International Journal For Technological Research In Engineering
Volume 8, Issue 11, July-2021 *ISSN (Online): 2347 - 4718*

MACHINE LEARNING BASED AUTOMATIC STATE SWITCHING

[1]Fahad Ansari, [2]Gaurav Kumar Singh, [3]Abhay Kumar, [4]Surendra Singh Chauhan
[1,2,3]student(UG), [4]Assistant Professor
Department of Computer Engineering
Galgotias University, Greater Noida, India

Abstract- Self-driving cars have become a trending subject with a significant improvement in the technologies in the last decade. The project purpose is to train a neural network to drive an autonomous car agent on the tracks of Udacity's Car Simulator environment. Udacity has released the simulator as open source software and enthusiasts have hosted a competition (challenge) to teach a car how to drive using only camera images and deep learning. Driving a car in an autonomous manner requires learning to control steering angle, throttle and brakes. Behavioral cloning technique is used to mimic human driving behavior in the training mode on the track. That means a dataset is generated in the simulator by user driven car in training mode, and the deep neural network model then drives the car in autonomous mode. Three architectures are compared with respect to their performance.

Though the models performed well for the track it was trained with, the real challenge was to generalize this behavior on a second track available on the simulator. The dataset for Track_1, which was simple with favorable road conditions to drive, was used as the training set to drive the car autonomously on Track_2 which consists of sharp turns, barriers, elevations and shadows. To tackle this problem, image processing and different augmentation techniques were used, which allowed extracting as much information and features in the data as possible. Ultimately, the car was able to run on Track_2 generalizing well. The project aims at reaching the same accuracy on real time data in the future.

1. INTRODUCTION

The purpose of a Self-driving car project is to build a better autonomous driver. The car should be able to drive itself without falling off the track, with accelerating and braking at appropriate places. This chapter covers the problem statement of the project in brief and the higher-level solution approach used.

1.1 Problem Definition

Udacity released an open source simulator for self-driving cars to depict a real-time environment. The challenge is to mimic the driving behavior of a human on the simulator with the help of a model trained by deep neural networks [1]. The concept is called Behavioral Cloning, to mimic how a human drives. The simulator contains two tracks and two modes, namely, training mode and autonomous mode. The dataset is generated from the simulator by the user, driving the car in training mode. This dataset is also known as the "good" driving data. This is followed by testing on the track, seeing how the deep learning model performs after being trained by that user data. Another challenge is to generalize the performance on different tracks. That means, training the model using the dataset created on one of the tracks, and testing it on the other track of the simulator.

1.2 Solution Approach

The high-level architecture of the implementation can be seen in Figure 13

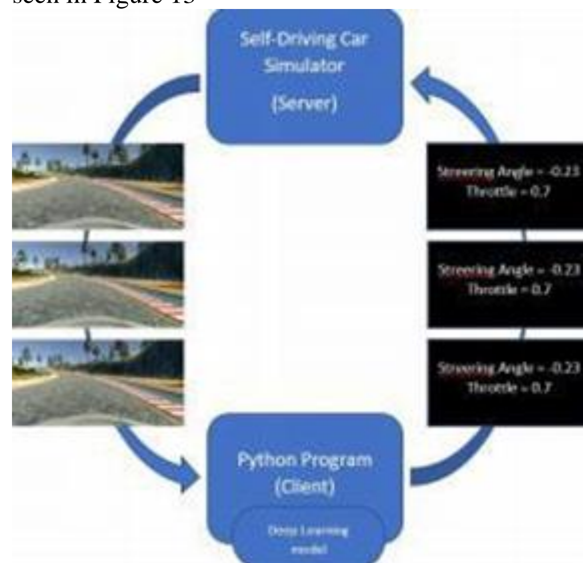

Fig 1 Implementation Architecture

The problem is solved in the following steps:

The simulator can be used to collect data by driving the car in the training mode using a joystick or keyboard, providing the so called "good-driving" behavior input data in form of a driving_log (.csv file) and a set of images. The simulator acts as a server and pipes these images and data log to the Python client.

