Behavioural prediction of traffic participants using DNN

Kosta Ivancevic
kosta.ivancevic@campus.tu-berlin.de

Kenan Bagci
k.bagci@campus.tu-berlin.de

Teham Bhuiyan
teham.bhuiyan@tu-berlin.de

***Abstract*—One of the prerequisites for successful autonomous driving, in which a car is capable of modifying its behaviour based on the behaviour of its environment, is successfully predicting driving trajectories. Unlike deterministic approaches that scale well for short-term predictions, Machine Learning models are used for generating long-term predictions. If experienced human drivers are able to predict the driving behaviour up to few seconds, at least the same performance is expected from autonomous vehicles.**

In this project, we propose an approach in using Deep Neural Networks as autonomous navigation systems for autonomous vehicles. More precisely, we develop a proof-of-concept prototype for generating long-term vehicle trajectory and lane changing predictions. Our approach relies on using Recurrent Neural Networks as a Machine Learning model, trained on the US-NGSIM-101 dataset. Our design outperforms a similar approach used in [1]. The trajectory predictions are 64 frames long and our model achieves an average error of 0.402m when predicting lateral road position, as well as 5.42km/h when predicting vehicle velocity.

These results are the proof-of-concept that Recurrent Neural Networks can achieve significant results for predicting sequences and trajectories, which imposes their usage in autonomous vehicles.

***Index Terms*—deep learning, recurrent neural network, autonomous vehicles, TensorFlow, US-NGSIM-101**

I. Introduction

As the name suggests autonomous driving is the property of a vehicle to be able to navigate without human interference. In order to do so, the vehicle needs to be aware of its environment. Vehicle environment consists of roads, signals and various traffic participants such as other vehicles and even humans. It is important to predict the behaviour of other traffic participants in order to react accordingly. For example, if a vehicle slows down all other vehicles in the lane need to slow down as well, in order to avoid accidents.

We can roughly divide autonomous driving into the following categories [2]:

- Recognition - Identifying the surrounding objects such as cars, traffic signs, pedestrians, etc.
- Prediction - Building internal models that predicts the future states of the environment.
- Planning - Incorporating Recognition and Prediction to plan the future sequence of driving actions that will enable the vehicle to navigate successfully.

The fundamental step towards autonomous driving is predicting a vehicle trajectory. With correct predictions, the vehicle can generate or modify driving patterns, perform, change or cancel certain actions. In order to create predictions, two approaches can be adopted. First are deterministic prediction algorithms that scale well for short-term predictions. The second approach to solving prediction problems is using Machine Learning, a concept in which a machine learns to mimic human behaviour.

Described as "The Computers ability to learn with data, without being explicitly programmed", Machine Learning is currently the most common approach in designing navigation systems for autonomous vehicles. Thomas Mitchell postulated the following: "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E." [3]

A division of Machine Learning models can be made based on the output values that are produced by the model. If the model solves the problem of classifying inputs to one of the discrete classes, we call this problem *classification*, and the ML model is called a classifier. A classification problem to trajectory prediction would be to estimate a driver intention from some set of possible intentions (turn left, turn right, brake, accelerate, etc.). In contrast to classification, *regression* problems predict a set of real values for each input. Therefore, the regression problem of trajectory prediction would be to produce a real-valued (x-y) trajectory coordinates that are the prediction to the input. Although they seem so, classification and regression are not very different, and one can derive the classification problem from regression one and vice versa. The approach of solving each of the problems is very similar.

For making a trajectory prediction, there are a lot of different Machine Learning models, each having advantages and drawbacks. Because the trajectory consists of consecutive frames that exhibit temporal dependency, Recurrent Neural Networks [4] are naturally imposed as the model for this application, due to their structure, shown on Figure 1.

As the name suggests, neural networks are comprised of a set of interconnected neurons. A mathematical model of a neuron, first introduced by McCulloch and Pitts, resembles the biological neuron. This concept, despite being rather simple, has enormous potential because neurons are connected together in more complex structures. Let us analyze arbitrary neuron from Figure 1. When evaluating the neural network, the bottom-up approach is adopted. In other words, the neurons from the bottom layers are evaluated first, followed by the progress further up the structure of a neural network.

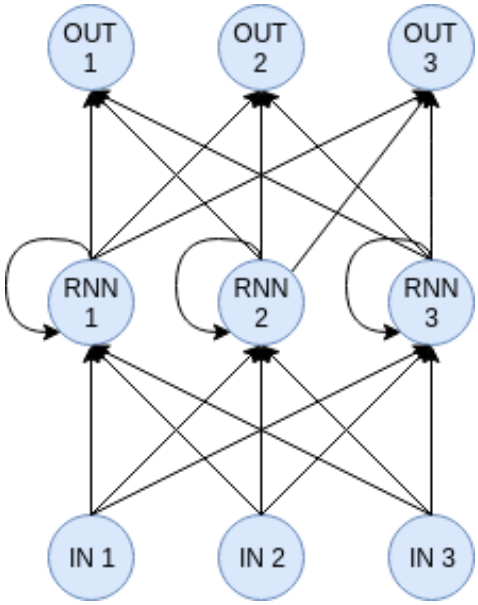


Fig. 1: Example Recurrent Neural Network consisting of input layer, output layer and one hidden layer with 3 recurrent neurons

When evaluating the network, the value of each neuron in the network is calculated. Any neuron from the neural network has a set of *m* inputs that interconnect it to all the neurons in previous layer[1]. The output of a neuron is used to connect it to all the neurons in the next layer. The output of neuron *j* is calculated as a weighted sum of all its inputs, which is subjected to the activation function in order to introduce non-linearity.

$$out_j = activation(\sum_{i=1}^{m} w_{ij} * in_i) \quad (1)$$

Here, weights correspond to the level of correlation between neurons. As the neurons are more correlated, the neuron *j* is more likely to be active when neuron *i* is active.

