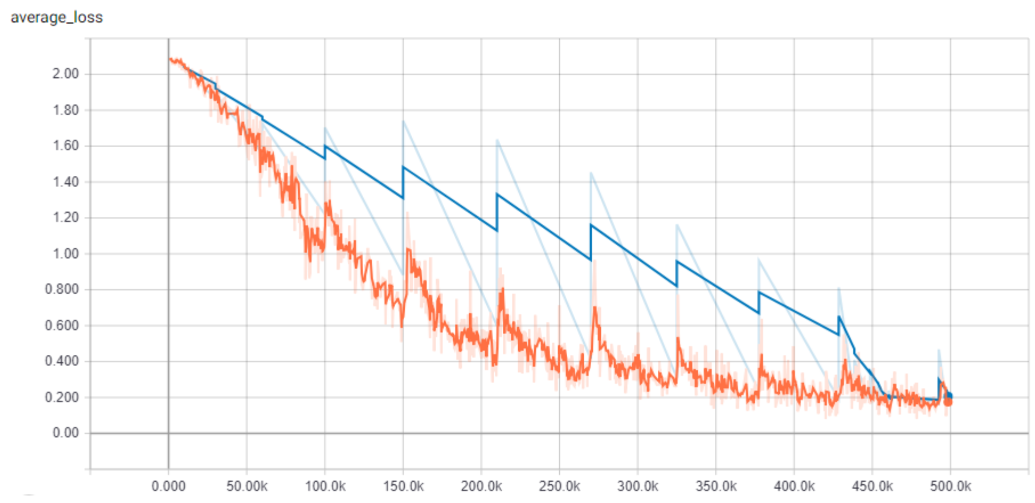


Figure 27. Average Loss of 13 Hidden Layers Network

Figure 28 and figure 29 show the accuracy and the average loss of 14 hidden layers network. The final accuracy of the network decreased by 12% compared to the 13 hidden layers network. This network reaches around 80% after 360,000 epochs but after that, the accuracy decreased gradually. The network stuck in local minima and it couldn't continue learning. If the training was stopped around 360,000th epoch, the epoch that the accuracy started decreasing after it, around 90% accuracy could be achieved that is still less than the previous network, but it is very close to the results of 9,10 and 11 hidden layers networks.

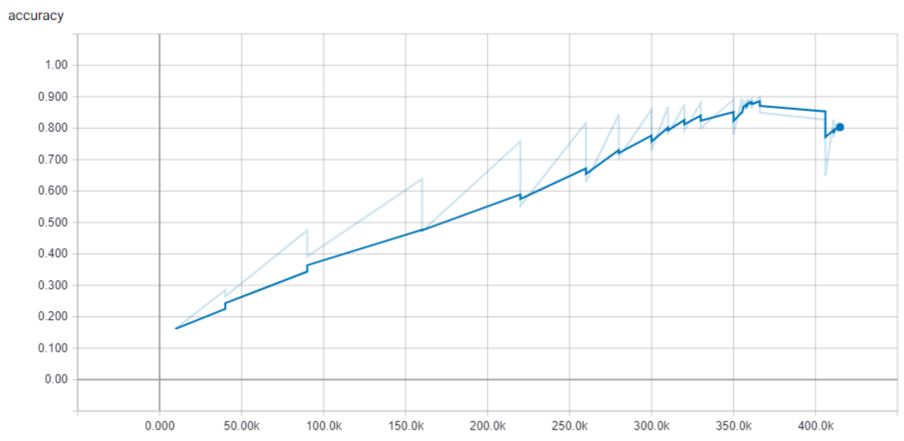


Figure 28. Accuracy of 14 Hidden Layers Network

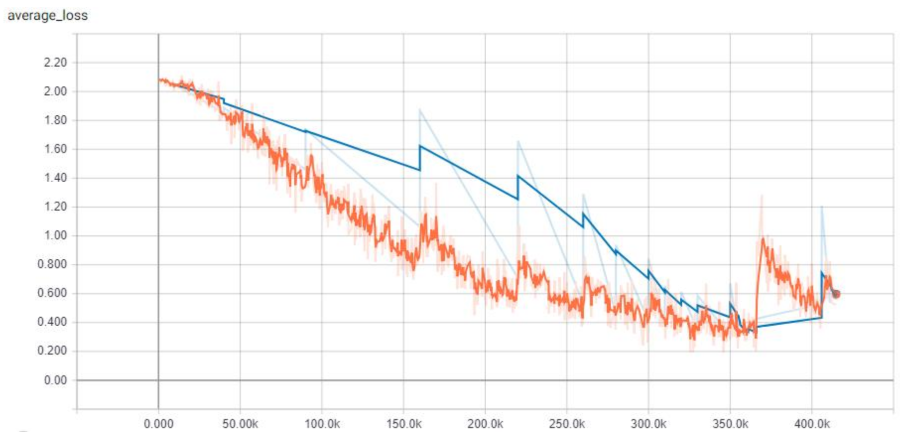


Figure 29. Average Loss of 14 Hidden Layers Network

Low and High Number of Neurons

All these networks were trained with 1000 hidden neurons in each layer since a variety of the number of hidden neurons was tried and between 500 and 1000 neurons in each hidden layer were resulted in the best performance. The network with 10 hidden layers was the second-best network based on the test and train accuracy, so this network was chosen and was trained with 500 neurons in each hidden layer to see the effect of reducing the number of connections in a network with good performance. Moreover, this network was trained with 2000 hidden neurons in each hidden layer to see if more hidden neurons can improve the accuracy of this network. Figure 30 shows the accuracy of these two networks.

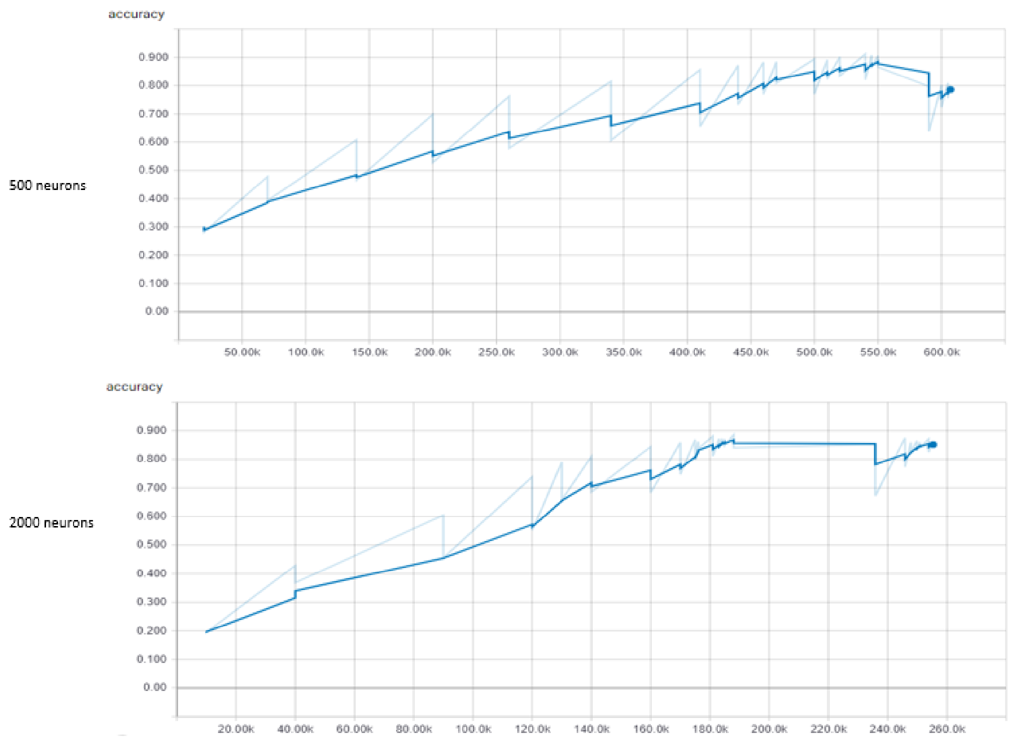


Figure 30. The 10 Hidden Layers with 500 Hidden Neurons and 10 Hidden Layers with 200 Hidden Neurons

The network with 500 hidden neurons learns much slower than other networks. It reached around 89% accuracy after 550,000 epochs and after that, the accuracy decreased for 50,000 epochs. The same network with 1000 hidden neurons reached 89% accuracy after 300,000 epochs which are 250,000 epochs sooner than this one and the accuracy didn’t change much after 300,000 epochs in that network, so decreasing the number of hidden neurons to half reduces both the training speed and the accuracy. Increasing the number of hidden neurons two times, increased the network training speed around 1.5 times. This network reached 88% accuracy after 190,000 epochs, but it stopped learning and the accuracy didn't improve after that. The final accuracy of this network is 2% less than the network with 1000 neurons. Figure 31 shows the average loss of these two networks. Both networks have the same pattern but the network with a low number of neurons needs around 3 times more epochs to reach the same accuracy. In summary 1000 hidden neurons was the best option for the system. Increasing or decreasing the number of hidden neurons reduced the accuracy of the network.

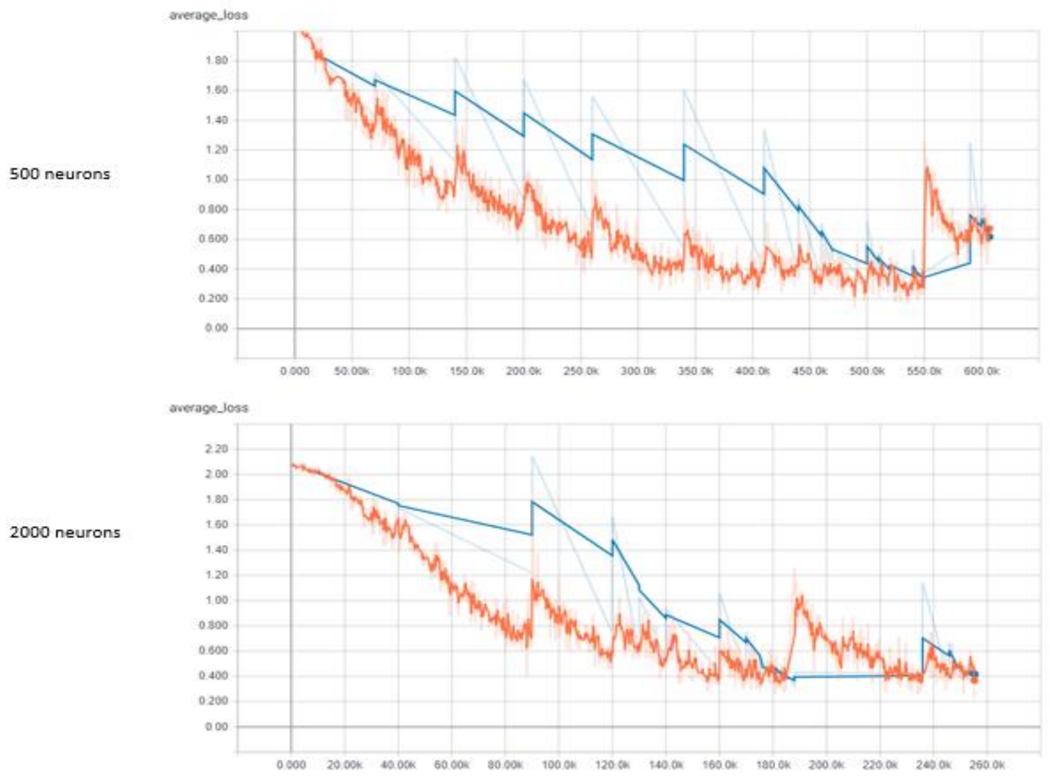


Figure 31. Average Loss for 10 Hidden Layers Networks with 500 and 2000 Hidden Neurons

Conclusion

The number of driver's errors was analyzed in different driving modes and the effect of driving context on driver behavior was discussed. After that, the context of driving was detected using a deep neural network with 13 hidden layers, 1000 hidden neurons in each hidden layer accurately. If the driver is distracted or not and what is the context of driving were detected. The context of driving has a significant effect on driver performance, so detecting the context of driving provides valuable information about driver behavior. The probability that a driver makes mistake in an adverse mode such as night driving is more than the ideal mode that is day driving, so the combination of the context of driving and driver situation can detect the severity of distraction.

DRIVING BEHAVIOR ESTIMATION USING LSTM NETWORK

Traditional Artificial Intelligence approaches are unable to efficiently model context-dependent applications such as language translation and time series. Although neural networks are very powerful models, they have some limitations. Recurrent neural networks (RNN) are the extended version of neural networks without these limitations. They are connectionist models that pass information across a sequence of steps and process the sequential data one at a time. They can model-dependent input and output sequences of elements [96].

LSTM networks, that are a type of RNNs, were chosen as an appropriate model for detecting the driver distraction level in the Intelligent Co-driver system. Two types of LSTM networks were built to predict the level of driver distraction considering the driving data. The first network is a multi-input single output network that considers a block of car-related data vectors (Vc) to estimate the corresponding driver-related data vector (Vh). It is similar to the neural network that was discussed in Chapter 2, but this network considers the effect of all Vc vectors that were collected during a task to compute the correlated Vh vector. This network was called Micro LSTM.

The second one is a multi-input multi-output LSTM network which uses a sequence of Vc vectors to estimate a sequence of Vh vectors. The output of this network is the driver behavior in a specific time range, and it is estimated by considering a sequence of driving data correlated to that specific time range. It was called Macro LSTM, and it is discussed with more details in chapter 6.

For these two LSTM networks, an input dataset was prepared including driving data vectors (Vc) with 10 features, which were chosen from the master dataset using t-test, including velocity, steering, accelerating, braking, headway time, headway distance, longitude accelerating, lateral acceleration and speed. The output vectors of these networks (Vh) have four features including the response time, number of errors, driving mode and number of the task's navigation steps that collected for each task separately.

Hidden Markov Model

Recurrent Neural Networks are not the only artificial intelligence approach with memory. Markov Chain is the simplest form of a Markov Model that models transition between states in an observed sequence. Hidden Markov Model (HMM) models the sequence of observed states as prebiotically dependent on an unobserved sequence of states. Markov Chain and hidden Markov Model are extensions of finite state automata. Markov Chain is a special case of the finite automaton in which the weights are the probabilities and the input sequence determines the states that the automaton will go through. Markov Chain can't represent inherently ambiguous problems [97].

Each Markov model includes three components: *states, transition probabilities matrix* and *specific start and final state*. The first essential assumption of a Markov chain is that each state only depends on the previous state (21), and based on the role of probabilities, the sum of the values of outgoing arcs from each state must be one.

$$p(q_i \mid q_1 \, q_2 \ldots q_{i-1}) = p(q_i \mid q_{i-1}) \; (21)$$

Markov chain can be used when the sequence of events is observable, but in many applications, some events are not directly observable, so Hidden Markov Model must be used. A Hidden Markov Model is a statistical Markov model that can model the system that has unobserved states. This model can be presented as the simplest dynamic Bayesian network. Each Hidden Markov Model has seven components including *a set of states*, *transition probability matrix*, *a sequence of observation*, *a sequence of observation likelihood*, *special start and final state*, *initial probability distribution* and *a set of acceptance states* which is the subset of states.

A first-order Hidden Markov Model has two assumptions. First, the probability of a particular state only depends on the previous state. Second, the probability of a particular output observation depends only on the state that produced that observation and it doesn't depend on any other states or observations (22) [97]:

$$p(o_i \mid q_1 \dots q_i, \dots q_T, o_1 \dots o_i, \dots o_T) = p(o_i \mid q_i) \quad (22)$$

Markov Model has some limitation since the states must be chosen from a medium-sized discrete state space S. The order of the dynamic algorithm which is used to perform HMM is O (|S^2|) and the transition table, which shows the probability of moving between two states, is of size| S^2|. If the size of hidden states grows large, the standard operation becomes computationally expensive and infeasible. Moreover, in Markov model, each state depends on the previous state. It is feasible to extend the context window, but this procedure increases the state space exponentially with the size of the context window. Therefore, rendering a Markov model for modeling long-range dependency is computationally very expensive and impractical [98].

When the model is known, it means all possible states and transitions between them are known, using the Hidden Markov model is an appropriate choice and the training process is easier and faster compared to a recurrent neural network. When the model is unknown, like most real-world data, a recurrent neural network should be used since it can find long-term dependency between data, besides it doesn’t need any pre-existing knowledge about the network. We are working with real data related to driving in different driving contexts. Besides, a model with memory needed to be made to observe the driver's behavior continuously for several seconds and make a decision about the level of distraction based on the length of abnormal behavior. Since there isn’t any pre-knowledge about the model’s features, and the possible states and actions are very large, a recurrent neural network was chosen to make this model.

Long Short Term Memory (LSTM)

As explained earlier, RNN is a type of neural network that can send feedback signals and it memorizes several input samples before the current state. RNNs model dynamic systems that the states of the system are not independent. The architecture of RNNs has cycles that incorporating the outputs of previous time steps as the input to the network in order to make a decision for the current input. This feature makes the RNN an appropriate model for sequential labeling [99].

Although RNN is a simple and powerful model, practically, it is hard to train. Gradient vanishing and gradient exploding are two major RNN’s training challenges. In simple RNNs the output of each time step computes using (24). In this equation *W* shows the weights matrix, u is the current input, b is the bias and $\sigma(x_{t-1})$ is the output of the previous step. Backpropagation through time (BPTT) is an approach for computing the gradient in recurrent neural networks. BPTT considers recurrent neural networks as a multi-layer network and it applies backpropagation on the unrolled mode.

$$x_t = f(x_{t-1}, u_t, \theta) \ (23)$$

$$x_t = w_{rec}\sigma(x_{t-1}) + w_{in}\, u_t + b \ (24)$$

simple RNN doesn't have long term memory and can't save the long dependency of input data. Long short term memory networks which are an extension of simple recurrent neural networks have long term memory and they can work with the varying length of inputs and outputs. The basic LSTM unit is called block and it has more complex architecture compared to RNN memory cells. Each cell has three types of gates including the input gate, the output gate and the forget gate [93] [94]. Equations 25-29 show how the output of each gate and the final output of each memory cell can be calculated. The first step is deciding about the information that the cell is going to forget using the forget gate (f_t) (25). The next step is deciding about the amount of information that the cell is going to keep using the input gate (i_t) (26), and candidate gate (c_t) (27). The final state is deciding about the cells' output using output cell using (28) and (29). In these equations b, w and h_t show bias, weight and hidden state in time step t [95].

$$f_t = \sigma(w_f * (h_{t-1}, x_t) + b_f) \ (25)$$

$$i_t = \sigma(w_i * (h_{t-1}, x_t) + b_i) \ (26)$$

$$c_t = tanh(w_c * (h_{t-1}, x_t) + b_c) \ (27)$$

$$o_t = \sigma(w_o * (h_{t-1}, x_t) + b_o) \text{ (28)}$$

$$h_t = o_t \; * tanh(c_t) (29)$$

Data Features

In Experiment 3, driver related data (Vh) was collected for 35 volunteers. Driver data including the number of navigation steps of the task, response time, the number of errors that the driver had during the task and the mode of driving have been collected manually. In non-distracted modes, the driver didn’t have any interaction with the car interface, so only the driver errors during the trip and the mode of driving were collected as the driver related data and zero was put for the response time and the number of steps.

Most drivers were expected perform tasks on the designed interface with no error since the designed interface is simple to navigate and easy to learn besides drivers became familiar with the interface before starting the experiment. Table 14 shows the number of errors, average and distribution of error in each distracted mode. The average error per task is 0.5 for three distracted modes and the average error in fog mode is 0.1 less than other modes which are not a notable difference. In all modes, 50% of tasks have been done without any error and in 75% of tasks, the driver had one or no error. It means the design of the interface is appropriate for all these contexts of driving.

Minimalist design was used as the user interface design of the experiments, so the average response time for each interaction’s step is expected to be less or equal to 2 seconds which is the safe range for eyes off the road time. It means for 2, 3 and 4 steps task the average response time should be 4, 6 and 8 seconds respectively. 2-step, 3-step and 4-step tasks were used, so the average response time for all tasks should be around six seconds.

Table 14. Errors in Distracted Modes

Mode	Tasks	Errors	Mean	Deviation	min	50%	75%	max
Day-distract	506	260	0.51	0.87	0	0	1	5
Night-distract	519	266	0.51	0.9	0	0	1	6
Fog-distract	534	222	0.41	0.8	0	0	1	5
Nightfog-distract	537	272	0.5	0.94	0	0	1	5

Table 15 shows the features of response time in distracted modes. The average response time in all modes is around 6 seconds which is in the safe range. Based on the collected data, 28% of the tasks had two steps, 40% had 3-steps and 32% had 4-steps. If each step takes 2 seconds the average response time for these three types of tasks should be 6.08 seconds. In all four distracted modes, the average response time is very close to this number and the mean of all modes is 6.27, so considering the delay that the drivers put between every two steps of interaction, the average response time for all types of tasks in all driving context is in an acceptable range. It can’t be said that all steps took less than two seconds, but the response time is not far from the ideal interaction time.

Tasks Features

Three types of tasks were defined in the experiment including 2-step, 3-step and 4-step tasks. Table 16 shows the features of each type of task. If one second is considered for each delay between interaction steps, the average response time for all three types of tasks is in the safe range. The average response time for the shortest task is five seconds which shows on average each step took around two seconds considering a one-second delay between two steps.

Table 15. The Response Times

Mode	Tasks	Mean	Deviation	25%	50%	75%	max
Day-distract	506	6.65	4.01	3.95	5.38	8.65	31
Night-distract	519	6.17	3.5	3.7	5.1	7.8	30.78
Fog-distract	534	6.21	3.7	3.7	5.19	8	27.93
Night fog-distract	537	6.07	3.5	3.8	5.27	7.57	39.73

Table 16. Response Time Based on the Type of Task

Task type	Number tasks	Mean response time	Deviation	25%	50%	75%	max
2-step	597	5.06	3.52	2.93	4	6	28
3-step	826	5.85	3.05	3.8	5	7	27.93
4-step	673	7.86	4.04	5.2	7	9	38

The average response time for 3-step and 4-step tasks shows on average each step took less than two seconds in these tasks. The response time of 55% of 2-step tasks is less or equal to 4 seconds and 64% of 3-step tasks response time is less than 6 seconds besides 65% of 4-step tasks' response time is less than 8 seconds. It means, even if we don't consider the delay between navigation steps the response time of more than half of the tasks is in the range of safe eyes off the road time. Table 17 shows the number of errors in different types of tasks. Based on this table, the length of the task has an adverse effect on the driver's performance. Longer tasks resulted in more error consequently more distraction.