The client (Python program) is the machine learning model built using Deep Neural Networks. These models are developed on Keras (a high-level API over Tensor flow). Keras provides sequential models to build a linear stack of network layers. Such models are used in the project to train over the datasets as the second
step. Detailed description of CNN models experimented and used can be referred to in the chapter on network architectures.

 Copyright 2021.All rights reserved.

· once the model is trained, it provides steering angles and throttle to drive in an autonomous mode to the server (simulator).
· These modules, or inputs, are piped back to the server and are used to drive the car autonomously in the simulator and keep it from falling off the track.

1.3 Technologies Used

Technologies that are used in the implementation of this project and the motivation behind using these are described in this section. Tensor Flow: This an open-source library for dataflow programming. It is widely used for machine learning applications. It is also used as both a math library and for large computation. For this project Keras, a high-level API that uses Tensor Flow as the backend is used. Keras facilitate in building the models easily as it more user friendly. Different libraries are available in Python that helps in machine learning projects. Several of those libraries have improved the performance of this project. Few of them are mentioned in this section. First, “Numpy” that provides with high- level math function collection to support multi-dimensional metrics and arrays. This is used for faster computations over the weights (gradients) in neural networks. Second, “scikit-learn” is a machine learning library for Python which features different algorithms and Machine Learning function packages. Another one is OpenCV (Open Source Computer Vision Library) which is designed for computational efficiency with focus on real-time applications. In this project, OpenCV is used for image preprocessing and augmentation techniques.

The project makes use of MiniConda Environment which is an open source distribution for Python which simplifies package management and deployment. It is best for large scale data processing. The machine on which this project was built, is a personal computer with following configuration:

· Processor: Intel(R) Core i5-7200U @ 2.7GHz
· RAM: 8GB
· System: 64bit OS, x64 processor

Network Architectures

There were various combinations of architectures tried, predicting the steering angle and input for the car to drive in autonomous mode. Neural Network layers were organized in series and various combinations of Time-Distributed Convolution layers, MaxPooling, Flatten, Dropout, Dense and so on are used in architectures. The best performing ones are shown in detail. Refer to the model listings for the parameters used to build them. The high-level view of layers used to build the models is shown in the accompanying architecture figures.

Model

NVIDIA released architecture for self-driving cars [4] and it is used in the project for reference to solve the problem and for comparing with the various other architectures tried. Different architectures have been standardized over the years for building sequential models of CNN like AlexNet, VGG-Net, GoogLeNet, ResNet and so on. Model_2 is architecture similar to AlexNet, with a slight variation by tweaking parameters to suit the problem in the project. Refer Figure 17 for overview of the architectures. Model_3 is architecture similar to VGG-Net, with variations by tweaking parameters to suit the problem in the project. Refer for overview of the architectures.

2. PROPOSED METHODOLOGY

In this section, key concepts that are used in the implementation of this project and the motivation behind using these concepts are described.

2.1 Convolutional Neural Networks (CNN)

CNN is a type of feed-forward neural network computing system that can be used to learn from input data. Learning is accomplished by determining a set of weights or filter values that allow the network to model the behavior according to the training data. The desired output and the output generated by CNN initialized with random weights will be different. This difference (generated error) is back propagated through the layers of CNN to adjust the weights of the neurons, which in turn reduces the error and allows us produce output closer to the desired one.

CNN is good at capturing hierarchical and spatial data from images. It utilizes filters that look at regions of an input image with a defined window size and map it to some output. It then slides the window by some defined stride to other regions, covering the whole image. Each convolution filter layer thus captures the properties of this input image hierarchically in a series of subsequent layers, capturing the details like lines in image, then shapes, then whole objects in later layers. CNN can be a good fit to feed the images of a dataset and classify them into their respective classes.

