’’’

Machine Learning Design

Author: Tyler Travis, Supervisor: Dr. Huajie Zhang
Undergraduate Program of Software Engineering
University of New Brunswick, Fredericton, NB, Canada
Email: *ttravis@unb.ca

***Abstract*—In compliance with requirements for the University of New Brunswick's SWE4913 curriculum. The purpose of this independent research project is to build knowledge on basic concepts of machine learning models.**

***Index Terms*—Machine Learning, Machine Learning Independent Study, Machine Learning Architectures, Housing Prices, Autonomous Vehicles**

I. Introduction

After looking at a single-layer perceptron neural network in my secondary education, a simple quadratic polynomial fitter made myself question the capabilities of neural networks in computing. This opportunity to expand knowledge on such a paramount part of the technological revolution is essential to my growth as a software engineer.

This paper is cohesive with a public GitHub repository with any relevant code/data to the research.

The goal of this paper is to have reference to the key elements of Machine Learning, as a guided description of my discoveries.

Topics will be drawn from academia, novels, and public GitHub repositories. Literature on Machine Learning will be useful to grasp concepts of the structural architecture and procedure of an algorithm. Albeit understanding is the most important factor, I also plan on looking at examples of machine learning code with the goal deciphering what is going on under the hood.

II. Notes of The Hundred Page Machine Learning Book

This book was chosen as it is a very popular read, and is described as a good introduction to machine learning. Its concise nature allows a general understanding, and was selected to be my first read in my deeper endeavour of ML.

Machine learning algorithms are often described as "supervised", or "unsupervised". There are also "semi-supervised" and "reinforcement" machine learning algorithms.

- **Supervised Models** uses labeled sets of data to train an algorithm, and by iterating can predict more accurately. They often take feature vectors in, and output information accordingly. It is the most commonly used learning method. The data is supervised to (input, output).
- **Unsupervised Models** are in charge of identifying patterns of unlabeled data. There is also a feature vector given into this learning alternative.
- **Semi-Supervised Models** is a mix of labelled and unlabelled data. The end goal is to build a strong algorithm.
- **Reinforcement Learning** is interpreted as a state, that can execute actions at every state. Depending on actions, a reward system is engaged accordingly. This helps a computer algorithm decipher policy. This is measured through excepted average reward.

Classification and Regression are often mixed terms. In our terms, **classification** entails automatically assigning labels to unlabeled data, where the then-labeled data can help develop a model. For example spam detection. For **regression** is the pursuit of predicting a label based off an unlabeled example. For example, judging a house price, based off location, number of bedrooms, area etc.

Machine Learning algorithms can be manually developed, however the industry standard is known to leverage libraries to source their work.

A **Neural Network** is a nested function,

$$y = f_{NN}(x) = f_3(f_2(f_1(x))) \tag{1}$$

hence the common statement of "layers" within a neural network. The vector functions follow the form,

$$f_l(z) = g_l(W_l z + b_l) \tag{2}$$

where l is the the layer index (spans from 1 to any number of layers), and g_l is the activation function. The parameters W_l is a matrix, b_l is a vector.

III. Case Study: House Prices

The first study that will be looked at through my pursuit of ML is its application to predicting housing prices. John Ade Ojo's article: "Predicting House Prices with Machine Learning" is a very organized, thorough explanation of how to identify patterns in the housing market, and predict a price dependant on it.

The pursuit of this machine learning implementation is to use a publicly available data-set of housing prices along with different aspects of the house (which will be used as our feature vector). The available data set will be used to train the model, and for that reason can be described as a supervised model.

Ojo decided to build his model in Python, taking advantage of its various assortments of libraries that are known to be useful in the language. His tools include,

- **Pandas** to structure his data properly. If data-science has taught me anything, data-wrangling is 75 percentage of the work. Pandas is very useful for data-wrangling, in the various ways it can be cleaned (NaNs, incomplete information disregarded).

- **Scikit-Learn** is one of the leaders in machine learning classification, regression, and clustering algorithms. Has close ties to Pandas, Numpy, and Scipy.

- **Numpy** is Python's leading tool for any sophisticated mathematical endeavours. It has its own array data structure that can speed up rigorous lengthy training, and is worth the difference. If that is not enough, it has basically all mathematical capabilities one could ask for, including matrix algebra.

- **Seaborn** is probably the least known tool out of those that were used. This is similar to MatPlotLib, and is used for data visualization.

Something interesting about this project is its very inconsistent flow. The order may overlap and there are occasions where going backwards is required, however the order typically is:

- **Data Ingestion**, the process of extracting the data that will be used from some source for further steps.

- **Data Cleaning**, also is commonly called data wrangling, and is dependant on data ingestion. Data wrangling can mean merging two different data sources, clearing gaps in data, or removing duplicate entries.

