Self-driving Through a Deep Recurrent Convolutional Neural Model

Antonio Luna-Álvarez, Dante Mújica-Vargas

Tecnológico Nacional de México CENIDET, Departament of Computer Sciences , Cuernavaca, México

jesus.luna18ce@cenidet.edu.mx, dantemv@cenidet.edu.mx

Abstract. This paper presents...

Keywords. Deep Learning, Convolutional Model, Recurrent Model, Time-distributed, Self-driving, Autonomy.

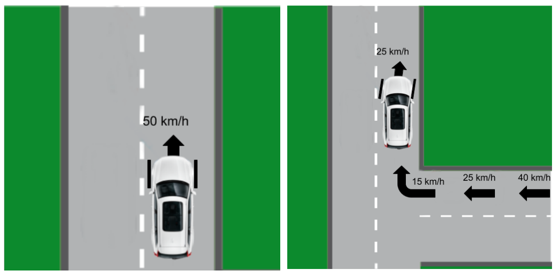
1 Introduction

Autonomous vehicles are robotic systems that must have the ability to navigate through the environment for which they were designed, independently and avoiding obstacles or situations that compromise the safety of the vehicle and its passengers, this task is named self-driving. The main component of a self-driving car is the decision module, which can be controlled by a classification model that will mainly predict the steering angle and vehicle speed.

Good results on self-driving have been documented using deep learning. Among the best are the research department of Nvidia, that proposes a convolutional deep model called Pilotnet [3][4] to provide autonomous driving in simulation and a physical system. Other authors have used this model to perform experiments on open source simulators [1].

There are also other deep neural models proposed with different architectures and paradigms, a reference is [9], which evaluates these models to detect prediction errors and possible collisions in static images. Using images of roads, compares two convolutional models like Pilotnet and a recurrent LSTM model getting better results with the last one.

Although satisfactory results have been reported, a deficiency has been detected in these. Particularly the driving task is an activity that depends on the actions taken in previous moments to take a new decision that controls the course of the vehicle. For example, a vehicle that just made a change of course. A reactive model such as those mentioned above, only receives the information of the current scene as shown in Fig. 1a, consequently it will provide a response that is valid for straight roads. However, a time-based model obtains information from previous scenarios such as Fig. 1b, thus allowing corrections and a response similar to that made by a human driver.



(a) Scene that sees a reactive model **(b)** Scene that sees a model based on time

Fig. 1. Difference between reactive and time-based models

In order to offer a solution to this deficiency, it is proposed to use a recurrent convolutional model that satisfies the need to make decitions based on time. Since recurrent networks have been used to identify relationships between succession of data [2], they are used to classify actions from a sequence of images processed by convolutional layers of the model as in other works. Based on the Chauffeur model [9], is adapted to receive sequences of images, thus creating time-distributed convolutional layers, and recurrent layers are used to make the clasifications based on the obtained information.

In the next section is presented the theoretical concepts necessary to understand the functioning of recurrent and convolutional neuronal models. Also in the section 3 the implementation made with the neural models and the Udacity simulator is shown. In the section 4 there is evidence of the results obtained, to finally give the final conclusions and acknowledgments in the last section.

2 Background

2.1 Recurrent Neural Network

Recurrent neural networks have been used in natural language processing to find relationships between word sequences and their dependence in the long and short term [2]. In the same way these networks are used to find relationships between the scenes shown.

There are three types of essential tasks that can be performed with this type of network [5]:

1. Sequence recognition: A particular output pattern occurs when an input sequence is specified.
2. Sequence playback: The network must be able to generate the rest of a sequence when you see part of it.
3. Temporal association: In this case a particular output sequence must be produced in response to a specific input sequence.

The Long Long-term Memory (LSTM) architecture was originally proposed by Hochreiter and Schmidhuber in 1997 [6], and is widely used today due to its superior performance in accurate modeling of short and long term dependencies.