2.2 Recurrent Neural Networks (RNN)

RNN are a class of artificial neural networks where connections between units form a directed cycle. Recurrent networks, unlike feed forward networks, have the feedback loop connected to their past decisions, ingesting their own outputs as input (like a memory). This memory (feedback) helps to learn sequences and predict subsequent values, thus being able to solve dependencies over time. For example, consider the case when the next word in a sentence is dependent on a previously occurring word or context. RNN will be an excellent choice for such scenarios. They are designed to recognize patterns in sequences of data, such as text, handwriting and so on. They are also applicable to images that can be separated (decomposed) into a sequence of patches. Neural networks have activation functions to take care of the non-linearity and to squash the gradients or weights in certain range. Some of these functions are

 Copyright 2021.All rights reserved.

sigmoid, tanh, RELU and so on that are the building blocks of RNN. Though these are very powerful, there are some shortcomings in conventional RNN, such as the well-known problem of vanishing or exploding gradients. For a detailed description of this problem, please refer to "the study conducted by Ujjwalkaran". Also, training might take a very long time. To overcome this, new classes of RNN implementations have been developed recently. Some of these are below:

2.2.1 LSTM (Long Short-Term Memory)

LSTM is a class of RNN, an improved version that tackles the vanishing and exploding gradient problems. LSTM block is made up of a forget gate, input gate and an output gate. A small demonstration of how these gates work in the LSTM cell. Each gate that is involved in designing the LSTM cell is discussed as follows: Forget gate: This decision is made by a sigmoid activation layer (shown in red) called the "forget gate layer".

Input gate: Only the selective input is passed through. The "input gate layer" decides which values to be passed through. Next, a tanh layer of activation will help create an update to the state. (as shown in orange).

Output gate: Selective parts of the cell state are going to output. Then, the cell state is put through tanh layer (to push the values to be between −1 and

1) and multiply it by the output of the sigmoid layer gate. Many remarkable results can be achieved with LSTM compared to RNN. A lot of people these days use the LSTM instead of the basic RNN and they work extremely well on a large variety of problems.

2.2.2 GRU (Gated Recurrent Unit)

The Gated Recurrent Unit is similar to the LSTM that was discussed in the "Section.2.2.1". Gated mechanisms are used, almost like LSTM and designed to update its memory content using the update gate that can be compared with the input gate. The GRU uses a reset gate to reset its memory, comparable to forget gate of the LSTM.Most research show the study that the LSTM and GRU outperforms the traditional RNN unit. However, studies have not found such big performance differences between the LSTM and GRU.

2.3 Time-Distributed Layers

Another type of layers sometimes used in deep learning networks is a Time distributed layer. Time-Distributed layers are provided in Keras as wrapper layers. Every temporal slice of an input is applied with this wrapper layer. The requirement for input is that to be at least three-dimensional, first index can be considered as temporal dimension. These Time-Distributed can be applied to a dense layer to each of the time steps, independently or even used with Convolutional Layers. The way they can be written is also simple in Keras. There is not much instructional information out there about the Time Distributed layers, but a discussion released by Jason Brownlee, "How to use Time Distributed Layers for LSTM" [6] can serve as a great tutorial for beginners. I have included the link in the references with this report.

2.4 RCNN (Combination of CNN and RNN)

The acronym used to denote this combination as RCNN (Recurrent Convolutional Neural Networks). In recent times, there have been many implementations using RCNN. There is another abbreviation for this term (R-CNN) as region-based CNN which is a popular technique for object detection in images. In this project, every time this term is used, it will refer to Recurrent CNNs. Lukas Weist, in a post on Wiki TUM writes, "It can be assumed that the combination of RNN with other networks, especially CNN, will be continued. The improvement and the ability to handle sequential data enhances the CNN a lot and brings new unexplored behavior. This is an exciting and promising area of artificial intelligence" There are several techniques or methods for which this combination can be realized. Individually, both CNN and RNN are extremely useful in image classification (more about spatial characteristics of data) and sequence prediction (temporal characteristics of data). The hybrid models can
have a bunch of convolution layers and another branch of RNN (may include LSTM or GRU or both) in parallel or they can be stacked in series. In this project, there are experiments for a variety of architectures. There are results plotted by different implementations that were tried on the driving dataset in the chapter on "Results".