- **Exploratory Data Analysis** is when things start to get interesting. This is mixed with the tool of data visualization, and is often used as an indicator of the quality of the data you are using.

  The **Correlation Matrix** is matrix with the width equal to the height, of which each element is one feature. By this format, a output is shown of the Pearson's calculation for element $X_{i,j}$ (also known as a correlation calculation factor). This shows how strong of a correlation there is of two variables where X is the correlation factor that is calculated on the map between two features: i, j. There is a consistent diagonal perfect correlation which makes sense as some specific features correlation with itself should be perfect (1 in a numerical case).

- **Feature Engineering** can be a bit messy, and sometimes conjoint into the exploratory data analysis section. Ojo clearly understands good ML practices, since he used a natural logarithm to enclose outliers in pursuit of a normal distribution. Outliers tend to be bad when building models, and are not helpful for loss-functions.

- **Machine Learning** is an iterative process, especially for inexperienced users in the pursuit of machine learning. When Ojo implemented his machine learning into this project, he explains different model mechanics. For most beginner projects using smaller data sets, random forests and gradient boosting are pretty solid models.

  A major part of a successful ML algorithm is the training process, where model hyperparamaters are tweaked in an iterative effort.

  There is a bias variance trade-off. If the data is simpler, often the training data is unfit. Model variance is with more complexity and higher variance, which leads to poor predictive capacity of unseen untrained data.

IV. Autonomous Vehicles

Autonomous vehicles are a growing interest for many engineering followers. Many major car companies are looking into making this a feasible future for the average car-driver. The thought of a car being able to work on its own might seem like a surreal endeavour, but machine learning is making it a reality.

There is a lot of criticism with the thought of self-driving cars. To say self-driving cars are a necessity in society in 2022 might be a stretch, many deem it as a benefit that should be looked into however. Others put emphasis on the reduced risk of road traffic deaths, which can limit the mistakes humans would make at the wheel.

The technology behind autonomous driving is referred to as ADAS or advanced driver-assistance systems. The perception of a vehicle is similar to the perception of a text-recognition software, and is primarily different in the form of magnitude and complexity. As text-recognition software identifies and detects letters as specific objects, many sensors and cameras on cars allow a machine to recognize objects, and classify them. Some easy objects to conceptualize for ADAS systems are other vehicles, people, lanes, trees etc.

Accidents can still occur when training a ADAS model, and this is when information is classified wrong (ex: a missed car that has a color very similar to the road). This is where training the ADAS model is important. Millions of hours of supervised data trains most ADAS models, that will help the algorithm and effectively teach it to do something similar. Driving is such a complex task with millions of parameters, so for an effective algorithm, it is important to have enormous amounts of data.

There are benefits to different types of sensors. Lidar radars are more active then the traditional camera sensors. Lidar sensors are excellent at measurements (ex: distance between cars, stop sign etc). Traditional cameras are more vulnerable to weather and misinterpretation, but can also identify more than just some abstract distance.

A common conversation with self-driving vehicles are why Machine Learning should be used compared to vision systems. ADAS replicates human driving, by identifying assortments of rules and evolving through iterations. A car produces varying situations (varying input vector), but at the end the car should make the correct decision with minor differences, and this adds value to machine learning which is notorious for finding patterns.

V. My First Project

I decided that after building a stronger knowledge in ML, that I wanted to take part in a small project. I've heard really strong opinions about TensorFlow's open source library. They have thorough documentation, which helped me build my first ever model.

I understand the work I am learning from is an expansion of what they have already created and understand this work is created under the MIT Licence, and plan on abiding to all that entails.

The plan during this project is to classify articles of clothing based off a given image. 70,000 articles of clothing were used, each picture was 28 x 28 pixels (however converted later to 1 x 784 pixels).

Firstly, we will import the libraries we require. Tensorflow will be required for building the ML Model, Numpy for array manipulation, and Matplotlib will be used for graphing purposes.

```
import tensorflow as tf
import matplotlib.pyplot as plt
import numpy as np
```

Listing 1. Importing External Libraries

The next step is data ingestion. We plan on using a built in TensorFlow clothing dataset, to simplify the process. In future projects, I plan on implementing my own data-set.

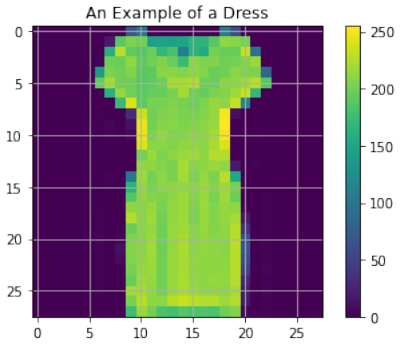
```
fashion_mnist = tf.keras.datasets.
    fashion_mnist

(train_images, train_labels), (
    test_images, test_labels) =
    fashion_mnist.load_data()
```

Listing 2. Data Ingestion MNIST Dataset

There are 10 output neurons, describing how confident the algorithm is in each specified article of clothing. From index[0] to index[9] of output neurons, they coincide with: t-shirt, trouser, pullover, dress, coat, sandals, shirt, sneaker, bag, ankle boot.