An LSTM neuron is composed of 5 different non-linear components, which interact with each other in a particular way. In this paradigm, 4 components are called gates, they protect and control the information in the cell, which stores the memory of the neuron. The gates are implemented by a sigmoid function $\sigma(\cdot)$ and Hadamard product $\odot$. To control the behavior of each gate, one trains with a set of parameters and by calculating the gradient descending. The 5 gates of an LSTM cell are expressed as:

$$Forget : \sigma_f[t] = \sigma(\mathbf{W}_f x[t] + \mathbf{R}_f y[t-1] + b_f) \quad (1)$$

$$Candidate : \tilde{\mathbf{h}}_f[t] = g_1(\mathbf{W}_h x[t] + \mathbf{R}_h y[t-1] + b_h) \quad (2)$$

$$Update : \sigma_u[t] = \sigma(\mathbf{W}_u x[t] + \mathbf{R}_u y[t-1] + b_u) \quad (3)$$

$$Cell : \mathbf{h}[t] = \sigma_u[t] \odot \tilde{\mathbf{h}}[t] + \sigma_f[t] \odot \mathbf{h}[h-1] \quad (4)$$

$$Output\ g : \sigma_o[t] = \sigma(\mathbf{W}_o \mathbf{x}[t] + \mathbf{R}_o y[t-1] + \mathbf{b}_o) \quad (5)$$

The final classification is made by:

$$Output : \mathbf{y}[t] = \sigma_o[t] \odot g_2(\mathbf{h}[t]) \quad (6)$$

Where $x[t]$ is the input vector at time t. W_f, W_h, W_u and W_o are rectangular weight matrices that are applied to the input of the LSTM cell. R_f, R_h, R_u and R_o are square matrices that define the weights of the recurrent connections, while b_f, b_h, b_u and b_o are vectors of bias. The function $\sigma(\cdot)$ is sigmoid, while $g_1(\cdot)$ and $g_2(\cdot)$ are point nonlinear activation functions generally implemented as hyperbolic tangents that reduce the values in $[-1, 1]$. Finally, $\odot$ represents the inner product [2].

2.2 Convolutional Neural Network

This type of neural network has great advantages of automation in the task of extracting image information. In this paradigm, no known image processing filter is applied, but through the training of the neural network, it learns the image processing filters. The CNN is based on the convolution of signals in two dimensions and the detection of filter-based characteristics (called kernel) that they learn through training. To extract the characteristics, the kernel performs the 2D convolution operation, expressed in 7, to the input image.

$$y(n_1, n_2) = \sum_{k_2=0}^{N-1} \sum_{k_1=0}^{M-1} x(k_1, k_2)\, h(n_1 - k_1, n_2 - k_2) \quad (7)$$

Where N and M are the dimensions of the image, n_1 and n_2 represent the column and row indices of the pixel being processed, k_1 and k_2 the kernel indexes. These indices must meet the condition: $0 \leq n_1 \leq N-1,\ 0 \leq n_2 \leq M-1$ [7].

To train the convolutional layers, the backpropagation through the convolutional layer technique is used. This is very similar to backpropagation for a multilayer perceptron network, the only difference being that the weight connections are scattered, since the different input areas share the same weights to create a map of output characteristics. In general, the map of characteristics is obtained through the function expressed in 8. Where w represents the weights of the kernel and a first map of characteristics obtained.

$$S_{ij} = \sum_{n=1}^{2} \sum_{m=1}^{2} w_{(3-m)(3-n)} \cdot a_{(i-1+m)(j-1+m)} \quad (8)$$

For convolutional layers, the assigned activation function is the Rectified Linear Unit Function (ReLU) which is expressed in 9 where x represents the input value. This function allows normalizing the data by not admitting negative values and thus avoid noise that may contain the input image.

$$ReLU = ln(1 + e^x) \quad (9)$$

On the other hand, the kernel weights are adjusted by calculating the descending gradient which is expressed in 10. Where L represents the error function.

$$\frac{\partial L}{\partial w_{ij}} = \sum_{i=1}^{2} \sum_{j=1}^{2} \frac{\partial L}{S_{ij}} \frac{\partial S_{ij}}{\partial w_{ij}} \quad (10)$$

3 System Description

4 Experiments & Results

To compare the results,the NVIDIA Pilotnet model [3][4] was replicated shown in Fig. 2. This model is the greatest reference in the self-driving domain and unlike the Chauffeur model (Fig. **??**), it doesn't have convolutional layers of distributed time and its classification is based only on multilayer perceptron.