The fundamental reason behind their excelling performance when predicting sequences is the feedback loop that uses previous outputs as inputs for predicting the current output. This allows the information to persist [5]. In other words, past frames contribute to better understanding the current frame.

The LSTM[2] [4] cell is a variation of regular RNN cell. The main idea is to replace the fixed self-loop weight with context-dependent weight. This is particularly useful when predicting long sequences, where regular RNNs often tend to fail. Three gates control the information flow as they regulate which information goes through, depicted on Figure 2.

- Forget Gate - regulates how fast the network forgets previous states.
- Input gate - decides which values are going to be updated.
- Output gate - regulates the output of a network.

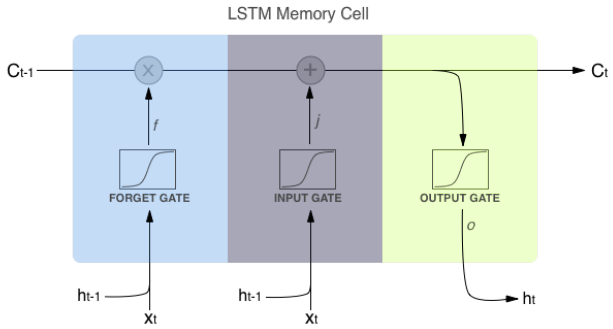


Fig. 2: Illustration of the LSTM cell. **C** corresponds to cell output, **x** corresponds to cell input wile **h** corresponds to the state of a cell

II. Related Work

Although using DNNs[3] in the field of autonomous driving is relatively young, its application is growing with significant research done in this field.

An approach similar to ours is shown in [1], where authors solved a regression problem of predicting trajectories using LSTM network, with RMS error of 0.7m for predicting long-term lateral vehicle position. Furthermore, the dataset that is used is based upon is NGSIM, as in our project. This served as a good reference point throughout the development phase of our project.

Instead of a supervised approach, autonomous driving via reinforcement learning is used in [2] and [6]. Motivated by the performance on games such as Atari and GO, these papers scale up this approach to the problem of autonomous driving. Using 3D simulations, [6] solves the problem of vehicle state transitions via Markov Decision Process, where the agent traverses through the set of states by performing actions [4]. Further, [6] introduces hybrid CNN-RNN agents that perform well for image-form input data. A similar approach is demonstrated in [2]. The downside of these approaches is that they are concerned with simulation datasets instead of real-world scenarios. Further, in order for the model to perfect generalization performance, the agent needs to explore all possible states, some of which may be dangerous for the driver. Therefore, this approach scales far better in simulations than in real-world scenarios.

An end-to-end learning approach is introduced in [7]. It relies on predicting the steering angle based on the image input, using a hybrid C-LSTM[5] neural network. The fundamental idea behind this approach is to exploit both spatial and

[1]Recurrent Neural networks are an exception because apart from all inputs, they have a feedback loop that enables them to exploit temporal dependencies in the data.

[2]Long short-term memory

[3]Deep Neural Networks

[4]Actions consist of accelerating decelerating and steering

[5]Convolutional Long Short-Term Memory Recurrent Neural Network

temporal dependencies. First is done by performing convolutions, while the second is done through recurrences. Likewise, [8] also incorporates 3D convolutions, that are followed by recurrent layers. The network input is a set of the Udacity dataset images. The prediction is a continuous steering wheel angle.

Recurrent neural networks are also used in [9] and [10]. In both approaches, vehicle coordinates are used in order for LSTM cells to produce a set of candidate trajectories for each of the surrounding vehicles. This produces an occupancy matrix for each time step in the predicted sequence. In other words, the network suggests that a certain vehicle will be at a certain road location with some probability. Although this approach is very interesting, the downside of predicting each vehicle trajectory with a separate LSTM network is not scalable for dense traffic. Isolating a vehicle from its surrounding is not practical, as the vehicle behaviour is very dependent on its neighbouring vehicles.

An interesting approach in posing trajectory prediction as a classification problem is demonstrated in [11]. In this paper, vehicle behaviour is predicted at the intersection. The input to an LSTM-based model is a sequence, and the output value corresponds to one element in a discrete set of possible driver intentions.

III. Programming Environment and Libraries

This project is a proof-of-concept so we implemented our solution in high-level programming language Python. The main motivation is the simplicity of Python. The possible drawback of using high-level language is inefficient performance. This was not a concern, because we intend only to demonstrate the correctness and feasibility of our approach, rather than to have a finalized application, which is ready for deployment. The source files are implemented as Jupyter Notebook files, grouped by their function. We used Jupyter Notebook because it enables us to segment the code into functional blocks. Moreover, it enables us to execute a segment of the code, rather whole code. For example, the time required to perform dataset derivation is about 20 hours. Instead of running it each time some part of the code after needs change, we run it once and ran the remainder of the code independently. This drastically reduces debugging time.