It is also important to check out our data, here is an example of the 20th index of our training images with a labeled output. This output describes a dress, and for us humans it is quite trivial. Our plan is to teach a computer in a similar way.



However, the computer prefers reading in grey-scale, so I wrote a function to do that to both our training images, and testing images.

```
def preProcessGrayScale(data):
  return data / 255.0
```

Listing 3. Grey-Scale Function

It was advised that we verify the labelled data and see its accuracy before training the model. Although hard to see, the labeled data is correct by labeling this data: ankle boots, t-shirt, t-shirt, dress.

Now for the juicy work, building a neural network model. We will use TensorFlows' Keras library, to build a sequential machine learning model. A sequential model is used when each layer has one input tensor and one output tensor. In any Machine Learning application where there is layer sharing, I will refrain from using a sequential model,

```
model = tf.keras.Sequential([
    # Switches 28 x 28 to 1 x 724
    tf.keras.layers.Flatten(input_shape
        =(28, 28)),

    # RELU activation function
    tf.keras.layers.Dense(128, activation
        ='relu'),

    # Densly connected neural networks
    tf.keras.layers.Dense(10)
])
```

Listing 4. Building Machine Learning Model

After flattening a layer into 1 x 784, and then having two dense layers of which the middle layer uses the RELU activation function, and the output layer proceeds to give confidence in all outputs, it is time to train the model.

By training the model, many steps are nested into this. 1. We are feeding training data to the model, 2. we are asking the algorithm to make a prediction, 3. we are checking the answer.

We used the Adam Optimizer for stochastic gradient descent for the suited properties, our loss function cross-entropy method helps guide our model in a correct direction, and we inform the model that it should use accuracy as its metric as that is what we plan on improving.

We now have our model created, and now need to train the model with the information we have. The neural network is trained, in our case for 25 iterations. Listed are the 1st, 10th, and 25th epoch of our training, describing the loss, and accuracy.

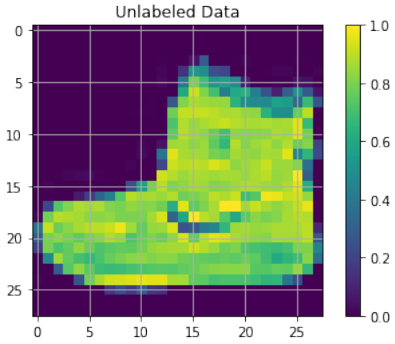
Epoch 1/25
1875/1875 [==============================] - 5s 2ms/step - loss: 0.4989 - accuracy: 0.8232

Epoch 10/25
1875/1875 [==============================] - 4s 2ms/step - loss: 0.2369 - accuracy: 0.9121

Epoch 25/25
1875/1875 [==============================] - 4s 2ms/step - loss: 0.1590 - accuracy: 0.9400

This is great news for our algorithm. After 25 iterations, we predicted the correct article of clothing at around 94 percent. As you perform more and more iterations, your accuracy increases less and less. It should be noted from epoch 1 to epoch 10 there was a greater increase, but from 10 to 25, it was half of what was achieved before.

To validate our results, lets look at the first index of our predictions. As humans, we can tell that these are ankle boots.



Lets see what our algorithm thinks,

- T-Shirt: 2.3181090380575142e-10
- Trousers: 3.5778595364188703e-11
- Pullover: 1.0420617412585167e-13
- Dress: 1.910504835388703e-14
- Coat: 4.149132981634551e-10
- Sandals: 1.7115862647187896e-06
- Shirt: 7.386435907363875e-10
- Sneaker: 6.548938108608127e-05
- Bag: 1.0021435237206333e-08
- Ankle Boots: 0.9999327659606934

Thankfully, our algorithm can detect similar to what us as humans think. This shows accuracy in our pattern. In most cases, there would be more than one test to validate algorithm.

VI. Review of Machine Learning For Absolute Beginners: Second Edition

I plan on reviewing the reading of Machine Learning for Absolute Beginners: Second Edition. Reviews on www.goodreads.com describe this as a reasonable first read in machine learning. The book includes the essence of machine learning algorithms, techniques and algorithms, and resources for ML.