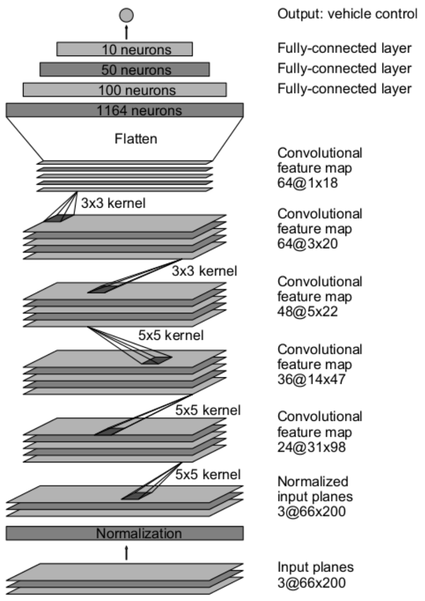


Fig. 2. Pilotnet model [3].

The tuning of parameters is the same for both models:

— Epochs: 10.

— Iterations: 10,000.

— Lot: 40 patterns.

— Learning rate: 1.0^{-5}.

— Training set: 80%.

— Validation set: 20%.

To compare the results with the Chauffeur model with different steps, it was trained with 3 and 5 steps using the same parameter configuration.

The training of a deep neuronal model requires great computational processing capacity, due to this the implementation was carried out with the support of special frameworks for the processing parallelization. The implementation was carried out with the help of the following software:

— OS: Xubuntu 14.04 Trusty Tahr.

- — Udacity Simulator precompiled.
- — Python 3.4.
- — Tensorflow 1.1.0 (gpu support).
- — Keras 2.1.2.
- — CUDA Driver 8.0.
- — cuDNN Driver 5.1.

In turn, this software requires high performance hardware, so a computer was used with the following characteristics:

- — CPU Intel Core 17-7700 3.60 GHz, 4 cores 8 threads.
- — 12 GB RAM.
- — 1 TB HDD.
- — 120 GB SSD.
- — GPU GeForce GTX TITAN X 3072 CUDA cores, 12 GB, 1000 MHz.

4.1 Metrics

Remembering that the problem have 87 classes, a difference of 0.01 in the prediction represents an absolute error in the metrics that are usually used for the evaluation of neural networks, such as precision, accuracy, etc. However, in the tests, a minimal error doesn't have such a drastic effect on driving, that is why this type of evaluation doesn't turn out to be ideal.

On the other hand, the metrics dedicated to measure the error in the prediction adapt much better to the problem and show the performance of these networks, since they calculate the difference between the expected and the obtained. A small angle of difference doesn't present as much error, that is why these are the ones that were contemplated to evaluate the training of the networks. The metrics used are: Mean Squared Error 11 and Mean Absolute Error 12.

$$MSE = \frac{1}{n}\sum_{i=1}^{n} = (\hat{Y_i} - Y_i)^2 \tag{11}$$

$$MAE = \frac{1}{n}\sum_{i=1}^{n} = |\hat{Y_i} - Y_i|^2 \tag{12}$$

In the study of the state of the art, few metrics related to the evaluation of autonomy were found. In [3], the metric shown in 13 is proposed, which is the most referenced in the literature.

$$autonomy = (1 - \frac{interventions \cdot 6}{elapsed\ time}) \cdot 100 \tag{13}$$

The metric is based mainly on the corrections made by the driver to avoid collision of the vehicle. This number of interventions is multiplied by a constant of 6 seconds, which represents the average time an average driver takes to correct the course of the vehicle, according to a study carried out by them.