*1) **TensorFlow**:* TensorFlow [12] is an open source library developed by Google. We used TensorFlow in order to create neural networks and to train them.

*2) **Pandas**:* Pandas is a data analysis library for Python [13]. It allows the user to read and write various data formats as CSV, Excel, SQL etc. The dataset we used was provided as a CSV-file. With pandas, we were able to read, reshape and create modified versions of the file.

*3) **Numpy**:* Numpy provides a large variety of mathematical functions. We used Numpy to perform mathematical-based and array-based operations. This is due to the fact that Numpy has fast internal implementation and it reduced computation time for performing various operations.

*4) **Matplotlib**:* Matplotlib is a standard plotting library for Python. All the plots in this paper are created using Matplotlib.

IV. Dataset

The dataset, that is used for training and evaluation of our model is derived from the US-NGSIM-101 dataset, which is provided by The Federal Highway Administration of the United States.

"Researchers for the NGSIM program [14] collected vehicle trajectory data on southbound US 101 in Los Angeles, CA, on June 15th, 2005. The study area was approximately 640 meters in length and consisted of five lanes. Eight synchronized digital video cameras, mounted from the top of a 36-story building adjacent to the freeway, recorded vehicles passing through the study area. NG-VIDEO, a customized software application developed for the NGSIM program, transcribed the vehicle trajectory data from the video. This vehicle trajectory data provided the precise location of each vehicle within the study area every one-tenth of a second, resulting in detailed lane positions and locations relative to other vehicles." [15]

The highway was recorded for approximately 45 minutes, segmented into three intervals of 15 minutes: 07:50 to 08:05; 08:05 to 8:20; and 8:20 to 8:35. It is done in such a manner because these segments correspond to three different levels of congestion. Therefore, a model that is trained with such diverse dataset, has high generalization power. This means that it is able to accurately predict future behaviour, independently of the scenario it finds itself in. A comprehensive dataset analysis is done in [16]

The original dataset consists of 25 columns representing each vehicle. In order to have a detailed description of the trajectory, we reduced the number of necessary columns and limited the section to highway 101.

Figure 3 shows a segment of the dataset [15] that we used. Each vehicle has its individual ID. Moreover, vehicle trajectory is identified by the time stamp. corresponding to the lateral and longitudinal location, acceleration and current velocity. For each vehicle, preceding and following vehicle IDs are given, as well as the identifier for the lane it is currently using. Also, vehicle class is used because different vehicles exhibit different behaviour (for example, a motorcycle is more likely to participate in overtaking than a regular truck.)

An example of the trajectory is depicted on Figure 4, where the lateral distance from the end of the road is represented through time. By close observation, it is possible to see two significant leaps that occur in a large majority[6] of vehicles. By further analysis of the time stamps for the frames before and afterwards the leap, it is conclusive that the leap represents the boundary between one section and another. The first step of our project was to split the dataset into all sections so that the sections don't have the leap. Apart from the leaps that are present in the vehicle trajectories, some vehicles exhibit repetitive behaviour that is also visible on Figure 3. This

[6]Some vehicles are not present in all sections. These vehicles may have one or none leaps in time

	Vehicle_ID	Total_Frames	Global_Time	Local_X	Local_Y	v_Vel	v_Class	Lane_ID	Preceding	Following	v_Acc
287162	1000	470	1118847268200	46.520	2071.419	59.56	3	5	0	992	5.65
287163	1000	470	1118847268300	46.520	2077.438	59.94	3	4	0	1022	1.00
287164	1000	470	1118847268400	46.519	2083.445	60.00	3	4	0	1022	0.00
287165	1000	470	1118847268500	46.519	2089.443	60.02	3	4	0	1022	0.28
287166	1000	470	1118847268600	46.520	2095.445	60.06	3	4	0	1022	0.54

Fig. 3: Original format of the US NGSIM dataset for the vehicle 1000. All remaining information that is used for learning is derived from this base format.

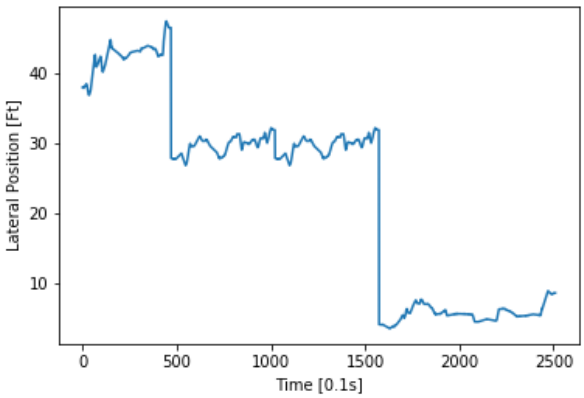


Fig. 4: Lateral position of the vehicle with ID 1000 through time. Two positional leaps are observable which divide the trajectory in three sections. Furthermore, in section 2, repetitive behaviour is observable.