This read isn't specifically about how to build algorithms, it branches into many things including sociological effects on what may happen as these algorithms become more relevant in society. Some are concerned that these algorithms will work against humanity, when looking for who may be the fittest, where some sort of offspring characteristics may do very bad to the world. Others are concerned about the job market. It is a reasonable thought to ask if ones job may be automated in the future. It turns out that more jobs are adoptable to automation than one may think. Jobs that are repetitive and do not require much user interaction can easily be adapted to some sort of algorithm that can achieve something similar. Accounting, serving, taxi-driving, many business roles are all common to what might be a shortage of jobs in the near future.

Olivier Theobald describes input information compared to output information in a machine learning model well. There is a entity of command, and an entity of data in the input section. It then is simulated in a sort of a machine learning model, which then leads to some output, and in many cases outputs.

There are different types of regression analysis. This is the act of attempting to estimate relationships based off a (or many) variable(s).

Linear regression is arguably the most popular type of regression model. This model attempts to predict a variable in a linear fashion which typically follows $y = mx + b$, where m is the slope, x is the dependent variables value, and b is the vertical offset.

Logistic regression is a very popular method, and I question its relevance to machine learning as they both seem to have a intent of some sort of activation function. The books example is a sigmoid function, but there are others such as RELU activation functions as well. Logistic regression is strong for yes or no questions, the book refers to "pregnant" or "not pregnant". Its binary classification nature allows for it to be a common utility in fraud detection, disease diagnosis, spam detection. There is also multinomal logic regression, that is known to have more than two specific outcomes.

There is also ridge regression. Ridge regression is notorious for having relevance in machine learning in the case that there is a strong correlation with independent variables. It does introduce a bias matrix to avoid the problem of over-fitting.

A polynomial regression is what one might expect, it is quite similar to a linear regression however can be fit with any power n. As linear regression also does, the pursuit of a polynomial regression is performed by reducing the mean-squared error. This uses a partial derivative matrix, but most common data entry applications have functionality to make it easier for us. We can be given some set of data, and see it looks similar to $f(x) = x^7$, so we can do a polynomial regression to find the curve of best fit. This won't yield the exact answer of $f(x) = x^7$, but it will inform the algorithm of what you think the original shape is, and will give some variation of it whether it be vertical/horizontal stretch, or vertical/horizontal change.

A new thought that has crossed my mind through reading about neural network architectures is the processing power you may need. As a beginner myself, I never have ventured into any machine learning projects with need for major infrastructure. For my previous project example, in terms of 70,000 pieces of data, it still only took around a minute to generate a machine learning algorithm that could predict unlabeled data. Some may say 70,000 data points is no small number, however that is not what is counted for in what is considered for the time it takes to complete the training of a machine learning model. The clothing prediction had a 28 x 28 array of data, sophisticated machine learning algorithms like GitHub Co-Pilot is known to have millions of parameters. That is where it gets complex, it is significantly more tough to test millions of parameters, as you want to see the significance of each parameter. My personal processor could perform a small machine learning project, but anything significant would need some sort of advanced chip or a cluster of GPUs.

Algorithms are an essential part to any machine learning process. Some popular models are Markov model, support vector machine (SVM), and q-learning. Thankfully by what I chose to use earlier, TensorFlow seems to be the industry standard for advanced machine learning models. Tensorflow allows for different structures of machine learning models in a very simplistic code model.

VII. A Deeper Look Into Self Driving Cars

Mindy Support does a great job of educating anyone who's interested in "How Machine Learning Makes Self-Driving Cars a Reality". This read goes through the decision-making process a self driving car makes, how it receives its information to base its decisions off of, the industry and competition in autonomous driving and many other thoughts through the process.

It seems that Tesla seems to have created this wave of self-driving vehicles, however other companies such as Waymo and Alibaba are also starting to develop infrastructure and models for self-driving vehicles. It seems to be that every major car manufacturer has some sort of electric vehicles in production in 2022, or at least is planning on it.

Imagine self driving cars being the wires between your eyes and your brain when you drive. Your eyes to the car are many different forms of sensors, and your actions would be performed automatically by the car. There is radar sensors, lidar sensors, and camera sensors. These assist in the identification of what objects are around it, such as other cars, lines, trees, construction, lights etc. The car also has an electronic inertial and GPS measurement system to assist the decisions the car makes.

It should be noted that not all cameras are made with the same purpose. There are a variety of cameras with different quality, FOV (field of view). Some cameras are capable of looking very far into the distance but do not have a wide FOV, which may be good to see that their is a flooding one-hundred metres ahead, but can't tell you if an innocent child is running across the sidewalk perpendicular to where you're driving. Thankfully we also have larger POV angles, that help assist in situations that you want a wide close view. A machine learning algorithm with only one camera would be very useless, would only mildly help, and would definitely not be approved by authorities. Thankfully, we can use a collaboration of all cameras, and allow the machine learning algorithms to make valid estimates off an arrangement of data.