Data Mapping

For the driver data vector of each task (Vh) several car related data vectors (Vc) have been collected by the simulator. The number of Vc vectors for each task depends on the response time of the task. To map Vc vectors to the corresponding Vh, the Vc vectors of each trip were divided based on the length of tasks' response time of that trip to Vc blocks with different lengths and mapped each block to one Vh vector. Equation (30) was used to find the percentage of Vc vectors in a trip that related to each task. In (30), N is the number of tasks that were done on a trip and $percentage(i)$ is the percentage of Vc vectors in the trip that are related to $task\ i$.

$$percentage(i) = (response(i) / \sum_{k=0}^{n} \text{respons(k)}) * 100 \quad (30)$$

Table 17. Number of Errors

Task type	Number Tasks	Mean Error	Deviation	25%	50%	75%	max
2-step	597	0.4	0.8	0	0	1	5
3-step	826	0.45	0.83	0	0	1	4
4-step	673	0.59	0.99	0	0	1	6

For example, if four tasks were done in one trip and the response times of tasks are 2, 3, 4 and 1 seconds, we map 20% of Vc vectors to the first task, 30% to the second task, 40% to the third task and 10% to the last task. In distracted modes, the drivers were asked to do distractive tasks continuously and an equal time interval was put between every two tasks. It means that when the driver finished a task, we waited for a few seconds before asking him/her to do the next task, hence eliminating any cross-task dependencies.

After preprocessing and mapping input and output vectors, there were varying lengths of input sections, so zero-padding was used to have the same input sequence length for all Vc blocks. The padding can be applied to the end of the sequence or to the beginning of it. Sequence truncation is another preprocessing method for varying length sequences. The length of sequences can be trimmed to a predefined length. Two ways can be used for sequence truncation: removing timesteps from the beginning or removing timesteps from the end of sequences. Zero-padding was used and add half of the zero cells were added to the beginning of the block and half of them to the end of the block. These data were used for training the LSTM models. The first model is called Micro LSTM which is a multi-input single-output network and estimates the driver data using the related sequence of driving data.

Micro LSTM

An LSTM network with one hidden layer was built using TensorFlow version 1.2.1 and tested different numbers of hidden neurons, learning rates and batch sizes for this network. Learning rate larger than 0.001 resulted in a very low accuracy, so 0.001 and 0.0001 were chosen as appropriate learning rates. After that, the model was started with

50 hidden neurons, and the number of hidden neurons was increased gradually to find the best choice. The first network was built with 50 hidden neurons, 0.001 as the learning rate and 100 as the batch size for the training phase. Figure 32 shows the result of training the network with 50 hidden neurons. The training was started with 75 epochs and gradually the number of epochs was increased to figure out in which point the network stops learning. Table 18 shows the results of training the model with different numbers of epochs in the range of 75 to 1000.

Based on the achieved results, the gap between the test and train error increased gradually from 0.04 for 75 epochs to 0.37 for 1,000 epochs training. The average absolute error was calculated by dividing the mean absolute error to the mean real output. The training error decreased gradually from 30% to 20% as the number of epochs was increased from 75 to 400, but after that increasing the number of epochs only increased the gap between test and train error. In conclusion, after around 400 epochs the network stopped learning and increasing the number of epochs resulted in overfitting.

Table 18. The Results of a LSTM Network with 50 Hidden Neurons

epochs	MAE train	MSE train	MAE test	MSE test	Train error	Test error
75	1.05	2.9	1.09	3.23	30.14	30.92
200	0.82	1.57	0.99	3	23.62	27.88
300	0.68	0.84	1	2.78	19.54	28.54
450	0.75	1.07	0.99	2.8	21.49	27.82
700	0.71	0.95	1.06	3.9	20.39	30.66
1000	0.69	0.91	1.06	4.18	19.82	29.56

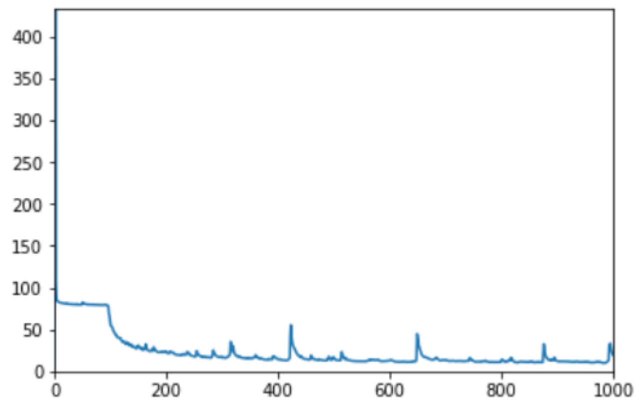


Figure 32. Absolute Error Plot of an LSTM Network with 50 Hidden Neurons

The learning rate of this network was decreased and 0.0001 was used instead of 0.001 then the network was trained with 50 epochs and the training epochs were increased gradually. Figure 33 shows the error plot after 100 epochs.

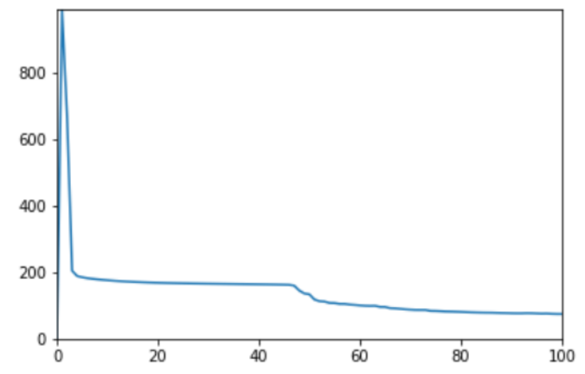


Figure 33. LSTM Network with 50 Neurons and 0.0001 Learning Rate

The mean absolute error is 0.94 for training and 1.07 for the testing phase besides the error is 26.82% for training and 30.64% for the testing phase. The gap between training and testing results increased even faster than the network with a higher learning rate. The result of the first 100 epochs of network with 0.001 learning rate in and the result of this network with learning rate 0.0001 are very similar, so changing the learning rate has no effect on the accuracy of this network.

Batch Size

When the batch training is used, the number of times that the weights of the network change during training is less than stochastic gradient descent training since for each batch of data the weights update once [99]. An LSTM network was built with one hidden layer, 128 hidden neurons and 0.001 as learning rate. Three batch sizes were tried including 50, 100 and 200 then the accuracy of them were compared. Table 19 shows the results.

Table 19. Batch Size Effect on Training Accuracy

Batch size	MSE train	MSE test	Train error	Test error	epochs
50	0.5	2.49	10.93	18.11	500
100	0.63	1.77	9.91	18.63	500
200	0.7	3.03	18.13	30.86	400

The results of batch size 50 and 100 are similar. Figure 34 (a) shows the plot of absolute error for training with 50 as the batch size and Figure 34 (b) is the plot of training error with the batch size 100. These two plots show the same process with a different start. Training the same network with two different batch sizes (50 or 100) didn’t show a significant difference in the final results but choosing 200 as the batch size increased the training and testing error. Different number of epochs were tried to find the point that overfitting started. Table 20 shows the results. After 200 epochs the network stopped learning and only the overfitting problem increased.

Table 20. Training Results of Network with Batch Size 200

epochs	MAE train	MSE train	MAE test	MSE test	Train error	Test error
150	0.87	1.69	1.05	3.66	24.65%	31%
200	0.63	0.74	1.03	3.16	18.39%	28.42%
400	0.63	0.7	1.06	3.03	18.13	30.86

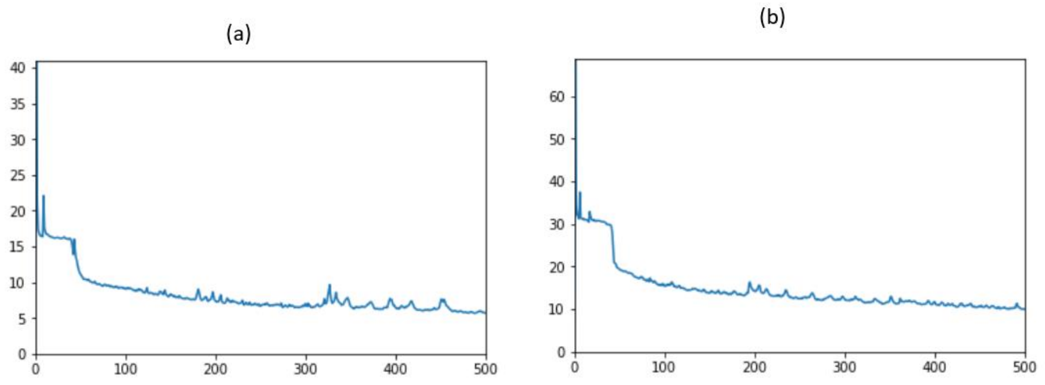


Figure 34. (a) Batch Size 100, (b) Batch Size 50

The batch size was set 200, and the training process started with 50 epochs. Then, the number of training epochs increased gradually. After 200 epochs the accuracy of the training phase didn't show significant improvement (Figure 35). The gap between the test and train error after 150 epochs training was similar to the results of two other batch sizes after 500 epochs, but the error is 14% higher. More epochs were tried to enhance the model's performance. Although the training error decreased around 6%, the testing error didn’t change. Batch size 100 was chosen for networks with one hidden layer network and batch size 50 was chosen for networks with multiple hidden layers.

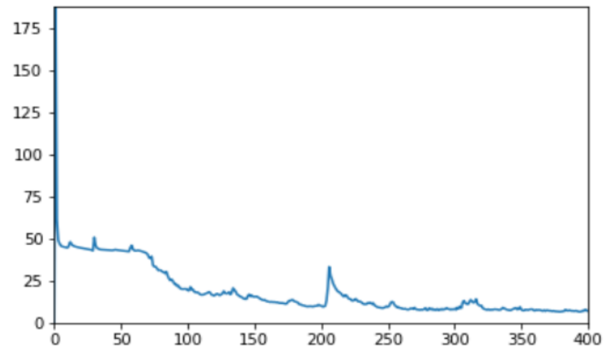
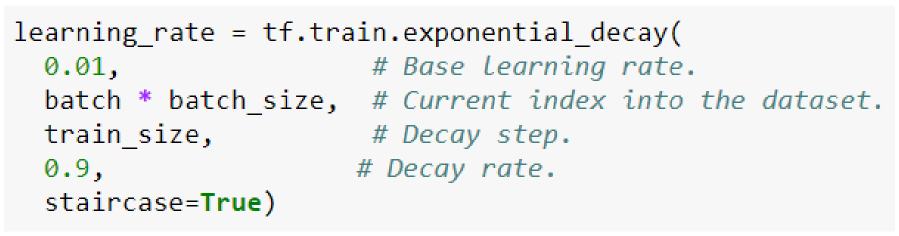


Figure 35. Training Error for Batch Size 200

Learning Rate

Adam optimizer which computes a separate learning rate for each parameter was used in this model. Adam is derived from adaptive moment estimation. In this optimizer, an initial value for the learning rate needs to be set, and the optimizer changes the learning rate for each parameter during the training process. An adaptive learning rate was used to initialize this optimizer instead of fix value and the results of these two networks were compared. Figure 36 is the adaptive learning rate that we used [78].



```
learning_rate = tf.train.exponential_decay(
  0.01,                # Base learning rate.
  batch * batch_size,  # Current index into the dataset.
  train_size,          # Decay step.
  0.9,                # Decay rate.
  staircase=True)
```

Figure 36. Adaptive Learning Rate

An LSTM network was built with one hidden layer and 400 hidden neurons then the network was trained with both fixed and adaptive learning rate. Different combinations of decay rate and initial values for the adaptive learning rate were tried (Table 21).

Table 21. Adaptive Learning Rate with Fixed Learning Rate

Learning rate	Decay rate	Initial value	MSE training	MSE test	Train error	Test error	Epochs
adaptive	0.5	0.01	2.56	4.79	17.79	24.25	400
adaptive	0.9	0.01	1.58	4.14	13.88	21.12	400
adaptive	0.9	0.001	0.33	4.77	7.6	22.44	500
adaptive	0.9	0.001	0.89	4.27	11.84	20.77	300
fixed	-	0.001	0.46	4.77	8.53	22.49	400

In this model, 0.5 was set as the decay rate and the network was trained for 400 epochs then the decay rate was increased to 0.9. Using 0.9 as the decay rate resulted in a better performance. As a result, 0.9 was chosen as an appropriate decay rate for an adaptive learning rate and the initial learning rate value was changed to 0.001. The network's training error decreased by 6.28%, but the testing error increased by 1.32%, so the network went toward overfitting. The number of epochs was increased to 300 and the train and test errors of the model after 300 epochs are close to the network with 0.01 as the initial learning rate.

Figure 37 (a) shows the result of training with an adaptive learning rate when the decay rate was set on 0.9 and the initial learning rate was set on 0.001, and Figure 37 (b) shows the same network with initial learning rate 0.01. Both resulted in almost similar accuracy, but the variance of the error is more when the learning rate was initialized with a larger value.

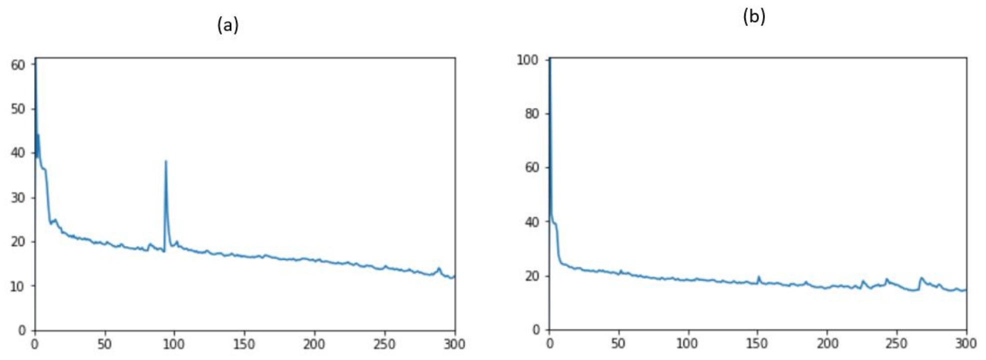


Figure 37. (a) Decay Rate 0.9 and Initial Value 0.001 (b) Decay Rate 0.9 and Initial Value 0.01

In this model, 0.001 was chosen as the fix learning rate of an LSTM model with one hidden layer and 400 hidden neurons. Figure 38 (a) shows the result of this network and Figure 38 (b) shows the result of the same network with an adaptive learning rate. The training error decreased when a fixed learning rate was used but the network overfitted faster.

The spikes in previous figures are because of using mini batches in the training process. These spikes are unavoidable since some batches have unlucky data for optimization that resulted in these spikes in the cost function. If batch gradient descent has been used, they wouldn’t be any spike since it uses all data in each epoch. If stochastic gradient descent has been used, there would be more spikes since the batch size is one and each data changes the network’s parameters.

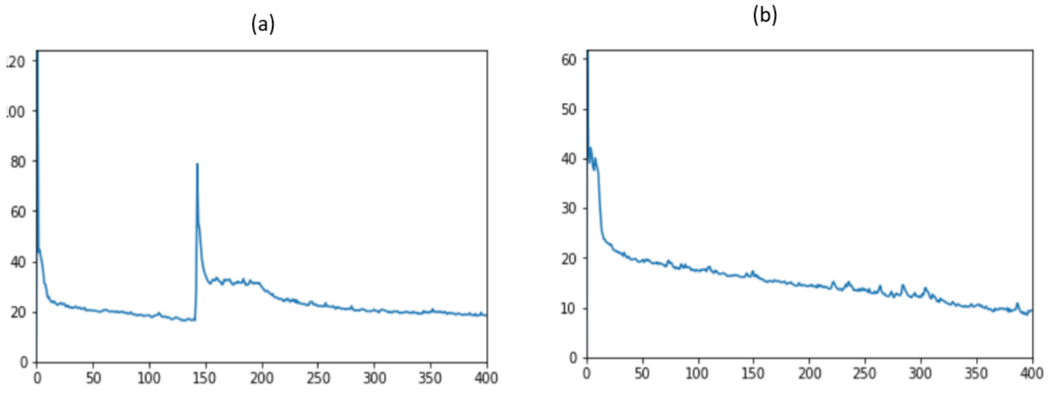


Figure 38. (a) Adaptive Learning Rate Initial 0.01 Decay Rate 0.9 (b) Fixed Learning Rate 0.001

The number of hidden neurons was decreased and a network with 250 hidden neurons was built. After that, it was trained with both fixed and adaptive learning rates. Also, 0.9 was chosen as the decay rate and 0.01 as the initial value of the adaptive learning rate. Table 22 shows the training results of the network with the adaptive learning rate. The training errors decreased gradually in the first 400 epochs, and after that both training and testing errors increased. Figure 39 compares the training error in the first 600 and 1000 epochs. There isn’t much improvement in the system’s accuracy after around 500 epochs.

Table 22. Network with 250 Hidden Neurons and Adaptive Learning Rate

Epochs	MSE train	MSE test	Train error	Test error
300	1.52	4.54	13.53	21.35
400	1.58	5.26	13.34	22.49
600	2.13	6.38	16.88	24.82
1000	1.41	5.63	15.47	26.47

The network with 250 hidden neurons was trained using fixed learning rate 0.0001 in 4 steps including 200, 300,400 and 1000 epochs. Table 23 shows the results of the training and testing phase. The best performance achieved after 200 epochs training. When the number of epochs was increased although the training error decreased, the testing error didn’t change until 600 epochs.

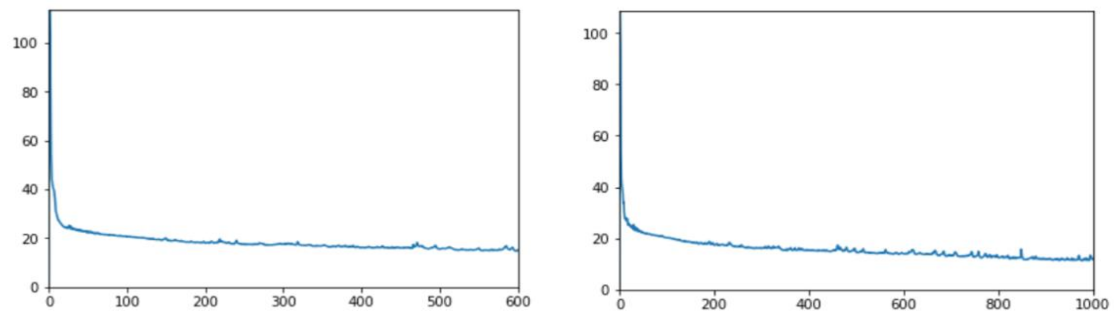


Figure 39. Training Error with the Adaptive Learning Rate

Table 23. Training with the Fixed Learning Rate 0.0001

epochs	MSE train	MSE test	Train error	Test error
200	0.9	4.6	12.7	22.42
300	0.7	3.7	10.95	22.07
400	0.51	6.36	9.2	22.98
1000	0.68	6.03	10.3	25.02

Figure 40 compares the results of 200 and 1000 epochs training the network with fix learning rate. The training rate decreased gradually in the first 200 epochs, but in the next 800 epochs the training didn’t cause significant improvement. The best results for both adaptive and fixed learning rate were after 300 epochs. Figure 41(a) shows the fixed learning rate errors and Figure 41 (b) shows the adaptive learning rate errors for 300 epochs training. The fixed learning rate resulted in a smoother change in the network and better accuracy. In conclusion, training with the fixed learning rate resulted in the lowest error for both testing and training phases, but the network overfitted faster compared to using the adaptive learning rate as the initial value of Adam optimizer.

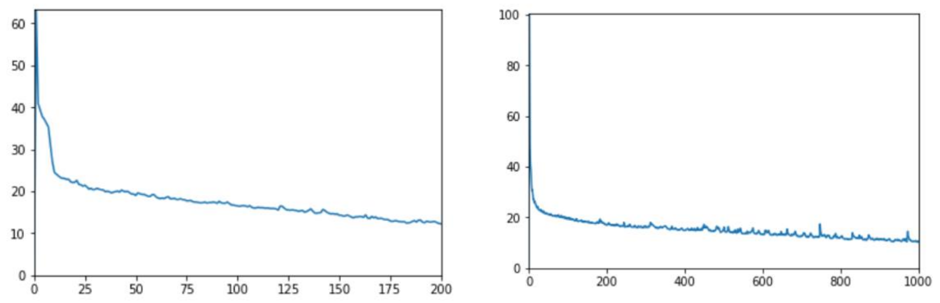


Figure 40. Training Error Using Fixed Learning Rate

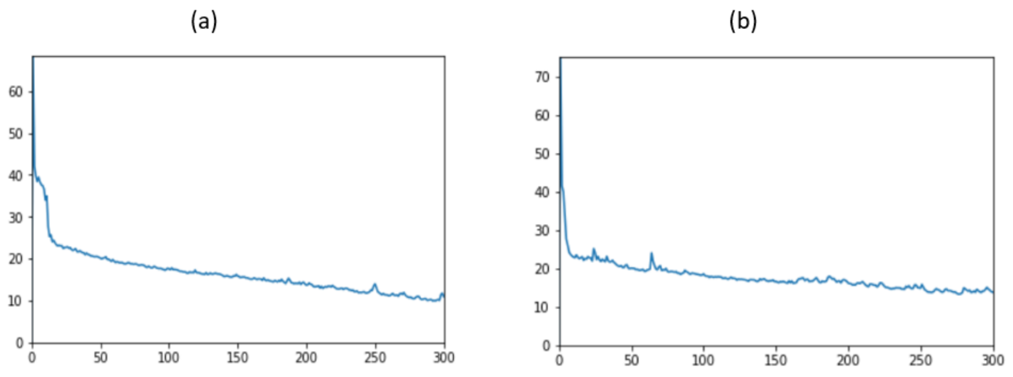


Figure 41. (a) Using Fixed Learning Rate as the Initial Learning Rate in Adam Optimizer (b) Using Adaptive Learning Rate as Learning Rate Initializer

Initial Weights

Initial weights have a notable effect on the training accuracy. If a trained network's parameter is used as the initial parameters of a new network, the new network would be trained more accurately and faster compared to using random numbers as initial weights [92]. A network was built with one hidden layer, 100 hidden neurons, 0.0001 as the learning rate and 100 as the batch size. The model was saved after training and it was restored as the initial network. The best result was a 15% training error and an 18% testing error. We tried to enhance the training process by adding more hidden neurons, so a network was built with 200 hidden neurons and one network was built with 300 hidden neurons. Table 24 shows the results of these networks. Increasing the number of hidden neurons didn’t have a significant effect on the training accuracy, and it increased the test error, so the model went toward overfitting. Figure 42 shows that the network’s error with 300 hidden neurons. After 100 epochs the network stopped learning, so adding more hidden neurons to a network with one hidden layer didn’t help us to enhance the model’s accuracy.