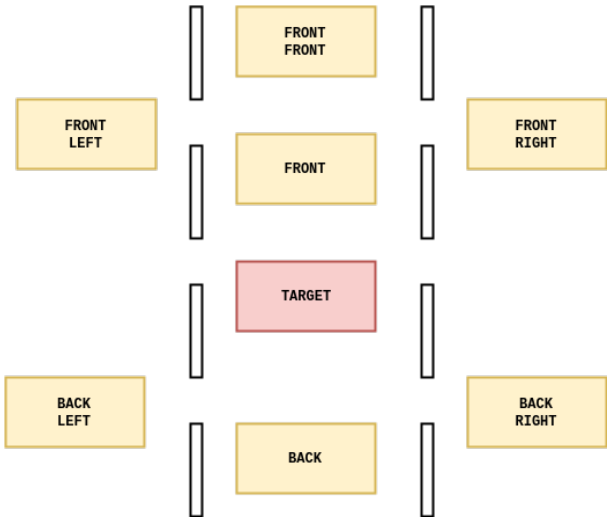


Fig. 5: Illustration of the target vehicle and all the vehicles that comprise the surrounding set for the given target

repetitive behaviour occurs only in the second section at some of the vehicles. It can be seen as repeating the whole trajectory once more, while the number of frames, in which the vehicle is present, is doubled. Our approach removes the second part of the doubled trajectories by detecting the repetitive behaviour and keeping only the first occurrence of the vehicle in time and removing all repetitions.

A. Dataset Derivation

It is natural to assume that the behaviour of a certain driver relies, to some extent, on all the surrounding vehicles. Therefore, predicting a sequence of future movement, only based on the past trajectory points leaves a lot of room for improvement. Instead, we adopted the following approach, graphically illustrated on Figure 5:

The prediction of the trajectory is made with respect to the given vehicle, but also all the surrounding vehicles.

The term surrounding is used rather loosely. In our approach a set of surrounding vehicles consists of the following:

- Font-front
- Front
- Front-left
- Front-right
- Back
- Back-right
- Back-left

It is, perhaps, possible to achieve better results using a different surrounding set, but this set is chosen as a compromise between performance and simplicity. A similar approach was used in [1]

Given that US-NGSIM-101 originally doesn't have such relationships, the step of creating the set of surrounding vehicles was done manually, thus setting the foundation for the dataset that can be used in the learning process.

For each vehicle, and for each frame in which is present, a set of surrounding vehicles[7] is found and represented with its frame-wise trajectory information.

A comprehensive trajectory description that we used consists of two current coordinates (x and y), where x describes the distance from the left road edge and y represents the distance from the beginning of the road. Furthermore, vehicle trajectory is described by projecting velocity on the horizontal and vertical axis, vertical acceleration and vehicle class. Together this forms a set of 6 parameters for each of 8 vehicles.

[7]If some vehicle is not present in the dataset, it is treated as all-zero input (e.g. driving in leftmost lane eliminates front-left and back-left vehicles)

This constitutes the format of the input to our neural network to the sequence vectors with the length of 48.

Having selected a target vehicle and a target time stamp for extraction, the surrounding set is created as follows:

- Front, Front-front, Back - vehicles are found using the `Preceding` and `Following` columns in the original dataset.
- Front-left, Front-right, Back-right and Back-left vehicles are found by eliminating all vehicles whose `Lane ID` is not a decrement or an increment of the `Lane ID` from the target vehicle. Afterwards, the vehicles with the closest Y position of the target vehicle are selected.

B. Data Filtering

Given that the original dataset is created from raw video extraction, it has a significant amount of noise. [16] After evaluation, we concluded that generalization performance is constrained with such a high amount of noise. In our approach, we used data filtering in order to reduce the noise. Approximation is done using Savitzky-Golay [17] 5^{th} degree polynomial filter. High order filtering fits the function better, while in contrast, low order filtering drastically simplifies the function that it approximates. The degree of a polynomial approximation was chosen to compensate between reducing noise and oversimplifying the function. An example of the filtering process can be seen on Figure 6.

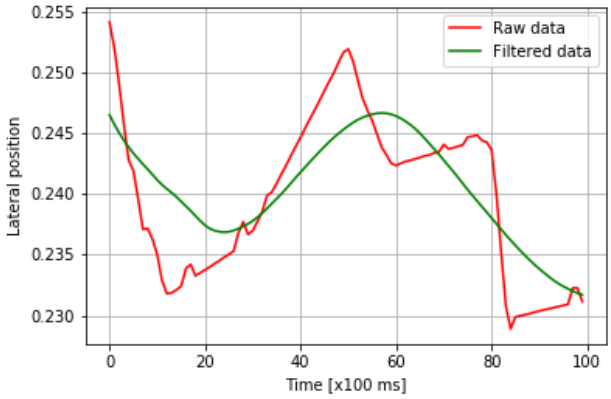


Fig. 6: A Sequence of consecutive lateral positions before and after filtering.

V. Implementation and Evaluation

For implementing a Machine Learning model for generating vehicle trajectory and lane changing prediction for highway driving, we decide upon a Neural-Network-based approach, implemented in Google's machine learning framework TensorFlow [12].

A. Model

Our Machine Learning model is implemented as a Recurrent Neural Network [4] consisting of 4 recurrent layers, each having 256 neurons. After further analysis, we have concluded that a recurrent cell, implemented as LSTM [4] yields far better performance than basic RNN cell. We used compared the usage of Rectified Linear activation function with RNN and Sigmoid activation function with LSTM. We concluded that Rectified Linear activation function does not work with LSTM. Further, we concluded that the Sigmoid activation function provides better results because it converges in the gates of the LSTM, unlike ReLU which makes a strong positive feedback loop within the gates. This is due to the fact that for positive values ReLU is not bounded from above, unlike Sigmoid function that is limited in the range ± 1. The illustration of our network is depicted on Figure 7.