Radar and LiDar perform with different functionalities, which ultimately output the same information. Radar sensors use waves, by sending them out and seeing when they reflect back and calculating based off that. This is very useful in the case that there is reduced visibility with rain, fog etc. These are not as accurate, but definitely more reliable. They describe the objects location based off radio waves. LiDar detection uses light, which can help replicate the physical world, but is susceptible to more noise.

Tesla refuses to use LiDar detection due to its lack of appeal to the sleek design of their models, but other cars like Waymo and other companies are known to use LiDar as it seems to have the advantage of data collection.

Data Annotation is a major bottleneck in autonomous artificial intelligence projects, by improving the thought of autonomous driving in itself, the easiest way would be to improve the capabilities of data annotation.

Although self-driving vehicles are the primary interest of many automotive enthusiasts, there is another major bottleneck in legislation. As you may predict, the government isn't easily allowing self-driving vehicles to take over the road, as there are many issues that need to be sorted out. A common problem talked about with self-driving vehicles is the trolley problem. If a self-driving car has to decide to hit four adults or two children, which children will they hit and why? The computer must follow some sort of algorithm, and this could in fact lead to racism, prejudice, or other forms of "ism's".

Self driving cars fall under five categories. Categories are listed from level 1 to level 5. In ascending order, the first level describes minimal autonomous capabilities whereas level five describes fully autonomous functionalities where the so called "driver" has absolutely no job to complete.

The most common self-driving car accident is rear-hit collisions. The cause of rear-ended collisions are always the fault of the person behind them, and there is little to nothing a autonomous vehicle can do to protect itself from this case.

The National Law View's research shows that there are only 9.1 driver-less car crashes per million km's driven. The average person's lifetime driving is around $37,000$ hours, where driving $75\frac{km}{h}$ leads to $2,845,125$ km driven. Which means if one was to use the current driver-less car crashes, in ones lifetime would get into almost 27 car accidents. This is way more than if you were to just drive normally, which is against the favour of self-driving vehicles.

It is very dangerous working with autonomous driving, and especially testing. Tesla had reported three team-members killed in pursuits of testing in the past. It is surprising that this has gone almost unnoticed, as I have not heard about this from mainstream media.

One applied science article I enjoyed reading to grow my learning in ML was "A Survey on Theories and Applications for Self-Driving Cars Based on Deep Learning Methods", authored by Jianjun Ni, Yinan Chen, Yan Chen, Jinxiu Zhu, Deena Alim and Weidong Cao.

The paper is responsible at overlooking some theories and applications of self-driving vehicles. Similar to what was described earlier in this paper about how there are five levels of autonomous driving. This paper disregards all vehicles that are in the range of level 0 to level 2. This means that the only self driving vehicles that are talked about in this research is level 3 to level 5. The reason for this is level 3 systems and above can be described in an easier framework. The four described frameworks are environment perception system, autonomous decision systems, control executive system, and monitor systems. The monitor system overlooks action monitoring, process monitoring, and problem management. Environment Perception System is responsible for identifying what is going on around the vehicle. This may include lanes, traffic, people, or any other major obstacles. This is referred to as prior environment knowledge". Prior driving knowledge is where autonomous decisions are made, including the behavior, local public path planning. The last part describes control executive system. This controls parts of the car such as steering, gas, and breaks.

The research describes a comparison of complex autonomous systems vs their hardware and software speeds. Autonomous driving requires a quite rigorous load, and therefor the bottleneck is then placed on how well rapid data analysis is handled. They describe the NVIDIA DRIVE PX2 driver-less car platform and how it is capable of 30 trillion deep-learning operations per second (which is surprisingly only level 4 autopilot).

TensorFlow is the most common Machine Learning Library used with self-driving vehicles. The benefits of TensorFlow is simplicity with complex neural networks in heterogeneous distributed computational environments. It is known best for its support to both model and train convelutional neural networks, and recurrent neural networks, but also supports others.

Convolutional neural networks are a very popular neural network for self-driving car models. They consist of one input layer, many convolution and pooling layers, connecting layers, and multiple output layers. This can be seen as useful to a car as there is many parts that need to be controlled.

These models use most commonly the three activation functions.

$$\text{Sigmoid: } R = \frac{1}{1+e^{-y}}$$
$$\text{Tanh: } R = \frac{e^{y}-e^{-y}}{e^{y}+e^{-y}}$$
$$\text{reLU: } \max(0, y)$$

The procedure of binocular obstacle detection is a subsidiary of binocular cameras mixed with disparity maps. The sequence of stereo matching describes cost matching, disparity computing, disparity imaging, which leads to obstacle determination. When using a MC-CNN which is an abbreviation for matching cost-convolutional neural network with multiple image patches, where the convolutions merge.