Table 24. The Results of Restore Networks

Hidden neurons	MAE	MSE	MAE test	MSE test	Train error	Test error
100	0.55	0.54	0.66	0.81	15.83%	18.95%
200	0.54	0.55	0.72	1.04	15.65%	20.32%
300	0.53	0.5	0.76	1.17	15.32%	21.9%

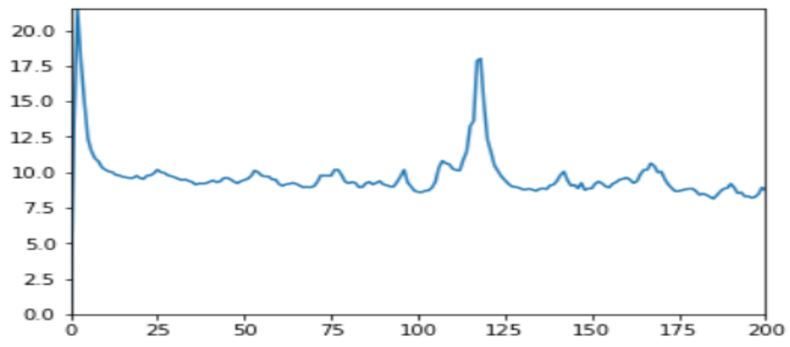
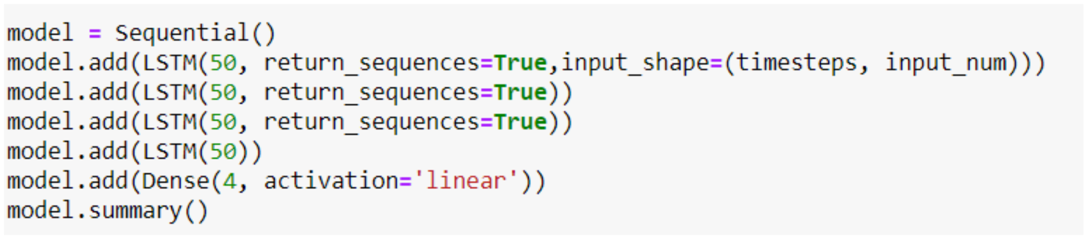


Figure 42. Training Process of a Network with 300 Hidden Neurons

Different combinations of hyperparameters were tried to improve the mean absolute error and mean squared error of the system to minimize the gap between test and test error. The best achieved result of one hidden layer LSTM network was 0.5 train mean absolute error and 0.66 test mean absolute error.

Number of Hidden Layers and Hidden Neurons

The last hyperparameter that was tuned is the number of hidden layers. Three networks were built with two, three and four hidden layers and for each network, 50, 150 and 300 were used as the number of hidden neurons in each layer. These three numbers were chosen as a low, medium and high number of hidden neurons. Besides, 0.0001 was used as the learning rate and 50 was used as the batch size of these networks. Keras was used to build these networks (Figure 43). The training was started with a medium number of hidden neurons, 150 hidden neurons, and it was considered as the baseline for this LSTM network. After that, less and more number of hidden neurons were tried and the results of them were compared.



```
model = Sequential()
model.add(LSTM(50, return_sequences=True,input_shape=(timesteps, input_num)))
model.add(LSTM(50, return_sequences=True))
model.add(LSTM(50, return_sequences=True))
model.add(LSTM(50))
model.add(Dense(4, activation='linear'))
model.summary()
```

Figure 43. Keras Model for LSTM Network with 4 Hidden Layers

Multilayer Network with 150 Hidden Neurons

A model was built with 150 neurons in each hidden layer. Table 25 shows the results of training three networks with two, three and four hidden layers,150 neurons in each layer and unscaled data. The network with three hidden layers after 500 epochs training shows the minimum gap between the train and validation mean absolute error and the best performance. The gap between the train and validation error was 0.3 and the validation error was 0.91 that is very close to other networks' validation errors.

Figure 44 shows the train and validation errors of these networks with 150 hidden neurons and unscaled data. The performance of the networks with two and four hidden layers are very similar. In both of them, the gap between validate and train error is close to zero before 50 epochs and after that, the variance of the gap increased in the most epochs. In conclusion with 150 hidden neurons in each layer, a network with three hidden layers shows the best performance in estimating user behavior using driving data.

Multilayer Network with 50 Hidden Neurons

The number of hidden neurons was decreased to investigate the performance of these three networks with a low number of hidden neurons. Table 26 shows the results of training them with 50 hidden neurons in each layer. The performance of the four hidden layers network improved compared to the 150 hidden neuron ones, and the gap between train and validation error decreased significantly. The gap between train and validation errors for networks with two and three hidden layers increased compared to the networks with 150 hidden neurons. In sum, the network with four hidden layers has the best performance.

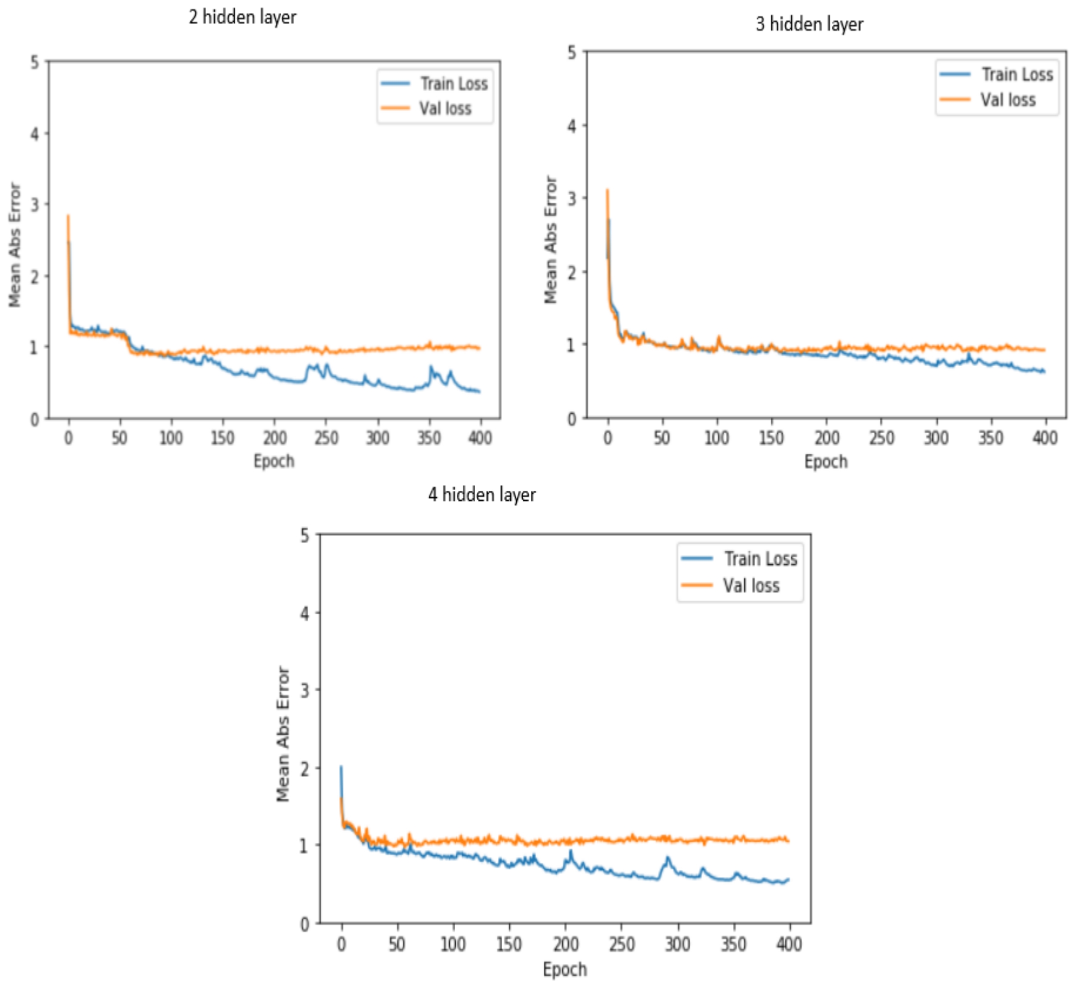


Figure 44. Networks with 150 Neurons in Hidden Layers

Table 25. Networks with 150 Hidden Neurons in Each Layer

Hidden layers	epochs	MSE train	MSE valid	MAE train	MAE valid
2	400	0.39	3.5	0.43	0.98
2	600	0.16	2.6	0.3	0.99
3	400	0.04	2.26	0.14	0.93
3	500	0.7	2.43	0.61	0.91
4	400	0.57	3.4	0.54	1
4	500	0.12	2	0.24	0.89

Table 26. Networks with 50 Hidden Neurons in Each Layer

Hidden layers	epochs	MSE train	MSE valid	MAE train	MAE valid
2	500	0.16	2.6	0.2	1.1
3	500	0.0085	2.3	0.062	0.87
4	500	0.099	2.65	0.61	0.91

Figure 45 shows the train and validation errors of these networks with 50 hidden neurons. For the network with four hidden layers, the training error decreased smoothly, but two other networks have a large number of sudden changes in the train error besides after around 100 epochs a specific pattern can be seen in errors.

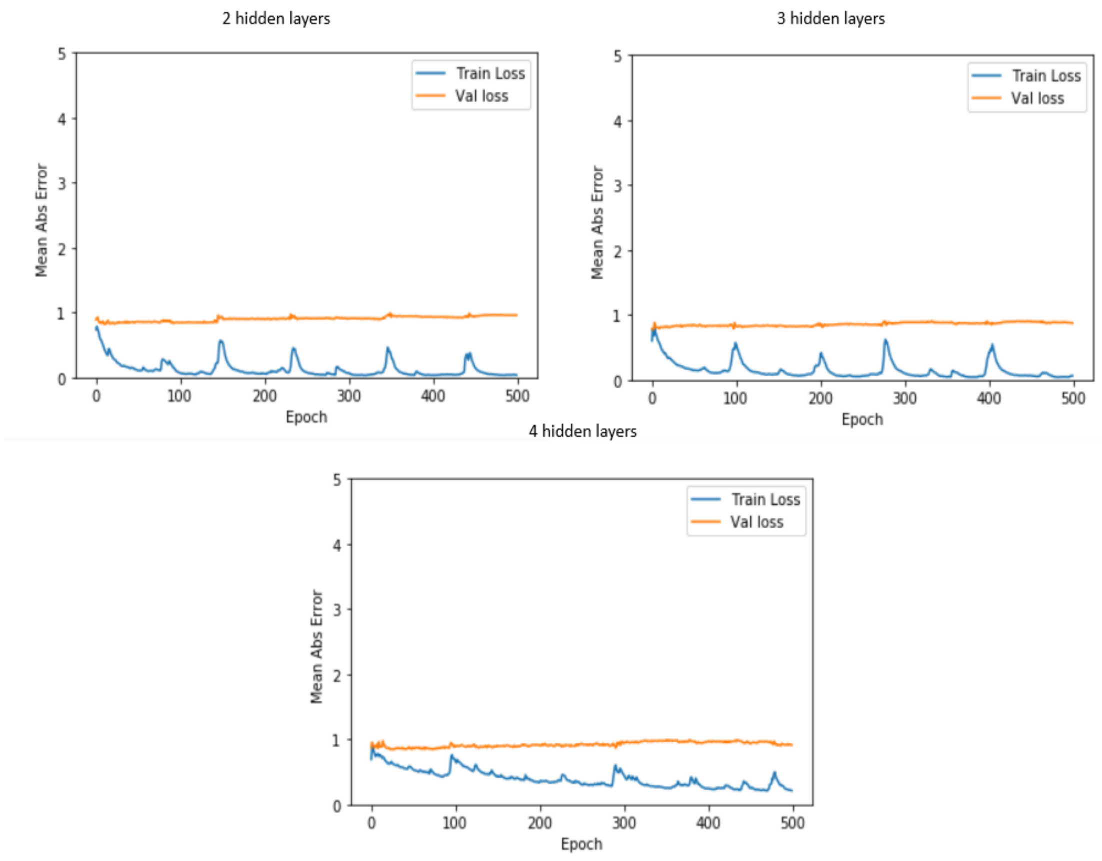


Figure 45. LSTM Network with 50 Neurons in Each Hidden Layer

The error is close to zero for 100 epochs and it increased to around one repeatedly. In the network with two hidden layers the error decreased gradually, and around 150 epochs an increase in the error can be seen. This sudden change repeated every 100 epochs on 250, 350 and 450 epochs. These changes decreased gradually but the overall variation in the training error is higher than the network with four hidden layers.

The error plot of the network with the three hidden layers shows a pattern similar to two hidden layers network but the sudden increase in the error happened faster. Besides the Gap between the validation and train error is smaller. The error of this network decreased in the first 100 epochs and there are some sudden changes with the different amounts on the epoch 200, 300 and 400. In conclusion, the four hidden layers network shows the most stable training progress, but the final errors of two other networks were less than this network. Two other networks have almost similar training processes, but the three hidden layers network shows less gap between the train and validation error compared to two hidden layers network during 500 epochs.

Multilayer Network with 300 Hidden Neurons

The number of hidden neurons was increased to 300. Table 27 shows the train and validation error of them. Increasing the number of hidden neurons didn’t have a positive effect on the accuracy of the networks. The network with four hidden layers shows the best performance considering the gap between the train and validation errors and overall accuracy. Figure 46 shows the training process of these three networks with 300 hidden neurons. The error of the network with two hidden layers decreased gradually and the gap between the validation and train error increased. It means the networks will be overfitted if we increase the number of training epochs.

The error of the network with three hidden layers after 500 epochs is almost constant for both training and validating process except a few sudden changes in epochs 75, 200, 270 and 340. The four hidden layers network shows almost zero difference between the train and validation error in the first 200 epochs and after that, the train error decreased gradually, and the validation error remained constant. If training would be continued, these networks would be overfitted and the final validation error won't decrease anymore.

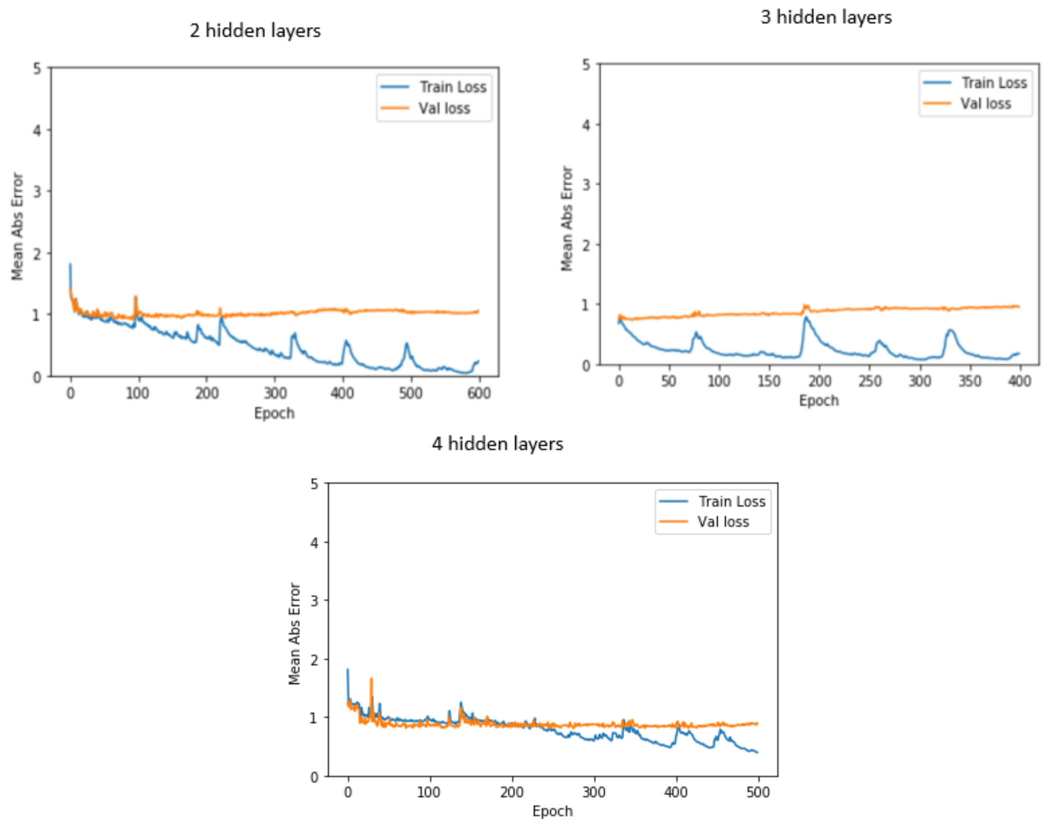


Figure 46. Networks with 300 Hidden Neurons

Table 27. Networks with 300 Hidden Neurons in Each Layer

Hidden layers	epochs	MSE train	MSE valid	MAE train	MAE valid
2	600	0.12	3.4	0.23	1.05
3	400	0.065	2.9	0.17	0.9
4	500	0.314	2.1	0.39	0.89

Results with Scaled Data

All three models including two, three and four LSTM layers networks were trained with the same number of hidden neurons, activation functions and optimization function.

Table 28 shows the summary of the achieved results. The best accuracy of two and four LSTM layers networks achieved with 50 hidden neurons which is similar to the results of these networks with not scaled data. Besides, the minimum error of three layers networks achieved with 150 hidden neurons which is similar to the performance of this network's result with not scaled data. Highlight rows in Table 28 and Table 30 shows the best performance of networks with scaled and unscaled data.

Table 28. Results of Networks with Scaled Data

layer	Hidden neuron	Train MAE	Train MSE	Val MAE	Val MSE	Test MAE	Test MSE	epochs
two-scale	50	0.19	0.09	0.23	0.18	0.23	0.19	1000
two-scale	150	0.21	0.14	0.22	0.16	0.22	0.14	700
two-scale	300	0.16	0.06	0.23	0.18	0.23	0.18	700
Three-scale	50	0.18	0.09	0.2	0.19	0.22	0.17	800
Three - scale	150	0.16	0.06	0.23	0.17	0.23	0.16	700
Three - scale	300	0.15	0.05	0.24	0.2	0.25	0.21	700
four-scale	50	0.23	0.172	0.24	0.174	0.23	0.15	700
four-scale	150	0.21	013	0.22	0.16	0.23	0.17	700
Four-scale	300	0.23	0.16	0.23	0.16	0.25	0.2	800

Dropout

Deep learning neural networks including MLP, CNN and LSTM have tendency to go toward overfitting a training dataset. Ensembles of neural networks with different model configurations is a suggested solution to reduce the overfitting problem which requires the additional computational expense of training multiple deep neural network models. A single model can be used to imitate many network architectures by randomly dropping out nodes during training. This method is called dropout that is a computationally cheap and remarkably effective regularization method to reduce overfitting and improve generalization error in deep neural networks of all kinds [36]. Keras supports dropout regularization and it can be added to the layers of deep neural networks' layers in Keras models.

Table 29. Results of Networks with Dropout

layer	Hidden	Train MAE	Val MAE	Test MAE	epochs	Dropout
Three	50	0.1899	0.2347	0.22	800	0%
Three	50	0.2176	0.2207	0.22	800	20%
Three	50	0.2136	0.2206	0.23	1500	50%
Three	50	0.2152	0.2301	0.23	500	50%
Three	50	0.2161	0.2186	0.22	1000	40%
Three	100	0.2135	0.2194	0.22	1000	50%
Three	500	0.2	0.2153	0.22	500	50%

Different dropout percentages were attempted in the range of 20% to 50% which is the suggested range for dropping the network's nodes. Using a high percentage in dropout layers might result in underfitting and using a low percentage can result in no enhancement in the network's accuracy. These three networks were trained with a variety of LSTM neurons, epochs and dropout percentages. Table 29 shows some of the best-achieved results of the three LSTM layers network, which is the most accurate network based on the previous and current achieved results and scaled data. The results show that using the dropout layer doesn't have a significant positive effect on the network accuracy since the best validation error is similar to the network without the dropout layer. Moreover, using dropouts increased the training time of the deep networks.

Table 30. Micro LSTM Results with Unscaled Data

Layers	neurons	epochs	MAE train	MAE valid	Step time	parameters
2	50	500	0.2	1.1	27.067	32,604
3	50	500	0.62	0.87	37.018	52,804
4	50	500	0.61	0.91	129.08	73,004
2	150	400	0.43	0.98	67.042	277,804
3	150	500	0.61	0.91	174.108	458,404
4	150	400	0.54	1	183.114	639,004
2	300	600	0.23	1.05	272.169	1,095,604
3	300	400	0.17	0.9	264.164	1,816,804
4	300	500	0.39	0.89	862.537	2,538,004

Summary and Conclusion

Predicting driver behavior using machine learning methods is one of the proposed solutions for reducing and preventing driver distraction. Three important features of intelligence and cognition are perception, attention and sensory memory. Intelligent systems have two types of memory including short-term and long-term memory. LSTM networks and stacked LSTM networks were used to add both short-term and long-term memory to the previous model. The network with three hidden layers and 50 neurons shows the minimum train and validation error between all trained networks, but the network is overfitted. The network with three hidden layers and 150 neurons has the best performance and minimum gap between the validation and train error. Memory and attention are two features of intelligent models. In this chapter, a model with short term and long term memory were discussed. In the next chapter, attention was added to the model and the performance of the model with attention and memory was compared to the LSTM network that only has memory and memoryless Neural Network (Chapter2) to show the effect of memory and attention on the intelligent level of the model and the training process.

THE LSTM ATTENTION MODEL

Driving is a continuous task and there is a dependency between driving data samples, so considering this dependency can enhance the performance of driver behavior analysis methods. Including memory in the behavior analysis could provide a higher level of understanding compared to the instantaneous response. The LSTM network with attention layer was chosen as the suitable model [66, 95, 104]. By building a model with memory and attention using an LSTM attention network, both correlation between driving data and driver behavior features, and the dependency between the input sequence of driving data samples during each task can be considered. Adding the "attention" layer to the LSTM network allows the model to consider the weighted impact of each step of the input sequence on the output. By putting more focus on relevant parameters and reduced attention on less relevant ones, it is expected to have a more intelligent network with less demand on computing resources. In this model, the combination of an attention layer and a bidirectional LSTM on the top of the stacked LSTM network improved the model's performance compared to the stacked LSTM network and the multilayer neural network (MLP).

Methodology

Intelligence is influenced by our understanding of a complex situation. The intelligence can be improved by reasoning, understanding, recall, and learning. Cognition includes every mental process that can be described as a new experience such as perceiving, reasoning and recognizing [105, 106]. If an artificial intelligence method and a cognitive system need to analyze a big data set, the artificial intelligence method determines an appropriate action based on the provided information, but the cognitive system provides some information for the user to help her make a better decision [107]. Some of the important features of intelligence and cognition are perception, attention and sensory memory. Intelligent systems have short-term and long-term memory [108].

In chapter 2, the focus is on the perception, and the system didn't have any kind of memory. In Chapter 4, short-term and long-term memory were added to the previous system to increase the intelligence level. In this chapter, the attention mechanism was added to the system with memory to enhance the performance of the system and reduce the computational expenses of the training process [106]. Figure 47 shows the system architecture. The "input Vc" layer shows driving data that were used as the model's input. The stacked LSTM layer adds memory to the model. The bidirectional LSTM layer provides access to both previous and next states and a combination of the bidirectional LSTM layer and attention layer adds attention to the model. Finally, the "Vh output" layer shows driver behavior.