Throughout the tests carried out in this work it was observed thatn't necessarily an intervention is usually 6 seconds in the simulation, even though it's in real time. Due to these variations, the metric proposed in the literature turns out to be less reliable, therefore it's proposed to measure the autonomy based on the intervention time of the driver to correct the address using the expression 14. This proposed metric evaluates the autonomy similarly to the previous one, with the difference that it takes an absolute time of interventions.

$$autonomy = (1 - \frac{total\ intervention\ time}{elapsed\ time}) \cdot 100 \tag{14}$$

As a third autonomy metric, it's proposed to perform the test without the driver's intervention to measure the percentage of the route that completes the vehicle on the runway until the first collision using the expression 15. In this way there's a reference on what each model He has learned from the road.

$$completed\ road = \frac{s \cdot t}{d} \cdot 100 \tag{15}$$

Where s is the average speed of the vehicle, t represents the driving time until the first collision or completing the route and d is the total distance of the test track.

Although there are metrics to evaluate the training quality of the neuronal model and the autonomy of the vehicle, during the experimentation it was observed that, although without colliding, the vehicle has to approach the ends of the road and change lanes in the form of zigzag. Because there is no collision, autonomy metrics do not penalize to the driving. Therefore, a metric based on the standard deviation 16 [8] is proposed.

$$s = \sqrt{\frac{\sum_{j=1}^{N}(x_j - \bar{x})^2}{N}} \tag{16}$$

This metric depends on a reference line previously made by a person. The standard deviation of the steering angle of the steering wheel made during driving is calculated, in the same way it is calculated for all the predictions of the neural model. Finally, the absolute difference of these deviations is calculated, obtaining an error in the conduction behavior in the function 17.

$$conduct = |s - s'| \tag{17}$$

4.2 Results

4.3 Result Discussion

5 Conclusion and Future Work

Conclusions here.

Acknowledgements

We would like to thank.. This work is funded by...

References

1. **Alexeev, V., Staravoitau, A., Piskun, G., & Likhacheuski, D. (2018).** End to end learning for a driving simulator. *Reports of the Belarusian State University of Informatics and Radioelectronics*, , No. 2 (112).

2. **Bianchi, F. M., Maiorino, E., Kampffmeyer, M. C., Rizzi, A., & Jenssen, R. (2017).** *Recurrent Neural Networks for Short-Term Load Forecasting: An Overview and Comparative Analysis*. Springer.

3. **Bojarski, M., Del Testa, D., Dworakowski, D., Firner, B., Flepp, B., Goyal, P., Jackel, L. D., Monfort, M., Muller, U., Zhang, J., et al. (2016).** End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316*.

4. **Bojarski, M., Yeres, P., Choromanska, A., Choromanski, K., Firner, B., Jackel, L., & Muller, U. (2017).** Explaining how a deep neural network trained with end-to-end learning steers a car. *arXiv preprint arXiv:1704.07911*.

5. **Bonet Cruz, I., Salazar Martínez, S., Rodríguez Abed, A., Grau Ábalo, R., & García Lorenzo, M. M. (2007).** Redes neuronales recurrentes para el análisis de secuencias. *Revista Cubana de Ciencias Informáticas*, Vol. 1, No. 4.

6. **Hochreiter, S. & Schmidhuber, J. (1997).** Long short-term memory. *Neural computation*, Vol. 9, No. 8, pp. 1735–1780.

7. **Pattanayak, S., Pattanayak, & John, S. (2017).** *Pro Deep Learning with TensorFlow*. Springer.

8. **Spiegel, M. R. (1991).** *Estadística*. McGraw-Hill Interamericana,.

9. **Tian, Y., Pei, K., Jana, S., & Ray, B. (2018).** Deeptest: Automated testing of deep-neural-network-driven autonomous cars. *Proceedings of the 40th International Conference on Software Engineering*, ACM, pp. 303–314.

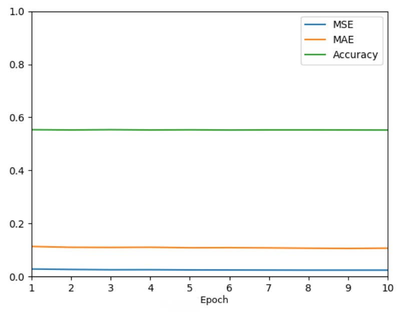
Article received on 06/12/2016; accepted on 16/01/2017.
Corresponding author is XXXXX.