There is still improvement to be made in terms of scene classification. A recommendation made was that cars should understand a high-level semantic, for example knowing where it is whether being close to a school or on a mountain-side cliff driving, both options can alter how one might drive. GPS seems like a reasonable solution for this.

Scene recognition has a plethora of convelutional neural networks. There varies structures from GoogLeNet, ResNet, VGGNet, 2D-LTSM, CNN/RNN. It is a bit tough to compare the performances of these networks against eachother as they've all been tested it seems on different driving data sets. For example, GoogLeNet has a 83.1 percent on its own data, but a 92.90 percent on Scenel5's dataset.

Research from the University College of London by Jack Stilgoe talks about autonomous vehicles, machine learning, responsible innovation, self-driving cars, social learning. An interesting part of their research describes the first 'truly' self-driving vehicles on the roads in 2015. A car driving off of autopilot drove through 15 states from San Francisco to New York City with no influence from a driver.

An interesting thought is if a car is 99 percent self-driving. The way you would react to that 1 percent where it needs your assistance would make you basically on guard at all times. Regulators are also suspicious of these technologies but more importantly these irregularities. Many leaders in the driving industry have strong doubts in autonomous vehicles due to the fact that even with the best training, it seems like a car will need some sort of support from supervision which ultimately could defeat the purpose of self-driving in all means for most who want to seek the benefits of it.

In 2016, there was the first ever self-driving death accident. This was published by Tesla and described as a technology failure. Although the company had 'the best safety rating of any car ever tested' (2013), many who once had trusted self-driving vehicles are now starting to have second thoughts on the matter.

I wanted to see what academia looks like in the improvements of safety in safety algorithms for processors in self-driving vehicles. Dr. Samuel Manoharan in "An Improved Safety Algorithm For Artificial Intelligence Enabled Processors in Self Driving Cars" describes his research as a better form of safety based off enabled processors. He describes that perfect safety from an algorithm is a goal that simply not be achieved, but work with artificial intelligence enabled processors helps the probability of safety.

The proposed solution on the front of the car has range sensors, distance sensors, steering sensors, lane camera, foot camera, the side of the car has two distance sensors and a hand camera, and a 360 degree camera. The back of the vehicles has a GPS unit, a range and a distance sensor.

With the AI processor, the flow of work would be categorized in terms of data collection unit, pre-processing and classification, decision making, decision-making to steering, brake, clutch, accelerator.

The implementation of human eyes (or lack thereof) is replicated with classification on a XEON scalable processor, built with deep learning to help with obstacles that are identified (or at least hopefully...) by sensors and cameras. This will help control the components of the automotive vehicle such as clutch, accelerator, brakes, doors, windows, etc.

To improve safety, Dr. Samuel Manoharan describes "[t]he conjugate nonlinear optimization, smoothens and interpolates the available paths and a proper trajectory planning, and the linear-quadratic gaussian propagation handles the vehicle control to be initiated on detecting the obstacle. The flow chart below in fig, 4 shows the path planning and the control initiated by the self driving car on detecting the obstacle."

Initially some form of data will be captured, sensor fusion will then be performed. When an obstacle is detected, the deep learning assists in the path of classification of the obstacle. This will help decipher the difference between lines, humans, colored lights, signs, etc. When sent in to the ECU, and applying the conjugate non-linear optimization and linear quadratic Gaussian propagation, it then becomes a simple decision tree. If at stop lights, or there is a pedestrian, or its a vehicle, there are all different actions.

I was interested in learning about automated tests of deep-neural-network driven autonomous cars, and was glad to come across a noted 2018 ACM/IEEE conference paper in Software Engineering. There is great diversity across this paper, written by Yuchi Tian at the University of Virginia, Suman Jana of the University of Columbia, Kexin Pei from Colombia University, and Baishakhi Ray from the University of Virginia.

This paper goes into automated tests in self-driving vehicles, the software and its engineering, testing and debugging, security and privacy, computing methodologies in neural networks.

One thing I enjoyed about this read was the variety of research they compiled about real-world accidents involving self-driving cars. Hyundai Competition, Tesla Autopilot and Google self-drivng car all had reported accidents in this report. The dates of accidents are in respect to when the testing took place, and do not really have a significant impact on the likelihood of accidents (despite ones that may make the argument that certain times of the year have significantly more dangerous weather). The causes of these accidents sparked my curiosity more than dates, and they all have different causes of failure. What this makes me think is these machine learning models are too sophisticated to describe a common failure reason. The Hyundai Competition was subject to failure based off rain fall, the Tesla autopilot failed due to image contrast, and the Google self-driving car failed to estimate its speed. These all end up having different outcomes, the Hyundai Competition crashed during testing, the Tesla autopilot mode killed the driver, and the Google self-driving cars hit a bus while shifting lanes. What I take from this is there is a magnitude of things that can go wrong with autonomous vehicles, and there is no simple solution for something that most likely (and statistically) will never be perfect.