Related Works

[110] Discussed a driver distraction detection method that models head tracking and long-term temporal context of driving data using an LSTM network. An experiment was performed with 30 volunteers, in a straight road in Germany with one lane for each direction and 100km/h speed limit, to collect driver distracted-related data in a realistic driving situation. An Audi, which was equipped with an interface to measure CAN-Bus data and a head tracking system has been used in that experiment. Eight tasks were chosen as distracting conditions including radio, CD, phone book, navigation point of

interest, phone, navigation, TV and Navigation sound. The main distracting functions are available through eight hard keys located on the left and right sides of the interface. In all, volunteers performed 53 non-distracted and 220 distracted runs. The average duration of distracted and non-distracted runs 41.6 s and 59.2 s respectively. They fed collected data from CAN-Bus and head tracking system to an LSTM network and predict the drivers state continuously with 96.6% accuracy in an ideal driving context and one specific driving scenario which is driving in a straight highway, so the performance might decrease in more complex driving situations.

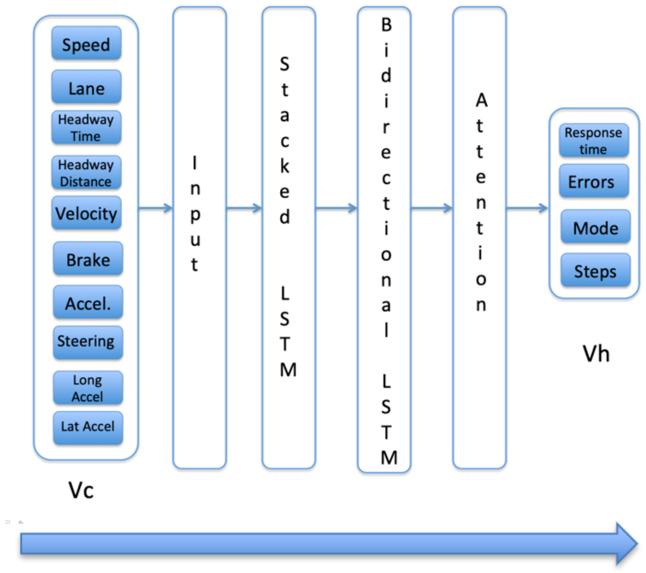


Figure 47. The System Architecture

[111] Proposed an action prediction system that defines driver behavior prediction as a time-series anomaly prediction problem. This system needed to extract features across the collected data from multi-modal sensory input to detect and predict patterns that precede an abnormal driver behavior. The proposed system includes real-time data acquisition, data processing and learning system which predicts future driver action. Five types of

input sensory were used in this system including CAN-Bus, face camera, dash camera, hand camera and GPS + Map. This system is based on Deep Bi-directional Recurrent Neural Network (DBRNN), and it outperformed the unidirectional recurrent neural network across all defined scenarios except one. The performance of both bi-directional and unidirectional models enhances with an individual driver compared to generic driver data. The system can predict driver actions such as braking, lane change turning and accelerating five seconds before the driver performs the action. This model shows the positive effect of considering both previous and next samples in each step, using the bidirectional layer, in the accuracy of driver behavior estimation. We used a combination of attention and bidirectional layer in our model which outperformed the unidirectional LSTM model.

Various psychological and physiological conditions such as fatigue, motion sickness, and distraction can affect driving safety adversely. [112] Discussed a model that aims to detect driver fatigue and drowsiness using a brain-computer interface. The model is a prediction system called Recurrent Self-evolving Fuzzy Neural Network. It uses an online gradient descent learning rule to estimate the correlation between the driver's brain activities and driving fatigue. The brain activities of the driver were monitored by Electroencephalography (EEG). The efficiency of approaches that use the EEG-based brain-computer interface has been limited by the low resolution of the collected data. This system, Recurrent Fuzzy Neural Network, is proposed to enhance the adaptability in realistic EEG applications and overcome the challenge of low-resolution data in EEG-based approaches.

[113] Proposed brain-inspired cognitive model with attention (CMA). The model has three parts including a convolutional neural network that simulates the human visual cortex, a cognitive map that describes the relationships between objects in traffic and a recurrent neural network combined with the cognitive map in order to add the short term, long term memory and attention mechanism to the model. The model performs three tasks accurately. It detects free space in the current lane and adjacent lanes, it estimates the distance to obstacles and vehicle attitudes, and it learns driver decision making process and driver behavior. The accuracy of this model depends on the training dataset and it might decrease in some unknown environments. The model shows the beneficial effect of using attention and memory in learning and detecting driver behavior.

Driving data were used as inputs of a stacked LSTM network with attention layer to predict driver behavior features while interacting with car infotainment system by considering the dependency between a sequence of input samples related to each interaction and weighted effect of each sample on the driver performance. Attention mechanism mostly is used in object detection, image captioning and language translation applications [109]. In this application, the attention mechanism decreased the training time and the model's error. Besides, it reduced the overfitting problem compared to the LSTM model significantly.

Models

Feed-forward neural networks are used in many pattern recognition applications. They allow signals to travel in one direction from input to the output layer and only the current input influences the current output of the network [107]. In some applications, there is some information in the sequence itself. Therefore, using LSTM instead of a feedforward network could increase the accuracy of the system. LSTMs preserve the information of a sequence in hidden states and incorporate previous experiences in the current decision to a varying degree [36].

Memoryless Models

This work was started with a multilayer neural network as a deep model with no memory, similar to the model in chapter 2, and considered it as the baseline. For each task, the mean of all driving data (Vc) vectors related to the task was calculated. These compressed driving data (Vc) vectors were used as the input and driver behavior vectors were used (Vh) as the output of the multilayer neural network (Figure 48). After that, a Decision Tree Regressor which is a memoryless regression model was trained with the same dataset. Regression models in the form of a tree structure can be made with Decision Trees. A Decision Tree model breaks down a dataset into smaller subsets and at the same time, an associated tree structure is incrementally developed. In this tree structure, each Leaf node represents a decision on the numerical target. The “DecisionTreeRegressor” model in Sklearn was used, and the whole dataset was used for training the model.

Models with Memory

After building one deep and one shallow memoryless model, memory was added to the model using recurrent neural networks. A stacked multi input single output LSTM network (Figure 50 (a)) and all Vc vectors were used to consider the effect of all correlated driving data and dependency between them on estimating the driver behavior in the new model with memory. The details of training this model were discussed in the previous chapter. The hypothesis is that using the LSTM network increases the model's accuracy compared to the baseline memoryless model, since the assumption of neural networks, that all input samples are independent, is not true in driving, so a model with memory is a more suitable choice.

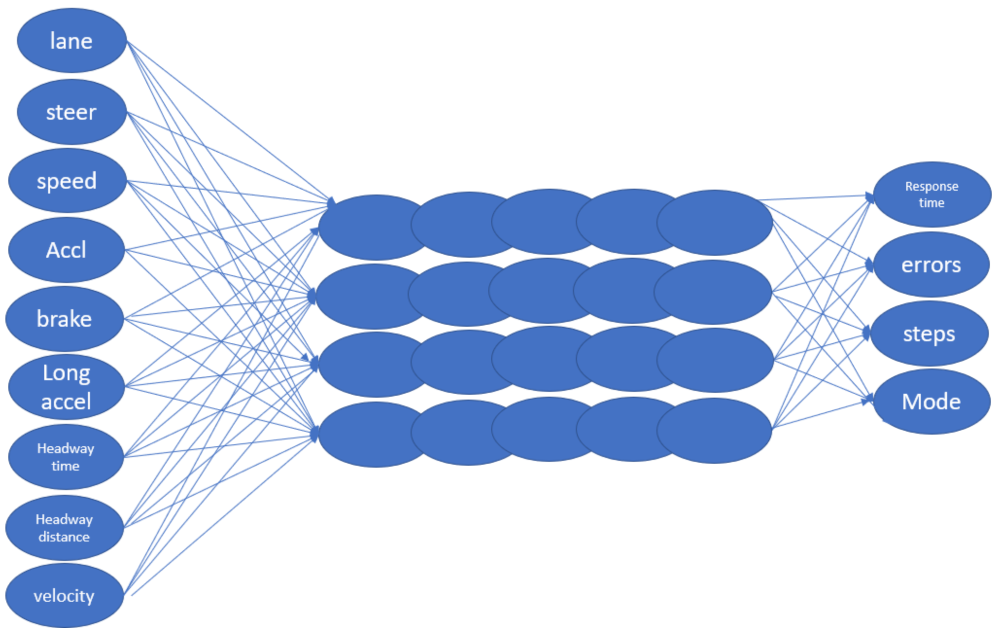


Figure 48. MLP for Driver Behavior Detection

Hidden Markov Model is another type of machine learning model. Hidden Markov Models are much simpler than Recurrent neural networks to train and interpret but they have a strong assumption that state transitions only depend on the current state, not on anything in the past which might not always be true. Moreover, a Markov chain only conditions on a fixed window and all states need to be defined in advance, so it is not a suitable choice when there isn't any pre-knowledge about the model's states and transitions. Furthermore, when the number of possible states and transitions is high, the

Hidden Markov Model can be computationally very expensive. The raw data set that was collected to use in the stacked LSTM model has 4,800,176 data vectors and 4368500 of them are unique. After preprocessing and compressing the dataset has 239,875 data vectors and 236408 of them are unique, so Hidden Markov Model is not a suitable choice for this dataset and model because the number of states is high, there isn't any pre-knowledge about the model's features, and the model is complex.

Model with Memory and Attention

Attention and memory are two features of intelligent models. A model with attention was built by adding a bidirectional LSTM layer and an attention layer on the top of the previous stacked LSTM network (Figure 50(b)). This model was able to learn the weight of each input sample in the input sequence on the final output. Attention primarily is used as a memory mechanism and it determines which part of the input sequence has more effect on the final output, hence greatly reducing the demand on resources. Attention mechanisms are used in some applications such as image captioning, language translation and document classification successfully. This mechanism finds the focus area related to each output in the input sequence, so it outperforms other recurrent networks that compress all the information of the input sequence in one vector. Considering the fact that driving data vectors can have different effects on the driver behavior data vector, using the attention layer might improve the model's accuracy.

In the previous model, in each step, LSTM cells have two inputs including the current input and the output of the previous step, so all previous inputs have an effect on the current step. A bidirectional LSTM layer provides access to both previous and next steps and an attention layer provides the weighted effect of each step on the output. The structure of an attention layer can be seen in Figure 49. In attention mechanism, $a_{t,i}$ is the weight of each step of the input sequence on the output. The attention mechanism calculates a score for each input sample considering both the result of the current input and the previous step's result. The score of each step can be calculated using (31). In (31) s_{t-1} is the output from the previous time step and h_i is the result of input x_i in the current time step. In the next step, scores are normalized using a "SoftMax" function (32) to calculate the final weight of each input sample on the final output. Larger a_{ti} shows more significant effect of input i on the model's output [109]. The hypothesis is that adding the attention layer to the model enhances both the model's accuracy and the training process compared to an LSTM network without attention.

$$e_{ti} = a(S_{t-1}, h_i) \quad (31)$$

$$a_{ti} = \frac{exp(e_{ti})}{\sum_{j=0}^{T} exp(e_{tj})} \quad (32)$$

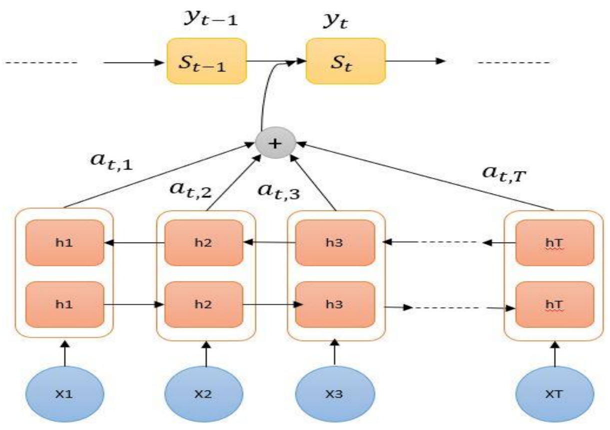


Figure 49. Attention Mechanism

Results

Results of Memoryless Models

A multi-layer (MLP) neural network was built and trained (Figure 51). In this network, 80% of data were used for training and 20% of them for testing. Besides, 20% of training data were used as validation data during the training process. This memoryless feedforward neural network was considered as the baseline to compare the results to the LSTM network, which is a deep network with memory. Different numbers of layers and hidden neurons were tried for the multilayer neural network.

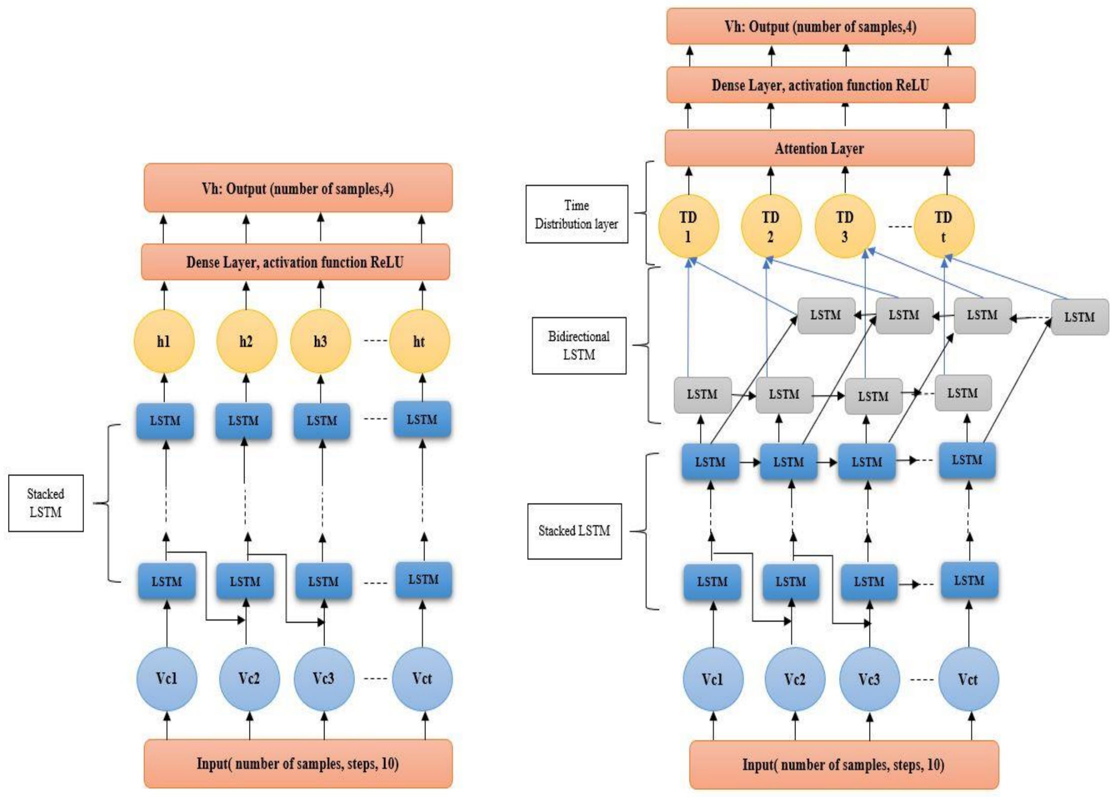


Figure 50. (a) Stacked LSTM Model (b) Stacked LSTM Model with Attention

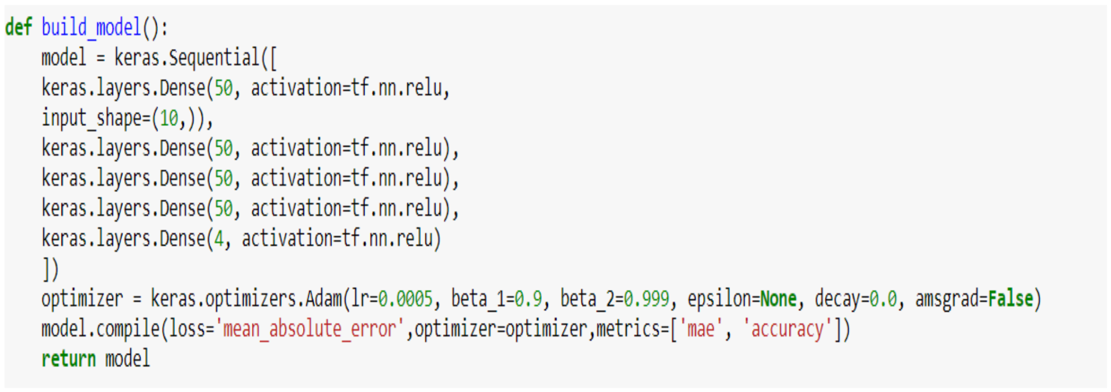

```
def build_model():
    model = keras.Sequential([
    keras.layers.Dense(50, activation=tf.nn.relu,
    input_shape=(10,)),
    keras.layers.Dense(50, activation=tf.nn.relu),
    keras.layers.Dense(50, activation=tf.nn.relu),
    keras.layers.Dense(50, activation=tf.nn.relu),
    keras.layers.Dense(4, activation=tf.nn.relu)
    ])
    optimizer = keras.optimizers.Adam(lr=0.0005, beta_1=0.9, beta_2=0.999, epsilon=None, decay=0.0, amsgrad=False)
    model.compile(loss='mean_absolute_error',optimizer=optimizer,metrics=['mae', 'accuracy'])
    return model
```

Figure 51. Deep Neural Network (MLP), Memoryless Model

Table 31 shows some of the achieved results with both scaled and unscaled data. For scaled data, four hidden layer model has the best performance, and this model with 50 hidden neurons reached the minimum test error and the best performance.

Less complex models, models with less hidden neurons and hidden layers, couldn't learn patterns in the dataset accurately and overfitted. Four hidden layers model with 50 hidden neurons shows the best performance with unscaled data. This deep memoryless model was considered as the baseline. After training a deep memoryless model, Decision Tree Regressor was chosen and was trained with both whole and a subset of the unscaled dataset. Considering the whole data set the train error is 0.0007 and the test error is 1.439. It shows that the model is completely overfitted, and it just remembers data instead of learning the patterns. The model with scaled data reached zero train error (0.00000081) and 0.15 test error. A dataset of 1000 unscaled data samples was tried and the model had almost the same performance. The train error decreased to absolute zero and the test error increased to 1.64. In sum, the shallow memoryless model doesn’t have acceptable performance and deep models are a more suitable choice for the application and collected dataset.

Table 31. MLP Results

layer	Hidden	Train MAE	Val MAE	Test MAE
4-not scaled	50	0.76	1.38	1.33
4-not scaled	150	0.4	1.4	1.46
4-not scaled	300	0.17	1.49	1.39
3-not scaled	50	0.76	1.44	1.38
3-not scaled	150	0.33	1.39	1.48
3-not scaled	300	0.31	1.4	1.34
2-not scaled	50	0.95	1.4	1.38
2-not scaled	150	0.54	1.49	1.41
2-not scaled	300	0.39	1.45	1.51
4- scaled	50	0.089	0.1367	0.24
4- scaled	150	0.012	0.14	0.2
4- scaled	300	0.045	0.13	0.16
3- scaled	50	0.096	0.12	0.24
3- scaled	150	0.056	0.13	0.32
3- scaled	300	0.046	0.14	0.21
2-scaled	50	0.11	0.14	0.27
2-scaled	150	0.082	0.14	0.23
2-scaled	300	0.06	0.15	0.46

Results of Models with Memory

A stacked LSTM model that has memory was built and trained, but it doesn’t have attention. The details of the training process mentioned in the previous chapter. In sum, different numbers of layers from 1 to 6 and a large range of LSTM neurons from 10 to 1000 were tried. In the LSTM network, 80% of data were used for training and 20% of them for testing. Besides, 20% of training data were used as validation data during the training process. Table 32 shows some of the achieved results. In this table, “mae” is mean absolute error and “mse” is mean square error. The model with four LSTM layers had the lowest error compared to shallower networks, but the one-layer LSTM model had the minimum gap between train and test error.

It was expected that the LSTM model, which has memory, shows better performance compared to the memoryless multilayer neural network. The minimum mean absolute error of multi-layer neural network for test set was 0.4 more than the best result of the stacked LSTM network. Besides, the gap between train and test error in different combinations of layers and neurons that we tried increased significantly in a multilayer neural network compared to the LSTM model. In sum, using the LSTM network instead of a feedforward neural network decreased the mean absolute error of the model and reduced the overfitting problem. On the other hand, the LSTM model is computationally more expensive than a feedforward neural network. In order to keep the advantages of LSTM model, improve the performance of the model and reduce the computational expenses of it, attention mechanism was added to the model.

Table 32. Stacked LSTM Model

LSTM layer	Hidden neuron	Train mae	Train mse	Val mae	Val mse	Test mae	Test mse
one	100	0.84	1.4	1	2.7	1.08	3.13
two	300	0.36	0.24	0.98	2.58	1.03	3.39
Three	50	0.4	0.32	1.01	3.33	0.98	2.88
four	50	0.57	0.61	0.97	3.36	0.9	2.19

Results of the Model with Memory and Attention

The stacked LSTM model with attention was also built and trained using the same train and test dataset. Similar to the previous models, different combinations of hyperparameters including the number of LSTM layers and LSTM cells were tried. In this model, a bidirectional LSTM layer on the top of the LSTM layers provides access to all data in the input sequence in each step. In the first step, the performance of this type of LSTM layer was compared to the regular LSTM layer. Bidirectional LSTMs can enhance the performance of the model on sequence classification problems when all time steps of the input sequence are available. Bidirectional LSTM networks train two LSTMs on the input sequence. The first one on the input sequence from the first step data to the last one, and the second one on a reversed copy of the input sequence. This can provide additional context to the network and result in faster and even fuller learning on the problem [114].

Before building and training the attention model, both the bidirectional LSTM network and the LSTM network were tried to estimate driver behavior data vector using 10 driving data as the input of the network. In these models, 80% of the data was used for training and 20% for testing these models. The assumption is that using a bidirectional LSTM network enhances the training process compared to a simple LSTM network. First, a bidirectional LSTM network was built. Adam optimizer was used as the model's optimizer and Mean Absolute Error (MAE) as the accuracy of the model. A wide range of LSTM neurons from 10 to 500 were tried. Table 33 shows some results. The best result was achieved with 300 neurons and gave 0.97 training and 1 testing MAE.

Table 33. Bidirectional LSTM Network Results

LSTM neurons	Train MAE	Test MAE
150	1	1.15
200	0.95	1
250	1.04	1.1
300	0.97	1

After that, a simple LSTM network was trained with the same train and test dataset to analyze the positive or negative effect of using a bidirectional LSTM network instead of a simple LSTM network on the model's accuracy. Adam was used as the model's optimizer and MAE as the accuracy parameter of it. A wide range of a number of neurons was used and the best results were achieved in the range 100 to 1000. Table 34 shows some of the achieved results of a simple LSTM network. The best result of this network is 0.84 training and 1.08 testing MAE. Using a bidirectional LSTM network instead of a simple one-directional network decreased the model's train and test error. Additionally, it decreased the gap between train and test error significantly, so using a model with memory that considers the effect of both previous and next samples on the current step's output, improves the accuracy of our model for estimating driver behavior. Although train and test error didn't decrease significantly, using the bidirectional LSTM network significantly diminished the overfitting of the model.