Table 1. Metrics of autonomy obtained.

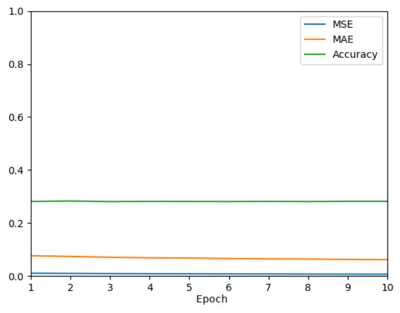
Model	**Metrics**			
	NVIDIA		Proporsal	
	Road 1	Road 2	Road 1	Road 2
Pilotnet (shortest db)	44.55%	83.11%	82.09%	96.35%
Pilotnet (largest db)	**100%**	77.57%	**100%**	90.05%
Chauffeur 3 steps (shortest db)	76.88%	85.96%	90.26%	92.48%
Chauffeur 3 steps (largest db)	**100%**	**100%**	**100%**	**100%**
Chauffeur 5 steps (shortest db)	88.89%	88.78%	95.43%	96.23%
Chauffeur 5 steps (largest db)	**100%**	**100%**	**100%**	**100%**

Table 2. Metrics of road completed obtained.

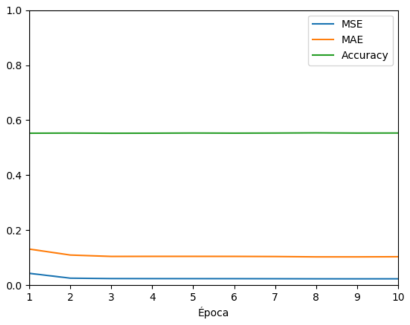
Model	**Road completed**	
	Road 1	Road 2
Pilotnet (shortest db)	9.47%	7.52%
Pilotnet (largest db)	**100%**	30.44%
Chauffeur 3 steps (shortest db)	33.61%	18.07%
Chauffeur 3 steps (largest db)	**100%**	**100%**
Chauffeur 5 steps (shortest db)	34.37%	16.93%
Chauffeur 5 steps (largest db)	**100%**	**100%**



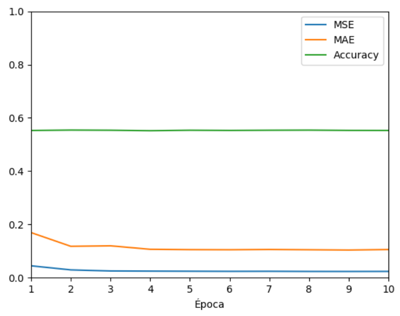
(a) Pilotnet, largest database.



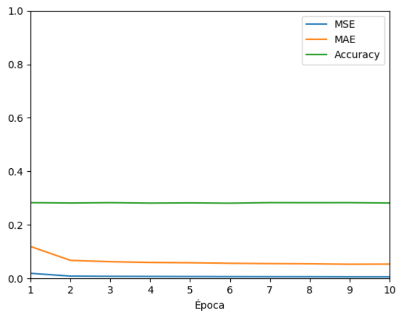
(b) Pilotnet, shorter database.



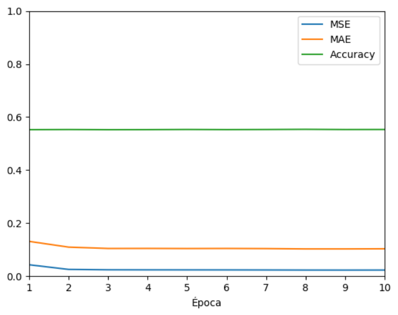
(c) Chauffeur 3 steps, largest database.



(d) Chauffeur 5 steps, largest database.



(e) Chauffeur 3 steps, shorter database.



(f) Chauffeur 5 steps, shorter database.

Fig. 3. Training metrics obtained.