3. UDACITY SIMULATOR AND DATASET

Udacity has built a simulator for self-driving cars and made it open source for the enthusiasts, so they can work on something close to a real-time environment. It is built on Unity, the video game development platform. The simulator consists of a configurable resolution and controls setting and is very user friendly. The graphics and input configurations can be changed according to user preference and machine configuration. The user pushes the "Play!" button to enter the simulator user interface. You can enter the Controls tab to explore the keyboard controls, quite similar to a racing game. The first actual screen of the simulator and its components are discussed below. The simulator involves two tracks. One of them can be considered as simple and another one as complex that can be evident. The word "simple" here just means that it has fewer curvy tracks and is easier to drive. The "complex" track has steep elevations, sharp turns, shadowed environment, and is tough to drive on, even by a user doing it manually. There are two modes for driving the car in the simulator: (1) Training mode and (2) Autonomous mode. The training mode gives you the option of recording your run and capturing the training dataset. The small red

 Copyright 2021.All rights reserved.

sign at the top right of the screen in the Figure 9 depicts the car is being driven in training mode. The autonomous mode can be used to test the models to see if it can drive on the track without human intervention. Also, if you try to press the controls to get the car back on track, it will immediately notify you that it shifted to manual controls. The mode screenshot can be as seen in Figure 10. The simulator's feature to create your own dataset of images makes it easy to work on the problem. Some reasons why this feature is useful are as follows: · The simulator has built the driving features in such a way that it simulates that there are three cameras on the car. The three cameras are in the center, right and left on the front of the car, which captures continuously when we record in the training mode.

· The stream of images is captured, and we can set the location on the disk for saving the data after pushing the record button. The image set are labelled in a sophisticated manner with a prefix of center, left, or right indicating from which camera the image has been captured.

· Along with the image dataset, it also generates a datalog.csv file. This file contains the image paths with corresponding steering angle, throttle, brakes, and speed of the car at that instance.
Column 1, 2, 3: contains paths to the dataset images of center, right and left respectively
Column 4: contains the steering angle Column value as 0 depicts straight, positive value is right turn and negative value is left turn.
Column 5: contains the throttle or acceleration at that instance
Column 6: contains the brakes or deceleration at that instance
Column 7: contains the speed of the vehicle

4. PROPOSED MODEL

This section consists of the configurations used to set up the models for training the Python Client to provide the Neural Network outputs that drive the car on the simulator. The tweaking of parameters and rigorous experiments were tried to reach the best combination. Though each of the models had their unique behaviors and differed in their performance with each tweak, the following combination of configuration can be considered as the optimal:

• The sequential models built on Keras with deep neural network layers are used to train the data.
• Models are only trained using the dataset from Track_1.
• 80% of the dataset is used for training, 20% is used for testing.
• Epochs = 50, i.e. number of iterations or passes through the complete dataset. Experimented with larger number of epochs also, but the model tried to "overfit". In other words, the model learns the details in the training data too well, while impacting the performance on new dataset.
• Batch-size = 40, i.e. number of image samples propagated through the network, like a subset of data as complete dataset is too big to be passed all at once.
• Learning rate = 0.0001, i.e. how the coefficients of the weights or gradients change in the network.
• ModelCheckpoint() is the function provided in Keras to save checkpoints and to save the best epoch according to the validation loss.

There are different combinations of Convolution layer, Time-Distributed layer, MaxPooling layer, Flatten, Dropout, dense and so on, that can be used to implement the Neural Network models. Out of around ten different architectures I tried, three of the best ones are discussed in the chapter on network architectures.

5. CHALLENGES AND ISSUES

In the implementation of the project the deep neural network layers were used in sequential models. Use of parallel network of network layers to learn track specific behavior on separate branches can be a significant improvement towards the performance of the project. One of the branches can have CNN layers, the other with the RNN layers and combining the output with a dense layer at the end. There are similar problems that are solved using RESNET (Deep Residual networks), a modular learning framework. RESNET are deeper than their „plain‟ counterparts (state-of-art deep neural networks) yet require similar number of parameters (weights). Implementing Reinforcement Learning approaches for determining steering angles, throttle and brake can also be a great way of tackling such problems. Placing fake cars and obstacles on the tracks, would increase the level of challenges faced to solve this problem, however, it will take it much closer to the real-time environment that the self-driving cars would be facing in the real world. How well the model performs on real world data could be a good challenge. The model was tried with the real-world dataset, but there was no way of testing it on an environment like a simulator. The big players in the self-driving car industries must be already trying this on their autonomous vehicles. This would be a great experiment to see, how this model really works in the real time environment.