In the next step, the stacked LSTM model with attention was built which includes stacked LSTM, bidirectional layer, time distribution and attention layer. Adam was chosen as the optimization function and 100 was chosen as the batch size. A large range of LSTM neurons in both bidirectional and LSTM layers was tried to find the most accurate combination of hyperparameters for this model.

Table 34. LSTM Network Results

LSTM neuron	Train MAE	Test MAE
100	0.84	1.08
300	0.64	1.15
500	0.89	1.2
1000	0.7	1.16

Table 35 shows some of the best achieved results of the LSTM network with attention network. The first column shows the number of LSTM cells in the bidirectional LSTM layer and time distributed layer. Time Distributed layer applies the same fully connected (dense layer) operation to every time-step to keep one-to-one relations on input and output. The second column shows the number of LSTM layers (one direction LSTM layers) and the third column shows the number of LSTM cells in each of these LSTM layers. Based on these results, adding the attention layer had a positive effect on the accuracy of the model. The model with the attention layer had less test error and a smaller gap between train and test error compared to the LSTM model with no attention.

Figure 52 compares the performance of the MLP network, the LSTM network and the LSTM attention network. The minimum train and test mean absolute error of stacked LSTM was 0.57 and 0.9. Using the attention layer and one bidirectional LSTM layer on the top of the stacked LSTM decreased the train and test error significantly. The best result of this model is 0.69 train and 0.75 test mean absolute error which is a 17% reduction of test error compared to stacked LSTM network and 32% reduction of test error compared to the multi-layer neural network. Moreover, the stacked LSTM model with attention layer trained faster than the stacked LSTM model with almost the same number of LSTM neurons.

The attention model needed between 50 to 100 epochs training to reach the accuracy that the stacked LSTM model achieved after at least 200 epochs. Adding the attention layer made our model less computationally expensive while keeping the advantages of the LSTM model and slightly reducing the model’s error.

Conclusion

A memoryless MLP neural network was used and it was compared with two models that use a sequence of data samples to decide about the driver's behavior. Ten driving data were used to predict driver behavior features while interacting with the car’s infotainment system. One distractive behavior was considered in this model, which is interacting with the car infotainment system, and four features were defined for it.

Table 35. Results of Stacked LSTM Model with Attention

Bidirec/ Time distribute	LSTM neuron	Train MAE	Train MSE	Val MAE	Val MSE	Test MAE	Test MSE
10, 10	10, 10,10	0.77	1.42	0.9	2.17	0.92	2.64
40,40	20,20	0.77	1.29	0.88	2.69	0.85	1.89
40,40	20,20,20	0.76	1.31	0.89	2.12	0.86	1.99
40,40	40	0.82	1.62	0.9	2.74	0.93	2.15
40,40	40,40	0.75	1.32	0.92	2.61	0.88	2.03
50,50	25,25	0.72	1	0.87	1.86	0.81	1.66
50,50	50,50	0.74	1.29	0.82	2.1	0.84	1.94
50,50	50,50,50	0.73	1.07	0.84	1.84	0.85	1.9
100,100	50,50	0.66	0.9	0.82	1.7	0.86	1.95
100,100	50,50,50	0.74	1.17	0.85	1.98	0.82	1.72
100,100	100	0.73	1.25	0.92	2.37	0.81	1.8
100,100	100,100	0.65	0.86	0.73	1.37	0.76	1.35
100,100	100,100,100	0.69	0.96	0.78	1.67	0.79	1.5
150,150	100	0.73	1.2	0.9	2.13	0.81	1.68
150,150	100,100	0.67	0.9	0.75	1.51	0.79	1.19
150,150	100,100,100	0.7	1.01	0.79	1.54	0.79	1.47
150,150	150,150	0.71	1.15	0.84	1,98	0.82	1.85
150,150	150,150,150	0.69	1	0.76	1.51	0.75	1.26
200,200	200,200,200	0.73	1.16	0.81	1.75	0.85	1.83

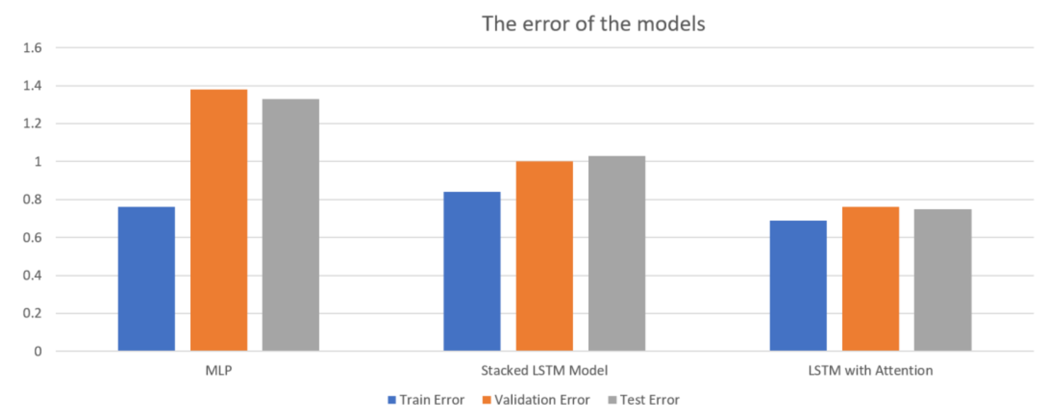


Figure 52. Comparing the Error of the MLP, LSTM and LSTM Attention Models

A multilayer neural network, a stacked LSTM network and a stacked LSTM network with attention layer were trained. The minimum train and test error of the stacked LSTM model were 0.4 less than MLP, so it outperformed MLP since it considers dependency between input data. Besides, adding the attention layer to the LSTM network enhanced the model's accuracy significantly since it considers the weighted effect of each data in input sequence on the current output. The minimum train and test error of stacked LSTM were 0.57 and 0.9 and adding attention decreased the train and test error to 0.69 and 0.75. Results also show diminished overfitting problem since the gap between train and test error reduced from 0.33 in the LSTM model to 0.06 in the attention model.

Different factors such as the length of distraction have an effect on the severity level of a distracting task. Four features were defined and combined into a driver vector (Vh). It was shown that different combinations of these features, when automatically inferred from corresponding measurements of how the car is driven, (Vc) could provide a good indication of a specific distracting task, namely the driver's interaction with the car's infotainment system. This inference could be used in an intelligent decision system to reliably determine if the driver was doing distracting tasks and what the severity level of distraction is.

THE ENCODER DECODER MODEL

Deep Neural Networks are powerful models that outperform shallow machine learning models in many complex and difficult tasks such as image processing, natural language processing and sentiment analysis. Although, neural networks are a suitable choice for large and complex labeled data sets, they unable to map sequences to sequences. Sequence to sequence learning is about training models to convert sequences from one domain to sequences in another domain. In applications that there is some dependency between samples of both the input sequence and the output sequence, using sequence to sequence learning instead of one to one model can improve the training process and the model's accuracy. For example, in machine translation application considering dependency between words in the input sentence can reduce ambiguity in selecting the most suitable word in the output sentence that can represent this word. Besides, sequence to sequence learning can handle different input and output sequence length in different domains. For instance, in image captioning, the output sequence length can't be defined in advance, moreover, there is a dependency between samples of output sequence and the samples of the input sequence are dependent, so sequence to sequence model is the most suitable choice for this application. The following are some examples of sequence to sequence applications.

1. Language translation: the input sequence is a sentence in one language (i.e. French) and the output is a correlated sentence in another language (i.e. English)
2. Speech translation: the input is a sequence of audio samples and the output is the text transcription of the input audio.

3. Learning to execute calculate the outcome of small programs.
4. Image captioning: the input is an image and the model generates a text that describes the image.
5. Conversational modeling generates answers to textual questions.

Movement Classification, e.g. generating a sequence of commands from a sequence of gestures. In driver behavior analysis both the length of a task and the type of task provide valuable information about the driver state. The level of driver distraction depends on several factors such as the driving context, type of task, the length of the distracting task and the driver’s experience and driving skill. The severity of a distracting task might increase significantly if the task continuous for several seconds or even minutes. In previous chapters, the positive effect of using models with memory on the accuracy of driver behavior detection was discussed. Using the LSTM model instead of memoryless models reduced the model’s error and enhanced the training process significantly. The features of one task were estimated using a sequence of driving data vectors. In this chapter, a sequence to sequence model was used to detect a sequence of driver behavior using a sequence of driving data vectors. If a driver does a distracting task and comes back to normal behavior soon, the level of distraction is low compared to a driver that has a continuous inattention for several seconds. Estimating a sequence of driver behavior can provide valuable information about the severity of driver distraction.

An encoder decoder LSTM model was used for detecting a sequence of driver behavior using a sequence of driving data. Encoder decoder is a sequence to sequence model that initially was used for machine translation. RNN Encoder-Decoder consists of two recurrent neural networks that one of them acts as an encoder and another one as the decoder. The encoder maps a variable-length input sequence to a fixed-length vector, and the decoder maps the fixed-length vector to a variable-length output sequence. Sequence-to-sequence prediction problems are challenging because the number of items in the input and output sequences can vary. For example, machine translation and image captioning are examples of sequence to sequence problems with varying input and output lengths. The structure of the language translation model was used and adapted to the data features of the collected dataset.

Related Works

Sequence to sequence learning is successfully used in an increasingly larger number of applications such as language translation. In these applications, there is some information in input and output sequence and data samples are not independent of each other. Encoder decoder is one of sequence to sequence models that has many successful applications including Machine Translation, Learning to Execute, Image Captioning and Conversational Modeling. Following are some of these applications in different areas [115-117].

[118] Proposed a model called RNN Encoder-Decoder. This model is a sequence to sequence model consists of two recurrent neural networks. One of them called encoder that encodes a sequence of symbols into a fixed-length vector representation, and the second one called decoder that decodes the representation vector to another sequence. The encoder and decoder are jointly trained to maximize the conditional probability of detecting the target sequence using the input sequence. The performance of a statistical

machine translation model improved by using the RNN Encoder-Decoder as an additional feature in the existing log-linear model. The results show that the proposed model learns a semantically and syntactically meaningful representation of linguistic phrases.

[119] Proposed a sequence to sequence model to evaluate the performance of a short computer program. A simple class of programs was considered as the input of this model. This class of programs can be evaluated using a constant memory and a single left-to-right pass. The results show that LSTMs can map the character-level representations of this type of program to their correct outputs. For this model, it was necessary to use curriculum learning, and since conventional curriculum learning proved ineffective, a new variant of curriculum learning was defined that improved the networks' performance in all experimental conditions. The improved curriculum had a significant positive impact on an addition problem, making it possible to train an LSTM to add two 9-digit numbers with 99% accuracy.

[120] Introduced a sequence learning approach that has a minimum assumption on the input and output sequences. This model uses a multilayered LSTM to map the input sequences with various lengths to a fixed size vector and convert this vector to the output sequence. It reached the BLUE score of 34.8 using the WMT-14 dataset. The success of this model on machine translation, translation from English to French, can suggest that this is suitable for many other sequence learning applications. The LSTM network in this model makes the translation of long-term phrases feasible. This model was adapted to the

dataset that was collected in Experiment 3 to predict the sequence of driver behavior using a sequence of driving data with the same or different sizes.

Video captioning is another application of sequence to sequence learning. Models for video captioning need to be sensitive to temporal structure and they should accept variable lengths of input and output sequences. [121] Proposed a model using the LSTM network that generates captions for videos. The model is an encoder-decoder model that converts a video as the input of the model to a text that describes the movie. They use movie frames as inputs of a pretrained convolutional neural network, which pretrained on 1.2 Million images ILSVRC-2012 object classification subset of the ImageNet dataset and uses the output of the top layer of a convolutional neural network as the input of LSTM unit. Moreover, they used optical flow measures as an additional input sequence in their architecture. This architecture had a good performance on two large and challenging movie description datasets and great performance on the MSVD dataset.

The encoder-decoder system has been used in a few driver behavior analysis methods. For example, a driver behavior visualization method is proposed in [122] that used continuous driving behavior and a sparse autoencoder to extract features of driving behavior. Multiple sensors were used to collect driving behavior data and a deep sparse autoencoder (DSAE) was used to find the most important features of a large number of driving data features. They proposed a method based on DSAE called driving color map. This method can visualize driver behavior by mapping the extracted 3D hidden features to the RGB color space. This method used external sensors and devices such as a camera to collect driving data and images of the environment were used as one of the driving data. Researchers used collected data by external devices to train an encoder-decoder model to estimate driver behavior. They added several external devices, such as cameras and sensors to the car to collect driving data and other cars situation. In another word, they used one of the common applications of the encoder-decoder model which is image processing. To the best of our knowledge, no research has used the encoder-decoder model to estimate driver behavior only using data that extracted from the car data buses.

The Encoder Decoder Model

Sequence to sequence learning has emerged as a new paradigm in machine learning and pattern recognition area. It is a sequence forecasting problem that uses a sequence of input samples and provides a sequence prediction as the system's output [35]. Sequence to sequence learning is about training a model to convert sequences from one information space to sequences in another information space such as language translation which converts a sequence of words in one language to a sequence of words in another language or image captioning which converts a sequence of image pixels to a sequence of words. This type of learning method has been a challenging prediction problem since the length of input and output sequences could be different [123]. One structure for this model that can handle various lengths of data sequences and has been effective in many different applications such as language translation and sentiment classification is called the encoder-decoder LSTM model. It consists of two LSTM networks that act as an encoder-decoder pair. The encoder converts a vary length input sequence to a fixed-length vector and the decoder maps this vector to a varying length output sequence. Encoder decoder LSTM networks have many successful applications including Machine Translation, Learning to Execute, Image Captioning and Movement Classification [115, 117]. Figure 53 shows the machine translation architecture that maps a sentence in English language to a sentence in French language.

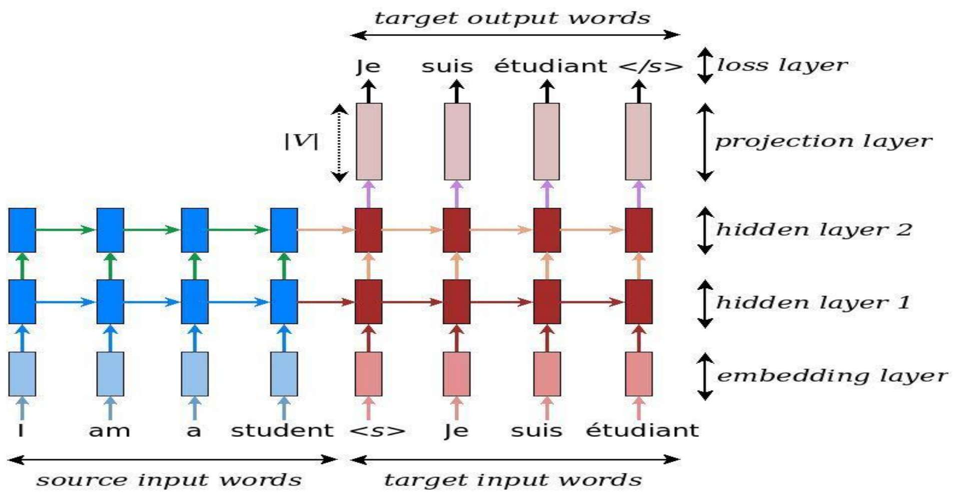


Figure 53. The Encoder Decoder Machine Translation Model [124]

In this machine translation model, the input is a sequence of words and a sign that shows the end of the sentence as the last word of the sentence. The words of an input sequence are converted to a hot vector, so all input samples of the input sequence have the same number of features. The length of the input sequence depends on the length of the sentence and can vary. The encoder converts the input sequence to a representation vector and this vector is used as the input of the decoder. The Decoder converts the representation vector to a various lengths sequence of words in another language and each word in this sequence is a hot vector, so similar to the input sequence all words in the output sequence have the same number of features.

The encoder is made up of two parts including an input layer, LSTM layers, and the decoder has four parts including an input layer, LSTM layers, dense layer and output.

The machine translation model was modified based on the data features of the collected dataset to predict a sequence of driver behavior using a sequence of car-related data. Each Driving data vector is like one word in the input sequence of the language translator and the features of them, input features, are like the maximum number of words' letters. Each driver behavior data vector has one corresponding driving data vector which is the mean of all driving data vector during the task, so the input sequence and output sequence have the same size. Figure 54 shows a simple architecture of the encoder-decoder model.

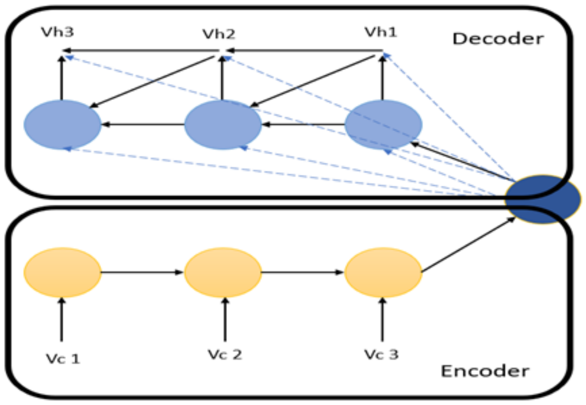


Figure 54. An Encoder Decoder Architecture

In all, 2025 driver data vectors have been collected, one for each task, and during each task, a large number of driving data vectors have been collected. To map the driving data vectors to the corresponding driver data vector, the driving data vectors were divided based on the length of response time to blocks with different lengths and each block was mapped to one driver data vector. The sum of the response time was calculated for volunteer X in mode Y and the portion of driving data that is correlated to each driver data vector was calculated. Then, the average of each driving data block was calculated and mapped to the corresponding driver data.

Results

A variety of combinations of hyperparameters such as batch size, activation functions and the number of LSTM neurons were tested. Several batch sizes in the range of 1 to 200 were tried. Table 36 shows some of the achieved results after 100 epochs. Using small batch size resulted in an obvious overfitting problem and as the batch size was increased, the overfitting problem reduced. Using batch size 100 resulted in the minimum train and test error after 100 epochs and it was chosen as the most suitable choice for the model.

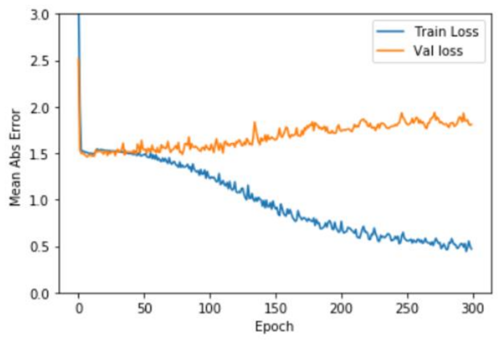
Table 36. Batch Size Effect on the Accuracy

Batch size	Train MAE	Valid MAE	Test MAE
1	0.24	1.75	1.78
50	1.19	1.65	1.79
100	1.23	1.58	1.56

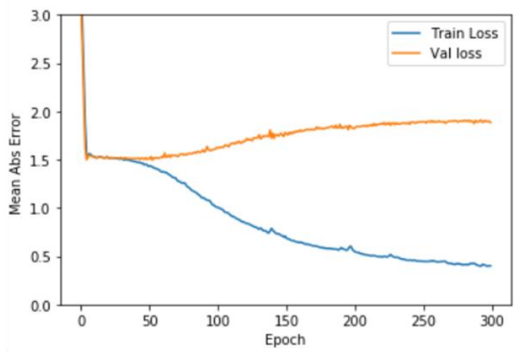
There are several activation functions that can be used in LSTM networks including Adam, Adagrad, Adadelta and rmsprop. A decoder-encoder with the batch size 100, 50 hidden neurons was built, and it was trained with all these activation functions for 300 epochs. Also, 300 was chosen as the number of epochs to see the long-term effect of each activation function. Although all three networks were overfitted the Adagrad shows smoother change during 300 epochs. Table 37 shows the achieved results. The model with Adagrad generated the best result, and Adam had the largest error (Figure 55).

Table 37. Effect of Optimization Functions on the Model's Accuracy

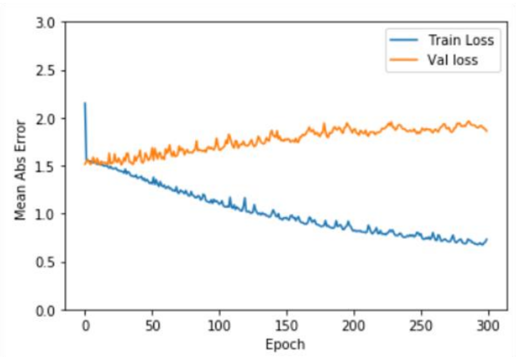
Activation Function	Train MAE	Valid MAE	Test MAE
Adam	0.72	1.86	1.92
Adagrad	0.33	1.88	2.2
Adadelta	0.46	1.8	1.89



(a)



(b)



(c)

Figure 55. (a) Adadelta Optimization Function (b) Adam Optimization Function (c) Adagrad Optimization Function

Between 20 to 1000 LSTM neurons were tried for the LSTM layer. The dataset was divided into three parts of train, validation and test data. In this model, 80% of data was used for training and 20% for the testing process. In the training process, 20% of training data were used as the validation dataset. Three models including three input and three output steps, four input and four output steps and five input and five output steps were trained using the same train, test and validation dataset. Adagrad was used as the activation function and 100 as the batch size for all models.

Three Input and Three Output Steps

Both scaled and un-scaled data were tried for this model. The model was trained with a large range of LSTM neurons from 20 to 1000. Table 38 shows some of the achieved results. For unscaled data, the model overfitted after around 100 epochs, and when the number of LSTM neurons was increased, the model overfitted faster. The best result achieved with 150 LSTM neurons. Using more than 150 LSTM neurons, increased the model's error. Figure 56 compares the performance of the model with 150 neurons and the model with 250 neurons after 100 epochs of training. To reduce the overfitting problem, this model was trained with scaled data.

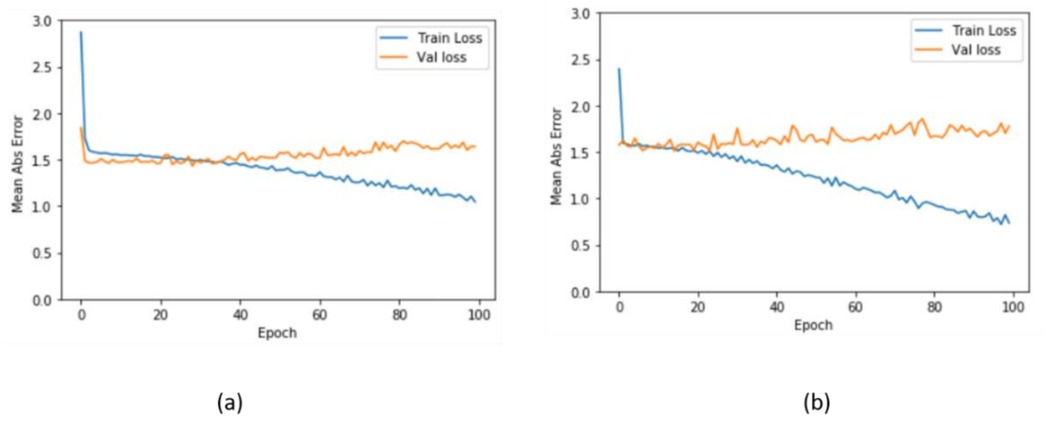


Figure 56. (a) Three Input and Three Output Steps Model with 150 Neurons Unscaled Data (b) Three Input and Three Output Steps with 250 Neurons Unscaled Data

With scaled data, the model could be trained longer without overfitting, but the performance of the model was similar to the model with unscaled data. Increasing the number of LSTM neurons to 150 had a positive effect on the model’s performance and after that, the error increased and overfitting started in lower epochs. Figure 57 shows the loss of the model with 125 LSTM neurons. The model continued learning smoothly until 1000 epochs, but when more LSTM neurons were added, the model went toward overfitting after around 700 epochs. In sum, using scaled data enhanced the training process and decreased the overfitting problem, but increasing the number of neurons with both scaled and unscaled data resulted in a larger error and between 100 to 150 neurons was the best range for this model.