Our approach can be defined as: ***Predicting sequence from a sequence***. In practice, this can be seen as feeding the network a sequence of 64 consecutive frames and predicting the target trajectory in 64 future frames. Naturally far better results are expected by predicting a single point in time from a sequence, for example, predicting one point in trajectory from previous 64 frames. The problem with this approach is that it is computationally a very intensive task to evaluate a complex neural network with such frequency. Consider sampling rate of 20 fps. This means that in order for the network to only keep up with time, it is necessary to evaluate a whole neural network each 20ms. For actual prediction, it is necessary to be able to predict multiple times faster than the sampling rate, which makes this approach infeasible and not scalable. In contrast, our approach yields a far better scalability while slightly increasing the error.

The input to our network is a sequence of values representing the target vehicle and its set of surrounding vehicles, each represented by:

- Local X - Lateral distance from the end of the road
- Local Y - Longitudinal position of the vehicle
- X velocity - X component of the overall velocity
- Y velocity - Y component of the overall velocity
- Acceleration - Vertical acceleration
- Class - Car, Truck or Motorcycle

A similar approach is used in [1] and we find that the given information set, that can be easily gathered (for example by LiDAR[8]), is complete and can be used to fully describe the trajectory and behaviour of the traffic participant.

The output of the network consists of a sequence of `Local X` and `Y velocity` for the given vehicle in 64 consecutive frames. Predicting a trajectory consists of predicting X and Y coordinates in a road-aligned coordinate system. Predicting directly `Local Y` degrades the generalization performance because the range of values is significantly larger than the range of values of `Y velocity`. Therefore, it is intuitive the best performance is predicting `Y velocity` and performing the deterministic calculation for the `Local Y`.

B. Training

The concept of machine learning is for a model to perform well in various scenarios without being explicitly programmed.

[8]A method of using laser reflection to measure the distance from a target

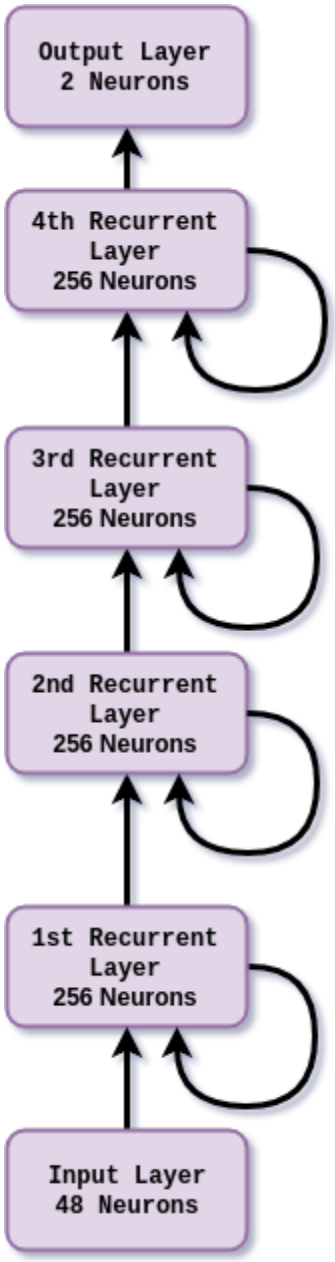


Fig. 7: An illustration of our Machine Learning model, the 4-layered Recurrent Neural Network having 256 neurons per layer and LSTM cell architecture.

We say that a Neural Network learns from experience if any given input maps as close as possible to the desired output. This is achieved by converging to the optimal set of weights in the network. This is done in the learning phase, that has the purpose to:

- Supply the model with a large amount of various example inputs, for which a given action is expected
- Minimize the overall difference between true values and model-generated outputs

In this project, we train our neural network by randomly selecting a batch of 100 different vehicles, from which we extract a set of 64 input values and 64 label values. During the training process, the weights between neurons are set in order for the error between the model prediction and the true labels to be minimized. We trained our network for 100000 iterations with the learning rate of 0.001. The training process is depicted on Figure 8.

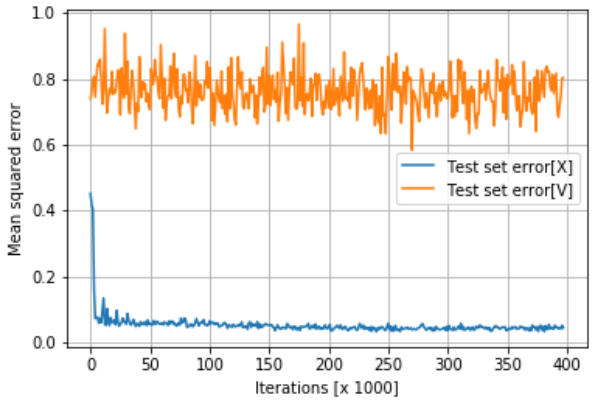


Fig. 8: Velocity and lateral position test errors during training phase

By observing the training performance, the generalization performance is characterized by oscillations and difficulty to converge to the optimum set of weights. For that purpose, we are iteratively (each 250 training iterations) testing the current model during training with test data in order to keep track of the generalization performance. The model with the best results on the randomly selected 100 trajectories is used as a final model. This means that the development of our ML model can be seen as 400 different models. At the end of the training, the model with the best performance on test data is kept as a final model. The frequency of model validation is set as 250 in order to have a fine resolution of the errors during the training phase. Increasing this frequency further yields in a better model selection, but as a result, significant increase in training time. Likewise, increasing the number of test trajectories would contribute to equality[9] between tests on different models. Similar to above, here we also compensate between the evaluation quality and training time.