Their present a systematic technique that maximizes neural coverage in a DNN based system, showing the autonomous cars behavior has a strong correlation to empirically demonstrated neuron coverage. The paper also digs into neuron coverage after the presence of fog or other visual alterations. They built a "DeepTest" model where the top three DNN's from a specific Udacity driving challenge are tested.

The paper looks at different DNN architectures. They look at FFCNN's (Feed Forward Convelutional Neural Networks), they look at RNN's (Recurrant Neural Networks). They do a great job at describing differences in neural network architectures. They talk about how "RNN's unlike CNN's, allow loops in the network. Specifically, the output of each layer is not only fed the following layer but also flow back to the previous layer. Such arrangement allows the prediction output for previous inputs (e.g: previous frames in a video sequence) to be also considered in predicting current input" (305). One thing I found interesting about RNN's are the leveraging gradient descent with back propagation which helps with training.

The paper looks at DNN autonomous cars by answering questions: how do we systematically explore the input-output spaces of an autonomous car DNN? How can we synthesize realistic inputs to automate such exploration? How can we optimize the exploration process? How do we automatically create a test oracle that can detect erroneous behaviors without detailed manual specifications.

One part of the paper I wish I understood a bit more about is the increasing coverage with synthetic images. From what I understand, some form of matricies are modified in pursuit of modeling brightness, contrast etc about self-driving cars. One of their tables describes how these matrices are modified. The affine transformations include translation, scaling, shearing, rotating. A transformation in essence is displacement, the scaling is maintaining an aspect ratio with the translation, shear is an angled bend, and rotation uses a rotation matrix. All of these matricies are represented in 2 x 3 matrix size.

To learn about increasing coverage, they proposed a greedy search algorithm which leads to a increase in neuron coverage. The algorithm takes a transformation, and seed images, and outputs synthetic test images. In the midst of this algorithm, it operates using a few linear data structures, more precisely a stack and a queue. The way they describe the algorithm is "[t]he algorithm take a set of seed images I, a list of transformations T and their corresponding parameters as input. The key idea behind the algorithm is to keep track of the transformations that successfully increase neuron coverage for a given image and prioritize them while generating more synthetic images from the given image. This process is repeated in a depth-first manner to all images." (307).

Although being interested in self-driving cars, I had yet to hear the Udacity self-driving challenge. After looking into Udacity, it seems like a strong entity in the self-driving vehicles industry. They also have a platform to educate individuals on the fundamental principles of self-driving vehicles. This is something that I find very interesting, they expect you know Python, C++, Linear Algebra, and Calculus. They educate users with a variety of different courses. The scope goes from computer vision, to sensor fusion, localization, path planning, control and trajectory etc.

There tests operated on a Keras framework, with a DeepTest prototype. There were a few described models including chauffeur, rambo, and epoch. They all have different amounts of neurons, chauffeur has 1427 513, rambo has 1625 3801 13473, and epoch has 2500. They all have close and almost exact MSE's being at most 0.2 off. Chauffer model is described as "one CNN model for extracting features from the image and one LSTM model for predicting steering angle. The input of the CNN model is an image while the input of the LSTM model is the concatenation of 100 features extracted by the CNN model from previous 100 consecutive images" (308). Proceeding to Epoch where it is described as "as the pre-trained model for epoch is not publically available, we train the model using the instructions provided by the authors. We use the CH2002 dataset from the Udacity self-driving challenge for training the epoch model.

When talking about transformations and parameters used by DeepTest for generating synthetic images, paramaters alter depeing on so. For translation, scale, shear, they all intake two separate parameters, either a movement, a scalar of both axes, and a tilt for x and y. In terms of talking about editing these matricies in terms of rotation, contrast, or brightness - they take a single parameter, degree, gain, and bias. There are forms of blurs but require a square matrix of some form, including averaging blur, gaussian blur - however median blur, bilateral filter just have a different form of parameters.

The relation between neuron coverage and test output should interesting results. This is the first question that was planned on being solved during this conducted research. The chauffeur's overall sub-model had a spearman correlation of -0.10. The chauffeur's CNN sub-model had a spearman correlation of 0.28. The chauffeur's LTSM sub-model had a spearman correlation of -0.10. The Rambo's overall sub-model had a spearman correlation of -0.11. The Rambo's S1 sub-model had a spearman correlation of -0.19. The Rambo's S2 sub-model had a spearman correlation of -0.10. The Rambo's S2 sub-model had a spearman correlation of -0.10. The Rambo's S3 sub-model had a spearman correlation of -0.11. The Epoch's only model had a spearman correlation of 0.78.