Table 38. Three Input and Three Output Steps Results

LSTM Neurons	Data	MAE train	MAE valid	MAE test
50	Un-scaled	1.38	1.49	1.55
150	Un-scaled	1.51	1.62	1.53
300	Un-scaled	1.51	1.65	1.56
500	Un-scaled	1.85	1.89	1.85
125	Scaled	0.156	0.172	0.17
250	Scaled	0.1	0.19	0.2
500	Scaled	0.1373	0.1652	0.18
1000	Scaled	0.3786	0.3905	0.38

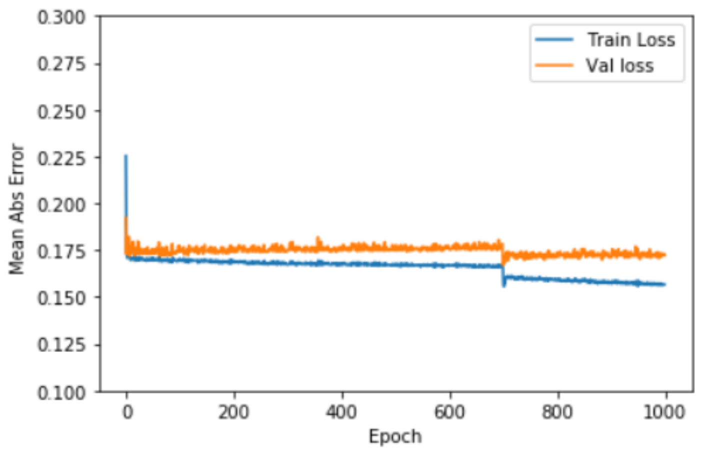


Figure 57. Three Input and Three Output Steps Model 150 Neurons and Unscaled Data

Four Input and Four Output Steps

A model with four input and four output steps was built and trained with scaled and unscaled data. Table 39 shows some of the achieved results. The model has a better performance compared to three steps model for both scaled and unscaled data. The best result of unscaled data achieved with 300 LSTM neurons and increasing the number of neurons more than 300 increased the test error and the gap between train and test error significantly. Figure 58 shows the loss of the model with 300 neurons and unscaled data. For scaled data, the model had less overfitting problem and the performance improved as more LSTM neurons were added. The best performance achieved with 500 LSTM neurons. After that, adding more neurons resulted in a less accurate model.

In sum, four steps model had better performance in all cases compared to three steps model and the best performance achieved with between 300 to 500 LSTM neurons. We decided to try more steps to see if considering longer dependency can improve the model’s accuracy.

Table 39. Four Input and Four Output Steps Results

LSTM neurons	Data	MAE train	MAE valid	MAE test
50	Un-scaled	1.52	1.47	1.53
200	Un-scaled	1.53	1.63	1.55
300	Un-scaled	1.5	1.59	1.5
1000	Un-scaled	1.32	1.59	1.6
125	Scaled	0.1512	0.1569	0.16
250	Scaled	0.1655	0.1711	0.16
500	Scaled	0.151	0.166	0.15
1000	Scaled	0.2952	0.2817	0.29

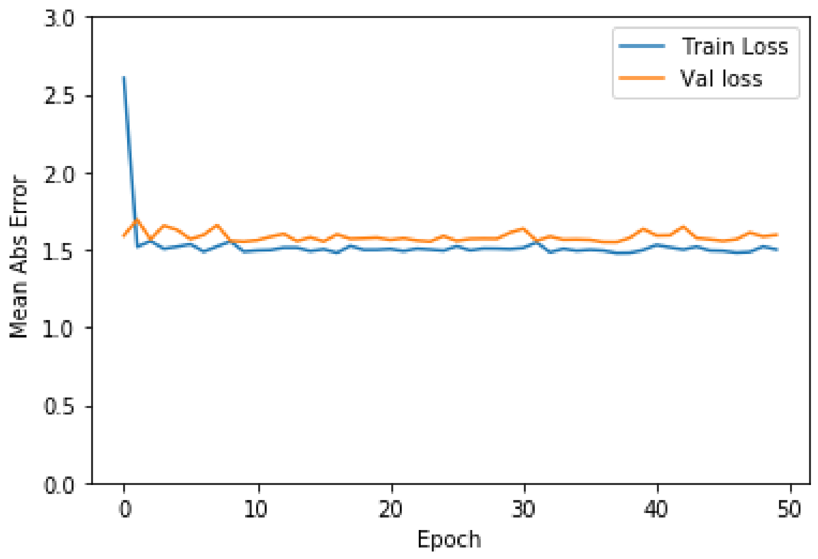


Figure 58. The Loss of Four Input and Four Output Steps Model with 300 LSTM Neurons and Unscaled Data

Five Input and Five Output Steps

Both scaled and unscaled data were used for this model and the hyperparameters of the previous models were used for training. Table 40 shows some of the achieved results. The performance of unscaled data is better than three steps model and it is almost similar to the four steps model. For unscaled data, although the four steps model has less error in range 125 to 500 neurons, five steps model shows better performance with a larger number of LSTM neurons. This model with 125 LSTM neurons and unscaled data resulted in the minimum error (Figure 59). Besides, the model best accuracy with scaled data achieved with 250 LSTM neurons. In sum, four steps model had the least error and the best performance for both scaled and unscaled data compared to two other models. The minimum error for unscaled data is 1.5 train and 1.5 test mean absolute error (MAE). For scaled data, the best result is 0.151 train and 0.15 test MAE.

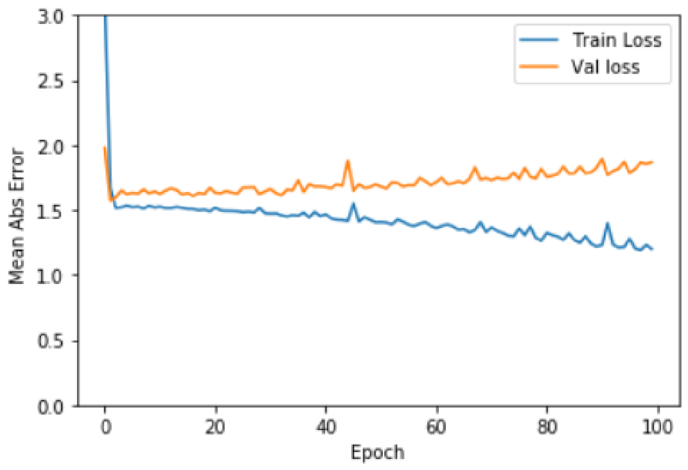


Figure 59. Five Input and Five Output Steps Model with 125 LSTM Neurons and Unscaled Data

Table 40. Five Input and Five Output Steps Model

LSTM neurons	Data	MAE train	MAE valid	MAE test
50	Un-scaled	1.51	1.61	1.54
100	Un-scaled	1.48	1.58	1.53
125	Un-scaled	1.52	1.53	1.52
1000	Un-scaled	0.93	1.94	1.95
125	Scaled	0.158	0.17	0.17
250	Scaled	0.1618	0.1718	0.16
500	Scaled	0.161	0.174	0.16
1000	Scaled	0.2496	0.2668	0.25

More Input Samples

In the previous section, the accuracy of the encoder-decoder model when the input and output sequences have the same size was discussed. The number of input steps was increased and instead of mapping the whole driving data block to one driver vector each block was divided to seven parts and we calculated the average of each part. Consequently, for each driver vector, there are seven corresponding driving vectors. Three different models were built including 21 input and 3 output steps, 28 input and 4 output steps, 35 input and 5 output steps. The rest of hyperparameters of models with more input vectors is similar to previous models. The goal is to analyze the effect of increasing the input sequence length on the training process and the model’s accuracy. Table 41 shows a summary of the achieved results with scaled data.

Table 41. More Input Vectors with Scaled Data

	21 inputs to 3 outputs		28 inputs to 4 outputs		35 inputs to 5 outputs	
LSTM neurons	Train MAE	Test MAE	Train MAE	Test MAE	Train MAE	Test MAE
50	0.151	0.16	0.153	0.16	0.163	0.16
125	0.15	0.17	0.155	0.15	0.163	0.16
250	0.15	0.16	0.152	0.16	0.155	0.15
500	0.38	0.39	0.154	0.15	0.172	0.17

In most cases using more information in the input sequence improved the accuracy of the models with scaled data. The performance of the model with 21 input and 3 output steps and scaled data improved for all cases except the model with 500 LSTM neurons. Besides, the model trained much faster compared to three input and three output steps model. Figure 60 compared these two models and shows that the model with more input vectors reached the same accuracy 700 epochs earlier. The performance of four steps and five steps models with scaled data didn't show improvement as much as three steps model. Although in some cases of four step and five step models the error decreased, the enhancement isn't significant, and it can be because of different weight initialization.

These three models were trained with unscaled data and the same hyperparameters. Table 42 shows the best achieved results. With more input vectors, the 21 input and 3 output steps model shows better results for one case which is the model with 500 LSTM neurons and the rest of cases reached larger error compared to the three input and three output steps model. Adding more data vectors to the model decreased the minimum error of three steps model from 1.56 to 1.5 test MAE.

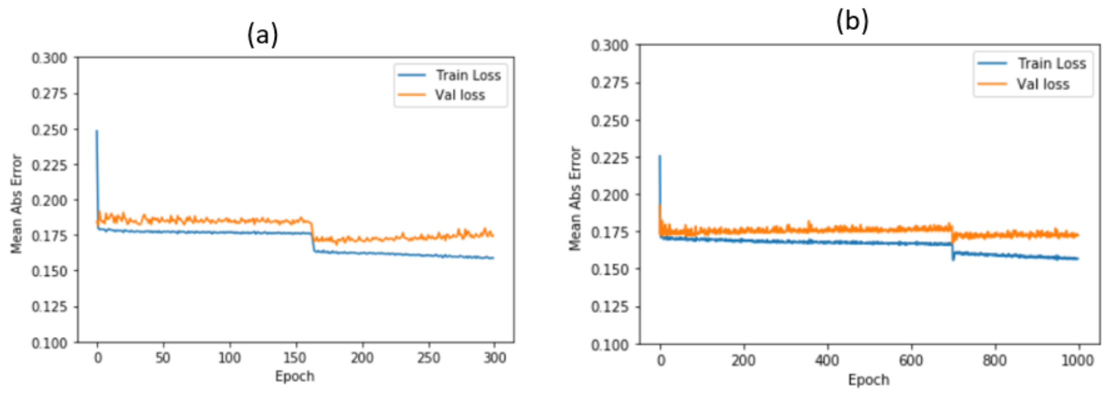


Figure 60. (a) The Performance of 21 Input to 3 Output Steps Model (b) The Performance of Three Input and Three Output Steps Model

Using more inputs didn't have a notable positive effect on the performance of almost all four steps and five steps models. Only 35 input and 5 output steps models show some enhancement and it reached the minimum error between all models which is 0.47 train and 0.48 test MAE that is 0.4 less than five input and five output steps model. In sum, adding more input vectors improved the performance of two out of three models and it didn't have much effect on the four input and four output steps model that had the best performance between models with the equal length of input and output sequence.

Conclusion

Driver behavior analysis is one of the solutions to learn driver behavior patterns and detect abnormal behavior. A variety of machine learning methods such as neural networks, decision trees and Markov models were used in driver behavior analysis. Deep learning methods in many driver safety applications outperform traditional machine learning methods. Driving data, driver status and combination of these two were used in these deep learning and machine learning approaches.

Table 42. More Input Vectors Models with Unscaled Data

21 input 3 output steps		
LSTM neurons	Train MAE	Test MAE
50	1.39	1.63
125	1.33	1.64
500	1.41	1.5
28 input 4 output steps		
LSTM neurons	Train MAE	Test MAE
50	1.51	1.52
100	1.53	1.53
200	1.50	1.52
35 input 5 output steps		
LSTM neurons	Train MAE	Test MAE
25	1.52	1.53
50	1.47	1.48
200	1.53	1.54

Some of these methods consider the input data of the model independent from each other, but this assumption is not true for driving data, so using deep learning methods that can extract and learn information in the sequence of data can enhance the performance of driver behavior analysis methods. An encoder-decoder LSTM model was used to estimate a sequence of driver behavior data (Vh) using a sequence of driving data (Vh). Two cases including the same input and output sequence length and using more input data vectors as the input sequence were tested. Using more input vectors for each driver behavior vector didn’t have a notable positive effect on the performance of most of the models and only reduce the error of a few of them. In addition, more input vectors resulted in a more complex and more computationally expensive model, so the model with the same input and output sequence length was chosen. In the next chapter, the attention mechanism was used to enhance the performance of the current model, and the results of the encoder-decoder model with LSTM were compared with the sequence to sequence learning with an attention mechanism.

THE ENCODER DECODER WITH ATTENTION MODEL

Deep learning methods outperform traditional machine learning models in the car safety applications such as lane detection, driver distraction detection and driver behavior analysis. Some deep learning and machine learning methods such as RNN, Markov Model (MM) and BDRNN have memory as a result, they are able to extract and learn information in a sequence of data. Different types of sequence learning methods were used in car safety and autonomous vehicle applications successfully, and they enhanced the performance of these applications compared to memoryless methods. The encoder decoder is a specific model for the sequence to sequence learning models that have been used in many applications such as language translation, image captioning and conversation generating successfully. The attention mechanism can enhance the performance of these encoder-decoder models by considering the weighted effect of each sample in the input sequence on each sample in the output sequence. Attention is one of the most influential ideas in Deep Learning area. Although it is now used in many different problems, it was initially designed in the context of Neural Machine Translation using sequence to sequence model.

The encoder-decoder architecture composed of two parts including the encoder and the decoder. The encoder processes the input sequence and summarizes the information into a context vector of a fixed length. This representation is expected to summarize the entire input sequence accurately. The decoder is initialized with this context vector and generates the transformed output. A critical problem of this fixed-length context vector design is the incapability of the model to remember long sequences, and it often forgets the earlier parts of the long sequence once it has processed the entire sequence. The attention mechanism was suggested to solve this problem.

The idea behind the attention mechanism is to use all input steps to create the context vector for each output and use this specific context vector to generate one output step. In the previous chapter, an encoder-decoder was used to estimate a sequence of driver behavior data vectors using a sequence of driving data vectors with the same length or larger. In this chapter, the attention mechanism was used as a sequence to sequence model and it enhanced the training process and the performance of the sequence to sequence model compared to the encoder-decoder model. In Chapter 5, using the attention mechanism on the top of the stacked multi input simple output LSTM model significantly enhanced the model performance. In the current sequence to sequence model, the input sequence length is not as large as the model in Chapter 5, but still using the attention mechanism might improve the model's performance. Besides, the performance of this model was compared with MLP which is a single output model.

In the next step, some rules were defined to detect the level of distraction. These rules use the combination of a sequence of driver data features to estimate the level of distraction. Five levels of distraction were defined for the Intelligent co-driver safety system. Finally, the possible reaction to each level of distraction and the future work which can enhance this system performance were discussed.

Related Works

Sequence to sequence learning with encoder-decoder models was used successfully in different applications such as language translation, natural language processing, image classification and driver behavior analysis [115, 117, 118]. Adding attention mechanism to the encoder-decoder enhanced the performance of sequence to sequence learning applications especially in natural language processing [125, 126].

Image captioning is one of the successful applications of attention-based models. [127] Discussed a new model that describes the content of images automatically using the attention mechanism. The model defined based on language translation with the attention mechanism model. The model takes a raw image as the input and produces a caption, which is a sequence of words, to describe the image. In this model, the encoder uses middle layers to extract features of the image instead of using a fully connected layer. The decoder is an LSTM network that produces the caption as a sequence of words. The model shows good results using three benchmark datasets using the METEOR and BLEU metric.

Image classification is another application of attention network. [128] Proposed a model that uses a two-level attention model for fine-grained image classification in deep convolutional neural networks. Fine-grained classification is detecting subordinate level categories under some basic categories. For example, classifying different bird types under the basic bird category. The difficulty of fine-grained classification is that discriminative features are on both foreground object and object parts, so the fine-grained classification models try to find image parts and extract discriminative features of them. This model is based on a simple intuition that for fine-grained classification we need to first find the object and after that find the discriminative parts of the object. In the object-level attention model, the model uses the raw image and removes noise to detect the object and do the basic classification. After that, part level attention model filters the object using mid-level CNN filters to detect the parts of the image and the model uses an SVM classifier to classify image parts. Using attention mechanism improved the performance of CNN fine-grained image classification since it can detect the important parts of the image that show discriminative features of it. The model validated on the subset of ILSVRC29112 dataset and CUB200 2011 dataset and it showed good performance under the weakest supervision condition.

[129] Proposed a novel attention-based machine translation model to translate image captions from English to German. Most machine translation models focus on the textual sentences of the source and target languages. This model considers additional information from the image to solve the problem of ambiguity in languages. It uses a convolutional neural network to extract the image features, and it adds the features of the image to the text features to enhance the performance of LSTM network that is used to generate the caption in the target language. Regional features of the image are used instead of general features. In sum, the best performance of the model shows 2% improvement in BLEU score and 2.3% enhancement in METEOR dataset compared to models that only consider text information.

Attention mechanism used in autonomous cars to simulate the driver's peripheral vision while driving and enhance object detection in autonomous cars. [130] Proposed a model to predict the steering wheel angel continuously using a visual attention model. A convolutional neural network was used to extract the encoded visual feature vector. These vectors have high-level information about the object's features and allow the attention model to selectively pay attention to specific parts of the image by selecting a subset of feature vectors. Three large datasets that contain more than 1,200,000 frames collected from Udacity, Comma.ai, and Hyundai Center of Excellence in Integrated Vehicle Safety Systems and Control (HCE) at Berkeley and they were used to train and validate this model. Datasets contain videos recorded by a camera that mounted behind a windshield of the vehicle. Besides, datasets contain a set of sensor measurement data such as vehicle velocity, accelerating, and steering angle. Videos were recorded during the highway driving in daytime and clear weather. The results of this model show that using visual attention doesn’t degrade the autonomous vehicle’s control accuracy compared to convolutional neural networks. Besides, using attention reduces the data complexity since it removes non-significant features.

All these works enhanced previous sequence learning models by considering the significance and weighted effect of each input sample on the model’s output. In this work, an encoder-decoder model with the attention mechanism was used to detect driver distraction level using driving data collected from car data buses and the performance of this model was compared with a simple encoder-decoder model with both scaled and un-scaled data.

Methodology

In the first experiment, [91] that it was called context only, three different distracted driving contexts were defined including ideal, adverse and double adverse context. Based on the experiment’s results, Vh can be estimated in different driving contexts using Vc, so Vc and Vh are correlated. After that, the Vc scenarios were extended from 3 distracted to 8 distracted and non-distracted ones and it was called binary output experiment. Four contexts of driving were defined including one ideal context, two adverse contexts and one double adverse context for both distracted and non-distracted driving. Also, 10 features of the Vc vector were chosen as inputs, and a single Vh feature chose as the output that shows if the driver is distracted or not. Driver distraction status could be detected in these scenarios using Vc vectors accurately [102]. Then, the first experiment was extended using the scenarios of binary output experiment and collected Vc and Vh data vectors in eight driving contexts. We tried to find the correlation between Vc and Vh with both the LSTM network [101] and MLP and we compared the results. In this chapter, we find a sequence of driver behavior using a sequence of driving data vectors.

We already showed that Vc and Vh are correlated and single Vc vector can be used to find a single Vh vector, so if we use a pipeline with the capacity of N and use a trained neural network N time, a sequence of Vh vectors can be estimated using a sequence of Vc vectors. Another solution is using a sequence to sequence model with the LSTM network. An LSTM network has two advantages that make it a more suitable choice for our system. First, it has memory, so it learns the correlation between Vc and Vh considering the effect of previous input samples on the current one. It considers both correlations between input samples and the correlation between input and output, so it is more accurate than the neural network that considers all inputs independent from each other which is not true in many real-life applications including this application. Second, it has forgiveness. It means if the driver has an abnormal behavior for a very short time and comes back to normal behavior, the model forgives the abnormal behavior and considers the driver’s behavior normal.

We validated the hypothesis with both the LSTM network and the Neural Network. An LSTM network was used to find a correlation between single Vh vector and related Vc vectors also a neural network was used to find the correlation between the Vh vector and the average of related Vc vectors. The LSTM network had a higher train and test accuracy besides, it has less overfitting problem and continued learning longer than the neural network. As a result, the sequence to sequence model with an LSTM network was chosen instead of the neural network to find a sequence of driver behavior. Based on previously achieved results [32] attention mechanism on the top of an LSTM model can enhance the training process and the model performance, so a sequence to sequence model with attention was used to enhance the performance of the model in the previous chapter.

The Encoder Decoder with Attention

Attention is one of the fundamental parts of cognition and intelligence in humans [131, 132]. It helps to reduce the amount of information processing and complexity [133, 134]. Attention can be defined as directing the human vision and hence the mind to a specific object rather than scanning the entire visual space [135, 136]. This is an essential human perception component that helps increase processing power while reducing demand on resources [137]. In the neural network area, attention is primarily used as a memory mechanism and it determines which part of the input sequence has more effect on the final output. The original attention mechanism made a significant enhancement in language translation applications compared to other deep learning methods such as encoder-decoder LSTM networks [138]. In the attention network, the encoder provides a context vector that is filtered specifically for each output in the output sequence.

Equation (33) shows the output of the encoder in the encoder-decoder without attention. In (33) h is the output vector of encoder that contains information of all samples in the input sequence. In the attention network, the encoder produces one vector for each output (34) shows the encoder output in the attention network. The decoder produces one output at a time and the model scores how well the encoded input matches the current output. Equation (35) shows the scoring formula to encode input i in step t. In (35) S_{t-1} shows the output from previous step and h_i is the result of encoding input x_i. In the next step, scores are normalized using (36) that is a “SoftMax” function. The context vector for each time step is calculated using (37) [139].

$$h = Encoder(x_1, x_2, x_3, \ldots, x_T, t) \quad (33)$$

$$(h_1, h_2, h_3, I, h_T) = Encoder(x_1, x_2, x_3, \ldots, x_T) \quad (34)$$

$$e_{ti} = a(S_{t-1}, h_i) \quad (35)$$

$$a_{ti} = \frac{exp(e_{ti})}{\sum_{j=0}^{T} exp(e_{tj})} \quad (36)$$

$$C_t = \sum_{j=0}^{T} a_{tj} * h_j \quad (37)$$

Figure 61 shows the encoder decoder with the attention mechanism. The bidirectional LSTM layer provides access to all samples of sequence in each step. The "SeqSelfAttention" layer of Keras was used in this model and this attention layer calculates the weighted effect of each sample in the input sequence for producing each sample of the output sequence. The length of input and output sequence are equal, and three different models were tried including three input and three output steps model, four input and four output steps model and five input and five output steps model. Variety combinations of batch size, activation function and the number of LSTM neurons were tried. Adam was chosen as the activation function, and 100 was chosen as the batch size. All models were trained with a range of LSTM neurons from 20 to 1000.