C. Evaluation

Model evaluation is done on a set of 64 frames, that correspond to 6.4 seconds of driving time. When evaluating the generalization performance of a Machine Learning model, we use the part of the dataset, that is previously unseen by the model. 20% of all vehicles are used for this purpose. From this set, we randomly pick a starting point in a trajectory and the sequence of 128 frames that follows. From this set, the first 64 frames are used to generate a prediction, while the last

[9]Some models may be tested with more complex trajectories and therefore their errors would be higher in comparison to the models which are tested with simple ”forward” trajectories

64 frames are used to compare the true label and the model prediction, therefore calculating the error.

1) Average Frame Error Distribution: Evaluating the average frame-wise error is done on a sample of multiple trajectories. Let `n` be the number of test iterations. Further, let $\texttt{Y}_{\texttt{true}}$ be true values of a trajectory and let $\texttt{Y}_{\texttt{pred}}$ be values of a model prediction. Than the single trajectory error is defined as:

$$e = \frac{1}{64} \cdot \sum_{i=1}^{64} |Y_{\text{pred}}(i) - Y_{\text{true}}(i)| \tag{2}$$

and total trajectory error is defined as:

$$E = \frac{1}{n} \cdot \sum_{j=1}^{n} e_j \tag{3}$$

Significant measurement of the generalization performance is the average value of `E` as well as the distribution of values $\texttt{e}_\texttt{j}$. Intuitively, a low average trajectory error is a characteristic of a good model. Likewise, a tight error distribution is also a desirable property. Tight distribution means that a model has a small derivation of expected performance, and therefore, with higher certainty, we can assume that the actual error is inside the ϵ-neighbourhood from its expected value. Figure 10 depicts the frame distribution for predicted lateral position and vehicle velocity. Our best model achieves an average error of 0.402m with the standard deviation of 0.149m for predicting lateral road position. When predicting velocity, our model makes an average error of 5.42km/h with the standard deviation of 1.59km/h. This is the most pessimistic approach, which is highly unlikely to occur in the real world. Here, all the errors contribute with the same sign, and no errors cancel each other out.

2) Average Frame-Wise Errors: Another interesting measurement to analyze is the distribution of the error within the prediction of 64 frames. Here, the errors are not added in absolute value. Instead, they are added as they occur in the real-world scenario. Let `n` be the number of test iterations. The average error for each frame is calculated as follows:

$$e(i) = \frac{1}{n} \cdot \sum_{i=1}^{n} Y_{\text{pred}}(i) - Y_{\text{true}}(i) \tag{4}$$

Figure 12 depicts the average error distribution for each frame. As it can be observed, the errors increase as the prediction frame is further, both when evaluating lateral position and velocity. Here, we add raw errors instead of their absolute values. This gives an average case error for each frame below 0.1m when predicting the lateral position and 3.0km/h when predicting velocity.

3) Individual Trajectories: It is interesting to analyze how the errors are distributed in distinct trajectories. Figure 14 depicts how the model errs on 10 different trajectories.

When predicting lateral position, the significant majority of trajectories maintain the error within ± 0.5m. Unfortunately, there are some outliers, which are not properly predicted by our model. We aim to reduce the number of outliers, as well as the errors, which are made in these situations.

When predicting the velocity, the model mainly makes an error in the range ± 5km/h. Unfortunately, in some scenarios, the error is quite significant. These scenarios correspond to stop-and-go scenarios where the model failed to predict hard braking. Our aim is to increase the prediction performance in stop-and-go scenarios.

D. Evaluation Remarks

All the above-mentioned results are for predicting a sequence of 64 frames from a sequence lasting 64 frames. It is conclusive that better results can be achieved by predicting shorter sequences or even predicting a single value from a sequence. For predicting shorter sequences, deterministic algorithms achieve better results than Neural-Network-based approaches, while predicting a single value from a sequence fails to scale for systems with the high sampling frequency.

We have used an approach in which we predict a set of two independent sequences from a sequence of frames. Naturally, having two distinct neural networks, one predicting velocity and one predicting lateral position would perform better.

VI. Future Work

Every project has room for improvement and in that sense, this project is not an exception. Furthermore, the lack of time for development further reduced the scope of the project, leaving some important concepts for future upgrade.

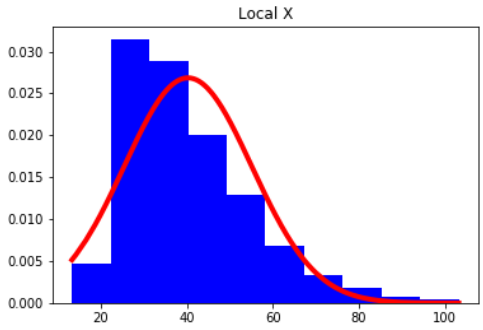
We managed to create a proof-of-concept prototype of Machine Learning model based on Recurrent Neural Networks, that is able to predict future trajectory based on a movement sequence of a target vehicle and its surrounding set of vehicles.