The authors did a strong job at visually representing these figures, in one example 309. A graph describes the Jacard Distance on the y axis, and the models described earlier. The graph is describes the "Difference in neuron coverage caused by different image transformations". What this means is depending on which type of neural network that is used, the Jaccard Distance is affected differently based off the individual logistics of each in different ways. It seems like the Chauffeur convelutional neural network, and the rambo S2 has around an average Jaccard Distance with respect to the others. The outliers seem to be Epoch and Rambo S3 which seem to be a few deviations above. The Chauffeur LSTM model is suspiciously low.

Depending on which model is used, unique erroneous behaviors can occur, but the interesting part of it is that different algorithms have different amounts of errors in different places. What this helps with is describing what algorithms have strengths in what. For example, epoch had no problem managing shear, but was extremely effected by brightness. The chauffeur managed blur well, but epoch did not as well. It seems like epoch has the largest sum of issues, both in simple transformations, and composite transformations in terms of rain or fog.

I wanted to build a stronger understanding of activation functions in neural networks. I took the chance to read deeper into research "Activation Functions in Neural Networks", written by Siddharth Sharma, Simone Sharma out of the Department of Computer Science and Engineering at GIT, Jaipur, as well also Anidhya Athaiya at GIT in Japiur as well. This paper does a strong job at describing neuro synaptic junctions, and outputs of functions and how they are effected by activation functions in terms of non-linear mapping and complicated mappings between the inputs and corresponding outputs.

Just to remind anyone who may not remember what an activation function is, "[a]ctivation functions are specially used in artificial neural networks to transform an input signal into an output signal which in turn is fed as input to the next layer in the stack. In an artificial neural network, we calculate the sum of products of inputs and their corresponding weights and finally apply an activation function to it to get the output of that particular layer and supply it as the input to the next layer" (310). Essentially, there are different styles of activating neurons, that are described by specific functions. The two that I am understanding of is the sigmoid activation function, and the reLU activation function.

Little did I know, there are a significant amount more of activation functions. Some listed in the paper: binary step function, linear, sigmoid, tanh, reLU, Leaky ReLU, Parametrized ReLU, Exponential Linear Unit, Swish, SoftMax. The paper does a good job at going more in depth with each activation function. The linear step function is a simple piece-wise function, saying 1 when bigger than 0, and 0 when not. The linear activation function is self-explanatory, the sigmoid activation function is arguably the most popular activation function in machine learning, as an S shaped graph that operates of $f(x) = \frac{1}{e^{-x}}$. The reLU activation function operates off $f(x) = max(0, x)$. Computer scientists also enjoy using hyperbolic functions such as $tanh$ which looks quite similar to the sigmoid function. Leaky reLU function is an altered version of the reLU activation function, where numbers smaller than 0 on the x axis are not zero but rather a very small number. The parametrized reLU (rectified linear unit) allows some sort of parameter for the negative slope. The swish function takes advantage of summation for individual classes.

Disease Prediction Model

For the last section of my machine learning study, I decided to build a neural network model based off disease prediction, and try to follow less of a guide - taking the challenge of learning it myself.

A popular free data set website called Kaggle led me to find a data set that if used correctly, can help in medical science. The idea of this data set is off of 132 parameters, the assortment can describe out of 42 diseases that can be predicted. Thankfully, the data set was made for machine learning, so there was comma separated values file (csv), for both training and testing.

First, we imported all required modules,

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

import tensorflow as tf
from tensorflow.keras.models import
    Sequential
from tensorflow.keras import layers
from tensorflow.keras.layers import Dense
```

Listing 5. Module Importing

Then proceed to load the data,

```
test_data = pd.read_csv('Testing.csv')
train_data = pd.read_csv('Training.csv')
```

Listing 6. Load Data CSV

Starting to fill the frame of the neural network, with three layers in a sequential model, both using activation functions reLU.

```
# Describing a Sequential Qualifer
classifier = Sequential()

# Adding an Input Layer with ReLU
    activation function.
classifier.add(Dense(64, activation = "
    relu", input_dim = X_train.shape[1]))

# Adding an Input Layer with ReLU
    activation function.
classifier.add(Dense(32, activation = "
    relu"))
```

Listing 7. Framework of Neural Network

After compiling this sequential model with optimizing with adam, crossentropy loss function, and the value metric of accuracy, the model being called with the .summary() function yields,

Layers:
Dense3 (Dense), Output Shape: (None, 64), Param (8512)
Dense4 (Dense), Output Shape: (None, 32), Param (8512)
Dense5 (Dense), Output Shape: (None, 41), Param (8512)

After training the model in just 3 epochs, it had performed at 100 percent accuracy.

References