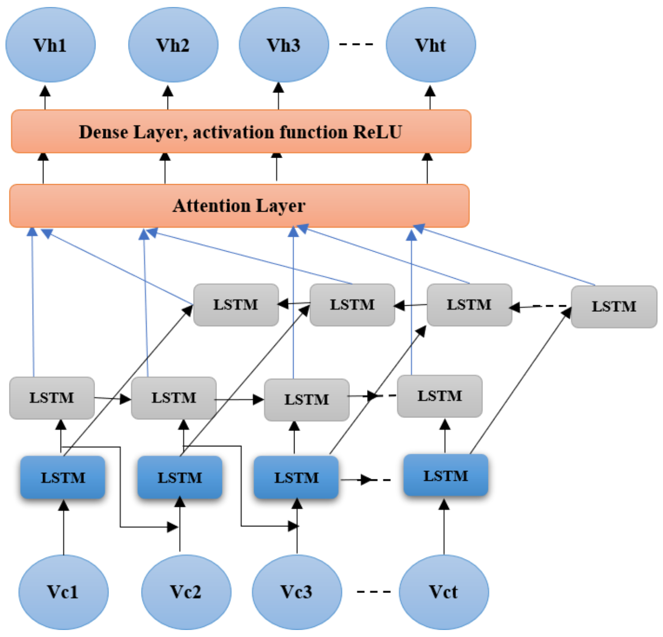


Figure 61. The Encoder Decoder with Attention

An encoder decoder model with attention was built to compare the performance of this model that has both memory and attention with the previous encoder decoder model in the previous chapter. We showed that adding attention to the model increases the intelligent level of the model and enhances the training process. The encoder in the encoder decoder model computes the hidden state considering two inputs including current input and the previous hidden state (38). In (38), h_t is the current state in step t and W_{hh} shows the weight between the previous hidden state and the current hidden state. W_{hx} is the weight between the current hidden state and the current input (X_t). The final hidden state computed by the encoder encapsulates the information of all input elements to help the decoder to predict the output sequence accurately.

Unlike the attention network that makes one context vector for each output, the encoder decoder uses one context vector for all outputs. The decoder computes the hidden state at step t using (39). The decoder only uses the previous hidden state to compute the current one.

$$h_t = f(W_{hh} * h_{t-1} + W_{hx} * X_t) \quad (38)$$

$$h_t = f(W_{hh} * h_{t-1}) \quad (39)$$

Three models with the encoder decoder were built including three input and three output steps model, four input and four output steps model and five input and five output steps model. Also, 100 was chosen as the batch size and Adam was chosen as the activation function. These models were trained with a range of LSTM neurons from 20 to 1000 to find the best combination. For both the attention network and simple encoder decoder model, 80% of the dataset was used for training and 20% for testing. Besides, models were trained with both scaled and unscaled data.

Results

Summary of the Encoder Decoder Results

In the previous chapter, three encoder decoder models were trained with a variety of LSTM neurons. Table 43 shows the summary of the achieved results with unscaled data that were discussed in the previous chapter in detail.

Table 43. The Encoder Decoder Model Results with Unscaled Data

LSTM Neurons	Sequence length	MAE train	MAE test
125	3	1.04	1.73
250	3	0.73	1.76
500	3	0.3	1.7
100	4	1.46	1.6
250	4	1.41	1.61
500	4	1.38	1.72
50	5	1.2	1.82
250	5	0.9	1.91
400	5	0.56	1.9

In summary, the four-step model reached the best performance which is 1.46 train and 1.6 test error. The performance of three steps and Five steps models have some similarities but the three steps model is more accurate. In all these three models adding more than around 150 LSTM neurons increased the model complexity and the model went toward overfitting. Table 44 shows the summary of encoder decoder models with scaled data that were discussed before in chapter 6. In sum, using four steps models resulted in the minimum train and test error with scaled data which is 0.15 train and 0.15 test error.

Encoder Decoder with Attention Results

Training the attention network was started with different combinations of hyperparameters. Different batch sizes, 50 hidden neurons, Adam optimizer and 50 epochs training were chosen as hyperparameters. The results show that the model with 100 as the batch size has the best performance. Figure 62 shows the error of the model with 50 and 100 as the batch size. Adadelta was chosen as the optimization function of this model since it shows better performance compared to Adam and Adagrad. Figure 63 compares the performance of the model with Adadelta and Adagrad optimization Functions.

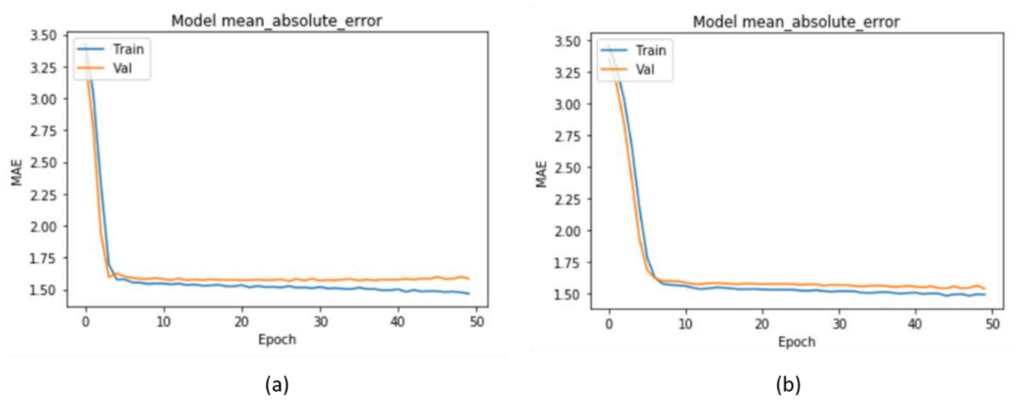


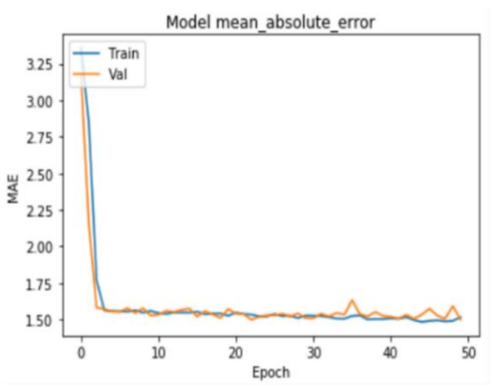
Figure 62. (a) Batch Size 50 for Encoder Decoder with Attention (b) Batch Size 100 for Encoder Decoder with Attention

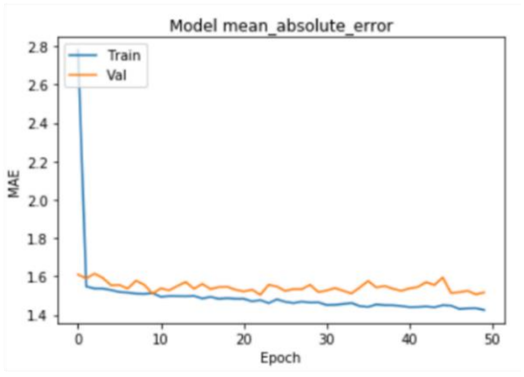
Three encoder-decoder with attention models were trained with a variety of LSTM neurons. Table 45 shows some of the achieved results with unscaled data. The three-step model reached the minimum error which is 1.5 train and test mean absolute error. The four-step and five-step attention models have almost the same performance. When the number of LSTM neurons was increased, the test error increased significantly for these two models and they went toward overfitting.

Table 44. The Encoder Decoder Model Results with Scaled Data

LSTM Neurons	Sequence Length	MAE train	MAE test
125	3	0.156	0.17
250	3	0.1	0.2
500	3	0.1373	0.18
125	4	0.1569	0.16
250	4	0.1655	0.16
500	4	0.151	0.15
125	5	0.158	0.17
250	5	0.16	0.16
500	5	0.161	0.16

The train and test error of three-step and five-step models show very significant change compared to the simple encoder decoder models. On the other hand, the four-step model had the minimum change compared to the simple encoder decode model. In all three models increasing the number of LSTM neurons degraded the model's performance. In sum, the effect of adding attention to the four-step model that already had good performance without attention wasn't significant, but adding attention reduced the overfitting problem of two other models notably. The three-step model with un-scaled data and 100 LSTM neurons showed the best performance. Figure 64 shows the mean absolute error of this model.





Adagrad

Figure 63. Adadelta and Adagrad Optimization Function Performance for Attention Model

Scaled data were used for training these models. Table 46 shows the best achieved results for three models with scaled data. The three-step model with 250 LSTM neurons reached the minimum error, and the five-step model had the weakest performance. The performance of the three-step and four-step model were very close to simple encoder decoder models with scaled data. In sum, the three-step model showed the minimum error and the best performance with both scaled and un-scaled data. Besides, using scaled data reduces the over fitting problem of the model, so the model continued learning for a longer period.

Table 45. The Results of Attention Network with Unscaled Data

LSTM Neurons	Sequence length	MAE train	MAE test
100	3	1.5	1.5
250	3	1.51	1.52
400	3	1.52	1.56
100	4	1.5	1.52
150	4	1.52	1.56
400	4	1.54	1.62
50	5	1.52	1.52
250	5	1.53	1.56
400	5	1.55	1.58

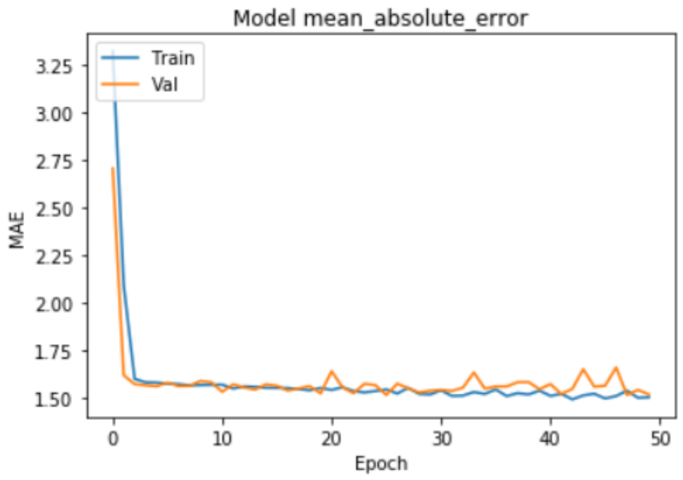


Figure 64. The Mean Absolute Error of Three-step Model with 100 LSTM Neurons and Unscaled Data

In sum, adding attention to the encoder decoder model enhanced the performance of the model and decreased the overfitting problem. Moreover, the best result of the attention model achieved with much less LSTM neurons. Three-step model and four-step model showed close performance in both simple and attention encoder decoder model. The five-step simple encoder decoder model and the five-step attention model had the weakest performance with both scaled and un-scaled data compared to two other models. The number of steps was increased, but the performance of the model degraded. Four-step encoder decoder model with attention was chosen as the most suitable architecture for the sequence to sequence model.

Table 46. The Results of Attention Network with Scaled Data

LSTM Neurons	Sequence length	MAE train	MAE test
150	3	0.1683	0.16
250	3	0.1641	0.16
100	4	0.1658	0.16
300	4	0.1656	0.16
150	5	0.1655	0.16
400	5	0.1646	0.17

Sequence to Sequence Model Compared to MLP Model and LSTM Model

MLP Results

As it is mentioned in Chapter 4, an MLP was built and trained with both scaled and unscaled data. Table 47 shows some of the best-achieved results with scaled data and Table 48 shows the achieved results with unscaled data. These results show the large different between train and test error that means the model has overfitting problem with both scaled and unscaled data and it doesn’t generalize well in most cases.

Table 47. MLP Scaled Data Results

layer	Neurons	Train MAE	Test MAE
Two	50	0.11	0.27
Two	150	0.082	0.23
Three	50	0.096	0.24
Three	150	0.056	0.32
Four	50	0.089	0.24
Four	150	0.012	0.2

Bidirectional LSTM with Attention

The results of driver behavior estimation using a bidirectional LSTM network with attention was discussed in chapter 5. Table 49 shows some results with unscaled data. For unscaled data, the best result achieved with 20 neurons and it is 0.85 training and 0.96. This model achieved less test error compared to the MLP model using a smaller number of hidden neurons. The best result of MLP achieved with four hidden layers and 300 neurons in each layer, so the MLP model compared to the LSTM attention model is less accurate and more computationally expensive. Besides, one layer of the LSTM model plus an attention layer had better performance compared to a large MLP model with four fully connected layers. In addition, the training process of the MLP model took around 400 to 600 epochs while the LSTM converged in around 100 to 200 epochs.

Table 48. MLP Unscaled Data Results

layer	Neurons	Train MAE	Test MAE
Two	150	0.54	1.41
Two	300	0.39	1.51
Three	150	0.33	1.48
Three	300	0.31	1.34
Four	150	0.4	1.46
Four	300	0.17	1.39

Table 49. Bidirectional LSTM Network with Attention Layer

LSTM Neurons	Train MAE	Test MAE
20	0.85	0.96
30	0.85	0.99
40	0.85	1.01
100	0.85	1
200	0.85	0.99
300	0.96	0.99

Table 50 shows the best achieved results with scaled data. The model with 40 neurons reached the minimum test error which is less than all cases in MLP network with scaled data except one case that is 4 hidden layers model with 150 neurons. In general, LSTM attention model with scaled data generalized better than MLP in all cases and achieved better performance with a smaller number of hidden neurons and smaller network.

Table 50. Bidirectional LSTM Network with Attention Layer Scaled Data

LSTM Neurons	Train MAE	Test MAE
30	0.23	0.24
40	0.22	0.22
60	0.22	0.23
100	0.24	0.25
150	0.22	0.23
200	0.23	0.23

Comparing Models

As it is mentioned earlier, to have a sequence of driver behavior data vectors a sequence to sequence model can be used instead of running a multi input single output model multiple times. The test error of the encoder decoder attention model with unscaled data is close to the MLP model with scaled data but the generalization of this model is much better than the MLP model and the network is smaller than the MLP model. Besides, it converged in around 50 epochs on average which is much faster than the MLP model that needs around 400 epochs training on average. Moreover, this model estimates multiple driver behavior vectors in one run. The error of this model with unscaled data is more than LSTM attention model but the generalization of this model is better and the number of input samples is much less than LSTM attention model since this model consider one driving data for each driver behavior vector, so it is computationally less expensive.

The encoder decoder attention with scaled data outperformed both the MLP model and the LSTM attention model. This model has memory and attention similar to the LSTM attention model. Besides, it considers the dependency between samples of the output sequence which is not possible if we run a multi input single output model multiple times to have a sequence of driver behavior vectors. Besides, this model is computationally less expensive than the LSTM attention model since it considers one input vector corresponding to each output, similar to the MLP model. The minimum test error of this model with scaled data is 0.06 less than the minimum test error of the LSTM attention model and 0.04 less than the minimum test error of the MLP model.

The encoder decoder attention model converges with less error and takes less time than two other models. The average number of epochs for this model with unscaled and scaled data was between 50 to 100 epochs which are half of the average epochs of the LSTM attention model and ¼ of MLP epochs. Besides, the model uses a smaller number of layers compared to MLP model and a smaller number of input samples compared to LSTM attention model making it computationally less expensive. It also estimates multiple driver behavior data vectors in each run.

Driver Distraction Level

The hypothesis that a sequence of driver behavior data vectors can be estimated using a sequence of driving data was validated using multiple deep learning models including MLP, LSTM, bidirectional LSTM, LSTM attention, encoder-decoder and encoder-decoder attention models. The encoder-decoder attention model is the most suitable and less computationally expensive one for this application. In the next step, we want to detect the level of driver distraction using the sequence of driver behavior data vectors and some predefined rules. We classified the sequence of driver behavior data vectors to five levels of driver distraction.

1. Mild: one step distraction in an ideal mode which is a mild distraction.
2. Moderate: one step distraction in adverse or double adverse conditions or two steps non-continuous distraction that at least the second step of distraction is in ideal mode.
3. Serious: two continuous steps distraction or two steps non-continuous distraction that at least the second step is in adverse or double adverse conditions.
4. Dangerous: a crash is likely to happen. We consider three steps of distraction out of four steps in adverse or double adverse modes or four steps of distraction in ideal mode without error as a dangerous level.
5. Catastrophic: a crash is happening. Four steps continuous distraction in adverse or double adverse modes or four steps of distraction in ideal mode with at least one error

Conclusion

Using machine learning methods to monitor driver behavior and detect the driver's inattention that can lead to distraction is an emerging solution to detect and reduce driver distraction. Different deep learning methods such as CNN, LSTM and RNN networks were used in several car safety applications. Some of these methods have memory, so they can extract and learn information in a sequence of data. There is some information in the sequence of driving data that can’t be impeded from processing them manually. Driving data samples are not independent of each other, so methods that have memory and attention such as recurrent models are a better choice for higher intelligence and hence more reliable car safety applications.

These methods utilize different mechanisms to perceive the dependency between samples and extract the latent information in the sequence. The MLP network which is a simple memoryless deep neural network was chosen as the baseline for the model. Then multiple models with memory or combination of memory and attention were trained including LSTM, Bidirectional LSTM, stacked LSTM, LSTM attention. These models outperform the MLP model with both scaled and unscaled data. These models trained multiple times faster than the MLP model and achieved better performance with a smaller network and a smaller number of training epochs.

In order to have a sequence of driving data there are two options: run a multi input single output model multiple times and using a sequence to sequence model. Multiple encoder-decoder models and multiple encoder decoder attention models were trained with both scaled and unscaled data to have a sequence of driver behavior data vectors. Encoder decoder models outperform the MLP model with both scaled and unscaled data. Moreover, encoder decoder model with attention outperformed the simple encoder decoder model. Besides, the encoder decoder attention model outperformed LSTM attention model with scaled data. Encoder decoder attention model trained at least two times faster than the LSTM attention model and four times faster than the MLP model.

Besides, in each run, it estimates multiple driving data vectors and it had the best generalization and minimum difference between train and test error. These results show that this would be a viable and scalable option for deep neural network models that work in real-life complex driving contexts without the need to use intrusive devices. It also provides an objective measurement of the added advantages of using attention networks to reliably detect driver behavior.

The encoder-decoder attention model is the most suitable and less computationally expensive one for this application. In the next step, we detect the level of driver distraction using the sequence of driver behavior data vectors and some defined rules. We classified the sequence of driver behavior data vectors to five levels of driver distraction including *Mild, Moderate, Serious, Dangerous* and *Catastrophic*. As a future job, the model can be extended using more complex driving scenarios and considering different types of distracting tasks to cover more possible driving distracting behavior.

REFERENCES

[1] N. Xi, T.-J. Tarn, and A. Bejczy, "Intelligent planning and control for multirobot coordination: An event-based approach," *IEEE Transactions on Robotics and Automation,* vol. 12, pp. 439-453, 1996.

[2] J. D. Lee, "Dynamics of Driver Distraction: The process of engaging and disengaging," *association of the advancement of automotive medicine,* vol. 58, pp. 24-32, 2014.

[3] A. Gaffar and S. Monjezi Kouchak, "Using Artificial Intelligence to Automatically Customize Modern Car Infotainment Systems," in *Proceedings on the International Conference on Artificial Intelligence (ICAI)*, 2016, p. 151.

[4] J. Heinzman and A. Zelinsky, "Automotive Safety Solutions through Technology and Human-Factors Innovation," *Springer Tracts in Advanced Robotics book series,* vol. 79, p. 11, 2014.

[5] M. Hafner, D. Cunningham, L. Caminiti, and D. D.Vecchio, "Cooperative Collision Avoidance at Intersections: Algorithms and Experiments," *IEEE Transactions on Intelligent Transportation Systems,* vol. 14, p. 14, 2013.

[6] D. Geronimo, A. M. Lopez, A. D.Sappa, and T. Graf, "Survey of Pedestrian Detection for Advanced Driver Assistance Systems," *IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE,* vol. 32, pp. 1239-1258, 2010.

[7] B. Pfleging, T. Doring, and I. Alvarez, "Auto NUI : 2nd workshop on Automotive Natural User interface " *AutomotiveUI '12,* pp. 37-38, 2012.

[8] A. Gaffar and S. Monjezi Kouchak, "Minimalist Design: An Optimized Solution for Intelligent Interactive Infotainment Systems," *IEEE IntelliSys, the International Conference on Intelligent Systems and Artificial Intelligence. London,* September 7th – 8th 2017.

[9] T. Sheridan and R. Parasuraman, "Human-Automation Interaction. Reviews of Human Factors and Ergonomics," vol. 1, p. 41, 2015.

[10] A. Gaffar and S. Monjezi Kouchak, " Quantitative Driving Safety Assessment Using Interaction Design Benchmarking," *to IEEE Advanced and Trusted Computing (ATC 2017), August 4-8, 2017, San Francisco Bay Area, USA,* 2017.

[11] NHTSA. (2016). *Distracted driving*. Available: www.distraction.gov/stats-research-laws/facts-and-statistics.html,

[12] J. Anderson, *Cognitive Psychology and its Implications,* Fifth ed.: Worth Publishers, 2000.

[13] J. Hetch. (2013, August 20th). *The Last Text You'll Ever Send.Why new driving safety technologies are really dangerous.* Available:

http://www.slate.com/articles/health_and_science/new_scientist/2013/07/voice_and_text_while_driving_research_shows_it_s_all_dangerously_distracting.html

[14] R. s. f. sheet, "The Royal Society for the Prevention of Accidents," 2017.

[15] " National Center for Statistics and Analysis Distracted driving in fatal crashes, 2017," 2019.

[16] N. s. Council, "Understanding the distracted brain," *White paper*, 2012.

[17] N.-A. R. S. T. P. Paper. (March 10th). *Driver Distraction. NRMA-ACT Road Safety Trust Position Paper*. Available: http://acrs.org.au/files/roadsafetytrust/1423448741.pdf

[18] DMV. (May 1st 2018). *Distracted driving*. Available: https://www.dmv.org/distracted-driving.php

[19] B. Darrow. (2016, Feburary 10th). *Distracted Driving Is Now an Epidemic in the U.S.* Available: http://fortune.com/2016/09/14/distracted-driving-epidemic

[20] Neale, T. Dingus, S. Klauer, J. Sudweeks, and M. Goodman, "An Overview of the 100-car Naturalistic Study and Findings," *National Highway Traffic Safety Administration United States*.