A. Additional Information Source

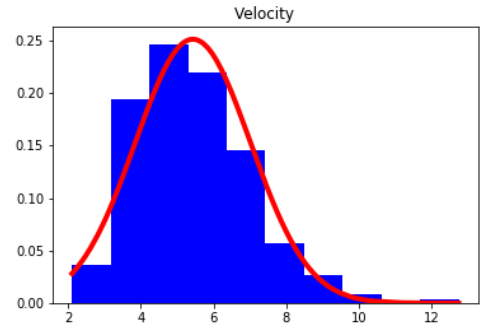
The most important and most challenging problem to tackle is further reducing the prediction error. Autonomous vehicles are performance constrained by limiting the maximum prediction error because, in practice, it is necessary to have a satisfying performance in safety-critical situations [18]. This can be done by increasing the amount of information in the dataset or increasing or decreasing model complexity, thus tackling the problem of underfitting[10] or overfitting[11]. Increasing or decreasing the model complexity doesn't increase the generalization performance of our model because the amount of information in the dataset limits the generalization performance. With this in mind, we propose adding additional information source such as output from a video camera.

Human driving relies on observing the environment and one of the most important human senses is the human vision. Obtaining an image associated with LiDAR measurements corresponding to the time of the visual observation would make a comprehensive dataset, which would further reduce the prediction error of the recurrent neural network.

[10]Insufficient model capacity with large training and generalization errors
[11]Model with low training error and high generalization error

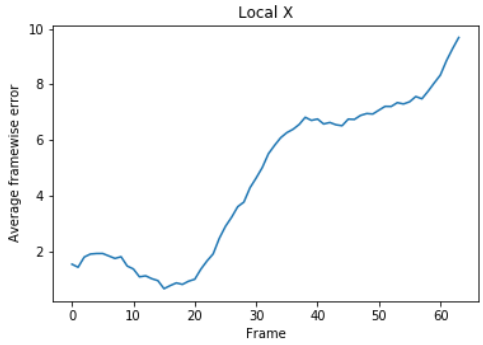


(a) Lateral position error

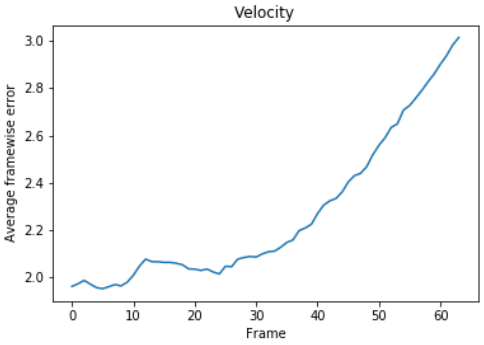


(b) Vehicle velocity error distribution

Fig. 10: Average frame error distribution on the sample of 10000 randomly chosen trajectories



(a) Average Per-Frame lateral position error



(b) Average Per-Frame velocity error

Fig. 12: Average Frame-wise prediction errors

For this approach, CNN[12] is imposed as the most natural approach. The approach with using CNNs and RNNs is used in [7] and [8]. A hybrid model consisting of convolution layers and recurrent layers. Convolution layers would extract features from the image, and downscale the image size. At this point, the convolution output is appended with LiDAR measurements, and together form an input to RNN, stacked onto the CNN. Together, CNN and RNN form a comprehensive hybrid model.

B. Hardware Acceleration

DNNs have high computational and storage requirements. Evaluating a large scale network is a computationally intensive task. Apart from constraining the accuracy, practical systems for behavioural prediction are constrained by the computational performance in terms of the time needed to generate a prediction. For example, predicting that a crash may occur is not useful if the crash has already occurred by the time of making a prediction. Therefore, it is necessary to reduce the computational delay in generating a prediction.

Hardware platforms for evaluating neural networks, such as CPU or GPU is inherently sequential or partially parallelized. Having in mind that the computations in neural networks are layer-wise independent of each other[13] further parallelization is possible, and furthermore, very desired. Unfortunately, such high level of parallelization is impossible for multicore processing units, such as CPU or GPU. Here, FPGA-based platforms enable designing custom tailored accelerators that would have higher throughput when evaluating neural networks. It would be interesting to design a custom IP core and compare its execution time when evaluating our network with the delay that is introduced when using a CPU or GPU.

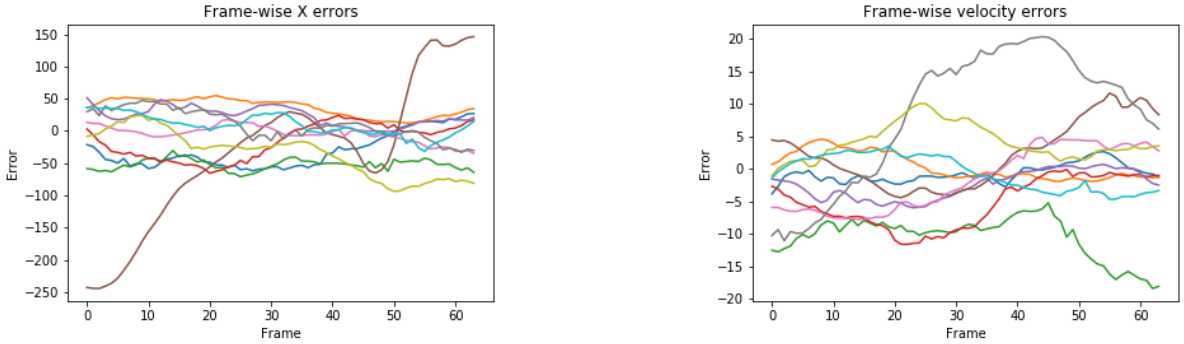
VII. Conclusion

Predicting the vehicle trajectory while driving on a highway can be seen as a *"hello world"*[14] of autonomous driving. Nevertheless, it is a fundamental step towards building more robust systems that perform on all large variety of roads and traffic patterns. During this project, we managed to develop a proof-of-concept prototype of the neural network capable to predict vehicle trajectories. Our long-term forecast is a sequence, consisting of 64 consecutive frames. In other words, we are able to predict a trajectory up to 6.4 seconds into the