6. RESULT

The machine learning based automatic state switching is a vehicle that is capable of sensing its environment and navigating without human input. It can detect environments using a variety of techniques such as radar, GPS and computer vision.

7. CONCLUSION

This project started with training the models and tweaking parameters to get the best performance on the tracks and then trying to generalize the same performance on different tracks. The models that performed best on 1 track did poorly on Track_2, hence there was a need to use image augmentation and processing to achieve real time generalization.

Copyright 2021.All rights reserved.

The use of CNN for getting the spatial features and RNN for the temporal features in the image dataset makes this combination a great fit for building fast and lesser computation required neural networks. Substituting recurrent layers for pooling layers might reduce the loss of information and would be worth exploring in the future projects.

It is interesting to find the use of combinations of real world dataset and simulator data to train these models. Then I can get the true nature of how a model can be trained in the simulator and generalized to the real world or vice versa. There are many experimental implementations carried out in the field of self-driving cars and this project contributes towards a significant part of it.

Acknowledgement

This is the matter of great privilege for all of us to submit this project entitled "Machine learning based automatic state switching" We take pleasure in expressing our deep sense of gratitude for providing necessary guidance to SURENDRA SINGH CHAUHAN in the department of Computer Science & Engineering at GALGOTIAS UNIVERSITY, Greater Noida for his kind and constant encouragement made it possible for us to complete this project work. He has given us pragmatic sense to look into the matter and we are also highly obliged for his persistence in making the project complete. It gives us great pleasure to extend our sincere thanks our colleagues, who helped us directly or indirectly to complete the project. Last but not the least, we would like to express our gratitude towards our parents for their kind co-operation and encouragement which helped us in completion of this project.

REFERENCES

1. Oliver Cameron, "Challenge #2: Using Deep Learning to Predict Steering Angles", Published on 11 Aug 2016, https://medium.com/udacity/challenge-2-using-deep learning-to-predict-steering-angles-f42004a36ff3, accessed Jul 2017
2. Ujjwalkaran,"An intuitive Explanation to Convolutional Neural networks",Published on 11 Aug 2016,https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets, accessed Nov 2017
3. Andrej Karpathy, "The unreasonable effectiveness of Recurrent Neural Networks", Published on 21 May, 2015, http://karpathy.github.io/2015/05/21/rnn effectiveness, accessed Nov 2017
4. Mariusz Bojarski, "End-to-End Deep Learning for Self-Driving Cars", Published on 17 Aug, 2016, https://devblogs.nvidia.com/deep-learning-self-driving-cars/, accessed Nov 2017
5. Jason Brownlee, "How to use Time Distributed Layers for LSTM", https://machinelearningmastery.com/timedistributed-layer-for-long-short-term memory-networks-in-python/, accessed Nov 2017
6. Lucas Weist, "Recurrent Neural Networks - Combination of RNN and CNN", Published on 7 Feb 2017, https://wiki.tum.de/display/lfdv/Recurrent+Neural+Networ ks+Combination+of+R NN+and+CNN, accessed Nov 2017
7. Dmytro Nasyrov , "Behavioral Cloning.NVidia NeuralNetwork in Action." , Published on 21 Aug 2017, https://towardsdatascience.com/behavioral-cloning-project-3- 6b7163d2e8f9 , accessed Jan 2017
8. Sihan Li , "Demystifying ResNet", Published on 20 May 2017, https://arxiv.org/abs/1611.01 186, accessed Jan 2017
9. Franchois Chollet, "Building powerful image classification models using very little data", Published on 5 June 2016, https://blog.keras.io/building- powerful-image classification- models-using-very-little- data.html, accessed Jan 2017
10. Ivan Kazakov, "Vehicle Detection and Tracking", Published on 14 May 2017, https://towardsdatascience.com /vehicle-detection-and- tracking-44b851d70508, accessed Feb 2017

 Copyright 2021.All rights reserved.