[21] T. Johanson. (2018, May 2nd). *Distraction Tops Drivers' List of Growing Dangers on the Road*. Available: https://newsroom.aaa.com/2018/03/distraction-tops-drivers-list-growing-dangers-road/

[22] N. s. N. C. f. S. a. Analysis. (2018, October 20th). *Driver Distraction*. Available: https://crashstats.nhtsa.dot.gov/Api/Public/ViewPublication/812517

[23] J. A. Lozano, "Witness: Truck driver in Texas crash that killed 13 was texting," ed: Associated Press, 2017.

[24] O. o. t. S. o. M. Vehicles, "Addressing the Problem of Distracted Driving and its Impacts to Road Safety " *Ministry of Public Safety and Solicitor General Office of the Superintendent of Motor Vehicles*

[25] *Top 10 Causes of Distracted Driving—and What They All Have in Common*. Available: https://safestart.com/news/top-10-causes-distracted-driving-and-what-they-all-have-common/

[26] B. Darrow. (2016). *Distracted driving is now an epidemic in the U.S*. Available: https://fortune.com/2016/09/14/distracted-driving-epidemic/

[27] EndDD. (2018, December 5th). *Distracted Driving Facts*. Available: https://www.enddd.org/the-facts-about-distracted-driving/?gclid=Cj0KCQiAwc7jBRD8ARIsAKSUBHJfLaE0XR4_qmG1UzvnJAkFhdJm2yTM-j9aSqsnd1942dN2vUYWTD8aAhAlEALw_wcB

[28] N. H. T. S. Administration, "NHTSA studies vehicle safety and driving behavior to reduce vehicle crashes," 2009.

[29] S. Kaplan, M. Guvensan, A. Yavuz, and Y. Karalurt, "Driver Behavior Analysis for Safe Driving: A Survey," *IEEE TRANSACTIONS ON INTELLIGENT TRANSPORTATION SYSTEMS,*, vol. 16, pp. 3017-3032, December 2015.

[30] U. o. Utah. (2006, May 1st). *Drivers on Cell Phones Are as Bad as Drunks*. Available: https://archive.unews.utah.edu/news_releases/drivers-on-cell-phones-are-as-bad-as-drunks/

[31] "Why so many people text and drive, knowing dangers," ed. CBS News.

[32] S. Monjezi Kouchak and A. Gaffar, "Using Bidirectional Long Short Term Memory with Attention Layer to Estimate Driver Behavior," *IEEE International Conference on Machine learning and Applications,* December 16-19, 2019 2019.

[33] T. Hirayama, K. Mase, and K. Takeda, "Analysis of Temporal Relationships between Eye Gaze and Peripheral Vehicle Behavior for Detecting Driver Distraction," *Hindawi Publishing Corporation International Journal of Vehicular Technology,* vol. 2013, pp. 1-8, 2013.

[34] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature,* vol. 521, pp. 436-444, 2015.

[35] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*: MIT press, 2016.

[36] A. Gibson and J. Patterson, *Deep learning*. OREILLY, 2017.

[37] L. Zhang, F. Liu, and J. Tang, "Real-Time System for Driver Fatigue Detection by RGB-D Camera," *ACM Transactions on Intelligent Systems and Technology,* vol. 6, pp. 1-17, 2015.

[38] A. Fernández, R. Usamentiaga, J. Carús, and R. Casado, "Driver distraction using visual based sensors and algorithms, Sensors," *sensors,* vol. 16, 2016.

[39] S. Baulk, L. Reyner, and J. Horne, "Driver Sleepiness—Evaluation of Reaction Time Measurement as a Secondary Task," *SLEEP,* vol. 24, pp. 695-698.

[40] K. Murthy and Z. Ahmed Khan, "Different Techniques Quantify the Driver Alertness. ," *World Applied Sciences Journal* vol. 22, pp. 1094-1098, 2013.

[41] A. Doshi and M. M.Trivedi, "Head and eye gaze dynamics during visual attention shifts in complex environments," *Journal of Vision February 2012,* vol. 12, pp. 1-16, 2012.

[42] F. Tango and M. Botta, "Real-Time Detection System of Driver Distraction Using Machine Learning," *IEEE TRANSACTIONS ON INTELLIGENT TRANSPORTATION SYSTEMS,* vol. 14, 2013.

[43] P. Boyraz and M. K. Acar, David, "Signal Modelling and Hidden Markov Models for Driving Manoeuvre Recognition and Driver Fault Diagnosis in an urban road scenario," *2007 IEEE Intelligent Vehicles Symposium Istanbul, Turkey, June 13-15,* pp. 987-992, 2007.

[44] A. Sathyanarayana, P. Boyraz, and J. Hansen, "Driver behavior analysis and route recognition by hidden Markov model.," *IEEE International Conference on Vehicular Electronics and Safety Columbus OH, USA. September 22-24.*, pp. 276-281, 2008

[45] A. Sathyanarayana, P. Boyraz, and J. Hansen, " Model-Based Analysis and Classification of Driver Distraction Under Secondary Tasks," *IEEE TRANSACTIONS ON INTELLIGENT TRANSPORTATION SYSTEMS,* vol. 11, pp. 276-281, 2010.

[46] P. Tchankue, J. Wesson, and D. Vogts, "Using machine learning to predict the driving context whilst driving," *SAICSIT 13Proceedings of South African Institute for computer Scientists and Information Technology Conference,* pp. 47-55, 2013.

[47] T. Kumagai and M. Akamatsu, "Prediction of human driving behavior using dynamic Bayesian networks," *IEICE TRANS. INF &SYST,* vol. E89_D, pp. 875-879, 2006.

[48] C. Craye and F. Karray, "Driver Distraction Detection and Recognition Using RGB-D Sensor," *Cornell university Library, computer vision and pattern recognition,* pp. 1-11, 2015.

[49] F. Vicente, Z. Huang, X. Xiong, F. De la Torre, W. Zhang, and D. Levi, "Driver Gaze Tracking and Eyes Off the Road Detection System," *IEEE TRANSACTIONS ON INTELLIGENT TRANSPORTATION SYSTEMS,* vol. 16, pp. 2014-2027, 2014.

[50] K. Belleghem, M. Denecker, and D. Schreye, "ON THE RELATION BETWEEN SITUATION CALCULUS AND EVENT CALCULUS " *THE JOURNAL OF LOGIC PROGRAMMING Elsevier Science Inc.*, pp. 3-37, 1997.

[51] J. Allen, "Towards a General Theory of Action and Time," *Artificial Intelligence,* vol. 23, pp. 123-154, 1984.

[52] V. Lifschitz, "FORMAL THEORIES OF ACTION," *The Frame Problem in Artificial Intelligence Proceedings of the 1987 Workshop,* pp. 35-57, 1987.

[53] H. Levesque, F. Pirri, and R. Reiter, "Foundations for the Situation Calculus," *Computer and Information Science,* vol. 3, 1998.

[54] R. J. Brachman and H. J. Levesque. (2005, January). *Knowledge representation and reasoning*. Available: https://www.cs.toronto.edu/~hector/PublicKRSlides.pdf

[55] C. Bishop, *pattern recognition and machine learning*: Springer, 2006.

[56] A. Gaffar, "Studies on Pattern Dissemination and Reuse to Support Interaction Design," PhD, Concordia, Library and Archieve Canada Publish Heritage Branch, 2005.

[57] C. Alexander, *The Timeless Way of Building*: Oxford University Press, 1979.

[58] E. T. Jaynes, *Probability theory the logic of science*: Cambridge university Press, 2003.

[59] A. R. Webb, *Statistical Pattern recognition*, second ed. UK: John Wiley & Sons Ltd, 2002.

[60] A. Rozza, G. Lombardi, E. Casiraghi, and P. Campadelli, "Novel Fisher discriminant classifiers," *pattern recognition,* vol. 45, pp. 3725-3737, 2012.

[61] A. Zurr, E. Leno, N. Walker, A. Saveliev, and G. Smith, "Limitations of Linear Regression Applied on Ecological Data," in *Mixed Effects Models and Extensions in Ecology with R*, ed: Springer, 2009.

[62] J. Zhang and X. W. Wang, "The application of feed-forward neural network for the X-ray image fusion," *Journal of Physics: Conference Series 312,* pp. 1-6, 2011.

[63] S. Elanayar and Y. Shin, "Radial Basis Function Neural Network for Approximation and Estimation of Nonlinear Stochastic Dynamic Systems " *IEEE TRANSACTIONS ON NEURAL NETWORKS,* vol. 5, pp. 594-603, 1994.

[64] T. Kohonen, "Self-organized formation of topologically correct feature maps," *Biological Cybernetics Journal,* vol. 43, 1982.

[65] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural networks,* vol. 61, pp. 85-118, 2015.

[66] W. Muldera, S. Behhardb, and M.-F. Monesa, "A survey on the application of recurrent neural networks to statistical language modeling," *Elsevier,* vol. 30, pp. 61-98, 2015.

[67] S. J.Russell and P. Norving, *Artificial Intelligence A Modern Approach*. USA: Prentice Hall, 2010.

[68] D. O. T. N. H. T. S. Administration, "Visual-Manual NHTSA Driver Distraction Guidelines for In-Vehicle Electronic Devices," 2012.

[69] H. Zhang and L. N. Wei, "speech recognition interface design for in-vehicle system," *AutumoviveUI , ACM 978* pp. 29-33, 2010.

[70] NHTSA. *Automated Vehicles for Safety*. Available: https://www.nhtsa.gov/technology-innovation/automated-vehicles-safety

[71] S. Bagloee, M. Tavana, M. Asadi, and T. Oliver, " Autonomous vehicles: challenges, opportunities, and future implications for transportation policies," *Journal of Modern Transportation,* vol. 24, p. 18, 2016.

[72] C. Burges, T. Shaked, E. Renshaw, A. Lazier, M. Deeds, N. Hamilton*, et al.*, "Learning to Rank using Gradient Descent," *IEEE 22 nd International Conference on Machine Learning, Bonn, Germany*, 2005.

[73] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," *the 13th International Conference on Artificial Intelligence and Statistics (AISTATS),* vol. 9, pp. 249-256, 2010.

[74] K. He, X. Zhang, S. Ren, and J. Sun, "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification," *arXiv:1502.01852v1 [cs.CV] 6 Feb 2015,* pp. 1-11, 2015.

[75] B. Xu, N. Wang, T. Chen, and M. Li, "Empirical Evaluation of Rectified Activations in Convolution Network," *arXiv:1505.00853v2 [cs.LG] 27 Nov 2015,* pp. 1-5, 2015.

[76] L. N.Smith, "Cyclical Learning Rates for Training Neural Networks," *arXiv:1506.01186v6,* 2017.

[77] M. Zeiler, "ADADELTA: AN ADAPTIVE LEARNING RATE METHOD," *arXiv:1212.5701v1 [cs.LG] 22 Dec 2012,* pp. 1-6, 2012.

[78] D. Kingma and J. Ba, "ADAM: A METHOD FOR STOCHASTIC OPTIMIZATION," *arXiv:1412.6980v9 [cs.LG] 30 Jan 2017*, 2017.

[79] J. Li, X. Mei, D. Prokhorov, and D. Tao, "Deep Neural Network for Structural Prediction and Lane Detection in Traffic Scene," *IEEE Transactions on Neural Networks and Learning Systems* vol. 28, pp. 690-703, 2017.

[80] I.-H. Choi, S. Kyung Hong, and Y.-G. Kim, "Real-time categorization of driver's gaze zone using the deep learning techniques," *2016 International Conference on Big Data and Smart Computing (BigComp),* pp. 143-148, 2016.

[81] A. Koesdwiady, S. M. Bedawi, C. Ou, and F. Karray, "End-to-End Deep Learning for Driver Distraction Recognition," *Springer International Publishing AG 2017,* pp. 11-18, 2017.

[82] X. Chen, K. Kundu, Z. Zhang, H. Ma, S. Fidler, and R. Urtasun, "Monocular 3D Object Detection for Autonomous Driving," *computer vision foundation,* pp. 2147-2156, 2016.

[83] C. Chen, A. Seff, A. Kornhauser, and J. Xiao, "DeepDriving: Learning Affordance for Direct Perception in Autonomous Driving," *15th IEEE International Conference on Computer Vision (ICCV2015),* pp. 2722-2730, 2015.

[84] Y. Tian, K. Pei, S. Jana, and B. Ray, "DeepTest: Automated Testing of Deep-Neural-Network-driven Autonomous Cars," *ICSE '18 40th International Conference on Software Engineering*, pp. 1-12, 2018.

[85] NHTSA. (Feburary 5th). *Overview of the national highway traffic Safety administration, driver distraction program*. Available: https://www.nhtsa.gov/sites/nhtsa.dot.gov/files/811299.pdf

[86] (2018, January 7th). *Road safety Data (UK)*, . Available: https://data.gov.uk/dataset/road-accidents-safety-data

[87] NHTSA. (2009, March 9th). *An Examination of Driver Distraction as Recorded in NHTSA Databases*. Available: https://crashstats.nhtsa.dot.gov/Api/Public/ViewPublication/811216

[88] NHTSA. (2016, January 15th). *Fatality Analysis Reporting System (FARS) - Online Query Tool*. Available: https://catalog.data.gov/dataset/fatality-analysis-reporting-system-fars-ftp-raw-data

[89] NHTSA. (2015, December 7th). *2015 Motor Vehicle Crashes: Overview, NHTSA's National Center for Statistics and Analysis*. Available: https://crashstats.nhtsa.dot.gov/Api/Public/ViewPublication/812318

[90] I. Idris, *Python Data Analysis Cookbook*: Packt, 2016.

[91] S. Monjezi Kouchak and A. Gaffar, "Non-Intrusive Distraction Pattern Detection Using Behavior Triangulation Method " presented at the 4th annual conference on computational science and computational intelligence (CSCI - ISAI), 2017.

[92] D. Erhan, Y. Bengio, A. Courville, P.-A. Manzagol, and V. Pascal, "Why Does Unsupervised Pre-training Help Deep Learning?," *Journal of Machine Learning Research 11*, vol. 11, pp. 625-660, 2010.

[93] S. Otte, M. Liwicki, and D. Krechel, "Investigating Long Short-Term Memory Networks for Various Pattern Recognition Problems," *MLDM 2014*, pp. 484-497, 2014.

[94] R. Grosse. (June 18th). *Lecture 15: Exploding and Vanishing Gradients*. Available: http://www.cs.toronto.edu/~rgrosse/courses/csc321_2017/readings/L15%20Exploding%20and%20Vanishing%20Gradients.pdf

[95] C. L. Zachary and B. John, "A Critical Review of Recurrent Neural Networks for Sequence Learning," *arXiv:1506.00019v4 [cs.LG] 17 Oct 2015*, pp. 1-38, 2015.

[96] A. Graves, A.-r. Mohamed, and G. Hinton, "Speech Recognition with Deep Recurrent Neural Networks," *arXiv:1303.5778*, pp. 1-5, 2013.

[97] D. Jurafsky and J. Martin, *Speech and language processing*, Second ed.: Pearson Education UK, 2013.

[98] G. Fink, *Markov Model for Pattern Recognition*, Second Edition ed.: Springer.

[99] G. Chen, "A Gentle Tutorial of Recurrent Neural Network with Error Backpropagation," *arXiv:1610.02583v3 [cs.LG] 14 Jan 2018*, pp. 1-10, 2018.

[100] (October 4th). *SemEval-2013: Sentiment Analysis in Twitter*. Available: https://github.com/peace195/context-based-sentiment-analysis

[101] S. Monjezi Kouchak and A. Gaffar, "Estimating the Driver Status Using Long Short Term Memory," *International Cross-Domain Conference (CD-MAKE)* 2019.

[102] S. Monjezi Kouchak and A. Gaffar, "Distraction Detection Using Deep Neural Network," *The Fifth International Conference on Machine Learning, Optimization, and Data Science,* 2019.

[103] K. Kaplan, M. Guvensan, A. Yavuz, and Y. Karalurt, "Driver Behavior Analysis for Safe Driving: A Survey," *IEEE Transactions on Intelligent Transportation Systems,* vol. 6, 2015.

[104] R. Pascanu, T. Mikolov, and Y. Bengio, "On the difficulty of training Recurrent Neural Networks," *arXiv:1211.5063v2 [cs.LG] 16 Feb 2013,* pp. 1-12, 2013.

[105] R. Rescorla, "The Computational Theory of Mind," *Stanford Encyclopedia of Philosophy,* 2015.

[106] D. Norman and D. Rumelhart, *Explorations in Cognition*: W. H. Freeman & Company, 1975.

[107] M. Bickhart and L. Terveen, *Foundational Issues in Artificial Intelligence and Cognitive Science*: Elsevier, 1995.

[108] N. Nilson, *Principles of Artificial Intelligence.* : Morgan Kaufmann, 1980.

[109] S. Frintrop, E. Rome, and H. Christenson, "Computational visual attention systems and their cognitive foundations: A survey," *ACM Transactions on Applied Perception (TAP),* vol. 7, 2010.

[110] M. Wöllmer, C. Blaschke, T. Schindl, B. Schuller, B. Färber, S. Mayer, *et al.*, "Online Driver Distraction Detection Using Long Short-Term Memory," *IEEE TRANSACTIONS ON INTELLIGENT TRANSPORTATION SYSTEMS,* vol. 2, p. 8, 2011.

[111] O. Olabiyi, E. Martinson, V. Chintalapudi, and R. Guo, "Driver Action Prediction Using Deep (Bidirectional) Recurrent Neural Network," *arXiv:1706.02257v1,* p. 7, 2017.

[112] Y. Liu, Y. Lin, S. Wu, C. Chuang, and C. Lin, "Brain Dynamics in Predicting Driving Fatigue Using a Recurrent Self-Evolving Fuzzy Neural Network," *IEEE transactions on neural networks and learning systems,* vol. 27, p. 13, 2017.

[113] S. Chen, S. Zhang, J. Shang, B. Chen, and N. Zheng, "Brain-inspired Cognitive Model with Attention for Self-Driving Cars," *IEEE transaction on cognitive and developmental systems,* 2017.

[114] M. Wollmer, Z. Zhang, F. Weninger, B. Schuller, and G. Rigoll, "Feature enhancement by bidirectional LSTM networks for conversational speech recognition in highly non-stationary noise," *2013 IEEE International Conference on Acoustics, Speech and Signal Processing,* pp. 6822-6826, 2013.

[115] V. Badrinarayanan, A. Kendall, and R. Cipolla, "SegNet: A deep convolutional encoder-decoder architecture for image segmentation," *IEEE Transactions on Pattern Analysis and Machine Intelligence,* vol. 39, p. 14, 2017.

[116] O. Vinyals, A. Toshev, S. Bengio, and D. Erhan, "Show and tell: a neural image caption generator," *IEEE Conference on Computer Vision and Pattern Recognition (CVPR),* 2015.

[117] K. Cho, A. Courville, and Y. Bengio, "Describing multimedia content using attention-based encoder-decoder networks," *IEEE Transactions on Multimedia, ,* vol. 17, p. 12, 2015.

[118] K. Cho, B. van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk*, et al.*, "Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation," *arXiv:1406.1078v3,* 2014.

[119] W. Zaremba and I. Sutskever, "Learning to Execute," *arXiv:1410.4615v3* 2014.

[120] I. Sutskever, O. Vinyals, and Q. Le, "Sequence to sequence learning with neural networks," *Advances in Neural Information Processing Systems 27 (NIPS 2014),* 2014.

[121] S. Venugopalan, M. Rohrbach, J. Donahue, R. Mooney, T. Darrell, and K. Saenko, "Sequence to sequence – video to text," *IEEE International Conference on Computer Vision (ICCV),* p. 9, 2015.

[122] H. Liu, T. Taniguchi, Y. Tanaka, K. Takenak, and T. Bando, " Visualization of Driving Behavior Based on Hidden Feature Extraction by Using Deep Learning," *IEEE Transactions on Intelligent Transportation Systems,,* vol. 18, p. 13, 2017.

[123] D. Beyer, *The future of machine intelligence*: OREILLY, 2016.

[124] L. Thang. (2017, October 4th). *Building Your Own Neural Machine Translation System in TensorFlow*. Available: https://ai.googleblog.com/2017/07/building-your-own-neural-machine.html

[125] F. Wang, M. Jiang, C. Qian, S. Yang, C. Li, H. Zhang*, et al.*, "Residual Attention Network for Image Classification," *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR),* p. 9, 2017.

[126] M. Johnson, M. Schuster, Q. V. Le, M. Krikun, Y. Wu, Z. Chen*, et al.*, "Google's Multilingual Neural Machine Translation System: Enabling Zero-Shot Translation," *Transactions of the Association for Computational Linguistics,,* vol. 5, p. 13, 2017.

[127] K. Xu, J. Bay, R. Kirosy, K. Cho, A. Courville, R. Salakhutdinovy*, et al.*, "Show, Attend and Tell: Neural Image Caption Generation with Visual Attention," *32 nd International Conference on Machine Learning,* 2015.

[128] T. Xiao, Y. Xu, K. Yang, J. Zhang, Y. Peng, and Z. Zhang, "The Application of Two-level Attention Models in Deep Convolutional Neural Network for Fine-grained Image

Classification," *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR),* p. 9, 2015.

[129] P. Huang, F. Liu, S. Shiang, J. Oh, and C. Dyer, " Attention-based Multimodal Neural Machine Translation,," *Proceedings of the First Conference on Machine Translation,* vol. 2, p. 17, 2016.

[130] J. Kim and J. Canny, " Interpretable Learning for Self-Driving Cars by Visualizing Causal Attention," *IEEE International Conference on Computer Vision (ICCV),* 2017.

[131] A. Gaffar, E. M. Darwish, and A. Tridane, "Structuring heterogeneous big data for scalability and accuracy," *International Journal of Digital Information and Wireless Communications,* vol. 4, p. 14, 2014.

[132] A. Gaffar, H. Javahery, A. Seffah, and D. Sinnig, "A pattern framework for eliciting and delivering UCD knowledge and practices," *Proceedings of the Tenth International Conference on Human-Computer Interaction,* 2003.

[133] A. Gaffar, "Enumerating mobile enterprise complexity 21 complexity factors to enhance the design process," *Proceedings of the 2009 Conference of the Center for Advanced Studies on Collaborative Research,* p. 13, 2009.

[134] A. Gaffar, "The 7C's: An Iterative Process for Generating Pattern Components," *11th International Conference on Human-Computer Interaction,* 2005.

[135] R. Solso, O. MacLin, and M. MacLin, *Cognitive psychology*, 8 ed.: PEARSON, 2007.

[136] J. Bermudez, *Cognitive Science: An Introduction to the Science of the Mind*, 2 ed., 2014.

[137] B. Garrett and G. Hough, *Brain & behavior: an introduction to behavioral neuroscience,* 5 ed.: SAGE.

[138] D. Bahdanau, K. Cho, and Y. Bengio, "Neural Machine Translation by Jointly Learning to Align and Translate," *ICLR,* 2015.

[139] M. Luong, H. Pham, and C. Manning, "Effective Approaches to Attention-based Neural Machine Translation," *Empirical Methods in Natural Language Processing,* p. 10, 2015.