[12]Convolutional Neural Network

[13]Calculating the output of one neuron is independent of all neuron outputs from the same payer.

[14]Highway driving exhibits a driving pattern with very simple characteristics. Therefore it is a straightforward way to learn the behaviour and achieve good performance results.



(a) Lateral position error

(b) Vehicle velocity error distribution

Fig. 14: Prediction errors for individual trajectories

future. Such long-term predictions are necessary for modern ADAS systems. Although we achieved a relatively large error of around 0.4m when predicting the lateral position and around 5km/h, we will focus our future work on further reducing the error as well. Also, we will try different approaches to solving the problem of trajectory predictions. The course of this project can be summarized with the following. Initially, the dataset was comprised of three sections that were merged together. Therefore, it was necessary to split the dataset into three distinct sections so that each vehicle ID corresponds to only one physical vehicle, instead of up to three. Afterwards, we derived the information about all surrounding vehicles for each vehicle in the dataset. Although it seems straightforward, we needed to detect doubled trajectories and to exclude them from the dataset. Having obtained the dataset in the required form, we filtered the data in order to reduce the amount of noise that is present, which was followed by merging all three sections with incrementally offsetting vehicle IDs in second and third section. Finally, split the dataset into the training set and test set and we developed a mechanism of providing the model with a train/test sequence, that will be used in the learning process. This resulted in the final form of our dataset. Afterwards, we developed the neural network model as an RNN with LSTM cells and trained it accordingly. In order to have a comprehensive evaluation metrics, we developed an evaluation framework with a purpose to track our progress and compare different network implementations.

VIII. Acknowledgment

We would like to thank Oliver Sawade and Eric Dörheit for their supervision, support and feedback. Further, we thank DCAITI[15] and TU Berlin for organizing this project.

References

[1] Florent Altché and Arnaud de La Fortelle. An LSTM network for highway trajectory prediction. *CoRR*, abs/1801.07962, 2018.

[2] Etienne Perot Senthil Yogamani Ahmad El Sallab, Mohammed Abdou. Deep reinforcement learning framework for autonomous driving. 2017.

[3] Thomas M. Mitchell. *Machine Learning*. McGraw-Hill, Inc., New York, NY, USA, 1 edition, 1997.

[4] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

[5] Andrew Simpson. Self-driving car steering angle prediction based on image recognition. 2017.

[6] Matt Vitelli and Aran Nayebi. Carma : A deep reinforcement learning approach to autonomous driving. 2016.

[7] Hesham M. Eraqi, Mohamed N. Moustafa, and Jens Honer. End-to-end deep learning for steering autonomous vehicles considering temporal dependencies. *CoRR*, abs/1710.03804, 2017.

[8] Andrew Simpson. Self-driving car steering angle prediction based on image recognition. 2017.

[9] SeongHyeon Park, Byeongdo Kim, Chang Mook Kang, Chung Choo Chung, and Jun Won Choi. Sequence-to-sequence prediction of vehicle trajectory via LSTM encoder-decoder architecture. *CoRR*, abs/1802.06338, 2018.

[10] Byeoungdo Kim, Chang Mook Kang, Seung-Hi Lee, Hyunmin Chae, Jaekyum Kim, Chung Choo Chung, and Jun Won Choi. Probabilistic vehicle trajectory prediction over occupancy grid map via recurrent neural network. *CoRR*, abs/1704.07049, 2017.

[11] A. Khosroshahi, E. Ohn-Bar, and M. M. Trivedi. Surround vehicles trajectory analysis with recurrent neural networks. In *2016 IEEE 19th International Conference on Intelligent Transportation Systems (ITSC)*, pages 2267–2272, Nov 2016.

[12] Martín Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G.S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[13] Wes McKinney. pandas: a foundational python library for data analysis and statistics.

[14] Ngsim. https://ops.fhwa.dot.gov/trafficanalysistools/ngsim.htm. Accessed: 2018-08-05.

[15] Us highway 101 dataset. https://www.fhwa.dot.gov/publications/research/operations/07030/. Accessed: 2018-08-05.

[16] Benjamin Coifman and Lizhe Li. A critical evaluation of the next generation simulation (ngsim) vehicle trajectory dataset. *Transportation Research Part B: Methodological*, 105:362 – 377, 2017.

[17] A. Savitzky and M. J. E. Golay. Smoothing and differentiation of data by simplified least squares procedures. *Analytical Chemistry*, 36:1627–1639, 1964.

[18] Who's responsible when an autonomous car crashes? https://money.cnn.com/2016/07/07/technology/tesla-liability-risk/index.html. Accessed: 2018-08-05.

[15]Daimler Center for Automotive IT Innovations