Noise and Error Prediction for Neural Networks

D. Vrajitoru[1] **and K. Albelihi**[2]
[1]Department of Computer and Information Sciences, Indiana University South Bend, South Bend, IN, USA

Abstract – *In this paper, we are studying the application of a neural network to a problem related to autonomous car driving. In particular, we are interested in the correlation between noise in the data and the average error that can be achieved by the neural network, and study the issue using several data collections. We propose a measure of the noise that can indicate issues in the collected data and can help improve the learning process. We are also interested in ways to use symmetry in the data for this particular problem and analyze how we can take advantage of it for a better training of the network.*

Keywords: neural networks, noise measurement, symmetry

1 Introduction

In this paper we propose a study of the correlation between noise in the data and error in artificial neural networks (NN). For this, we use a problem where a NN is applied to predict the steering angle for an autonomous car driver. This problem presents an interesting challenge for the NN because of the size of the data - about 1000 data points after filtering - and because of the complex nature of car driving rules. We started from a previous research that was published in [1] and [2], and used the data collected for the Master's thesis in the experiments.

Autonomous car driving is a very relevant topic at the moment. Significant progress is made all the time and self-driving cars are currently being tested on the road. More and more, car companies provide assistive driving technologies, such as parallel parking and accident prevention. While it may be hard for academic research to keep up with the industrial pace, this still represents an interesting and challenging problem for us and an ideal test case for learning algorithms. Our interest in the subject is in seeing how well NNs can help with the driving problem.

Noise is a fascinating subject with many applications in computer science. In computer graphics in particular, noise generation plays a big role in procedural content generation for special effects and games. One of the popular methods is that of Perlin Noise [8]. It can be used to generate realistic-looking clouds, rocks, wood, and other textures. More complex methods such as the M-Noise [10] using several noise octaves were developed from it later.

Noise was also used in conjunction with the study of NNs and other related systems. For example, [5] proposes a method using NNs to reduce background noise for a better performance of speech-recognition systems. In [6], Bayesian networks are being used to analyze noisy biological data and uncover key biological features in cellular interaction. In [9], the authors use Short-Time Fourier Transform, self-organizing maps, and NNs to analyze the integrity of audio signals and their noise composition. Some studies [11] are concerned with developing NN-based systems that are robust with respect to the noise in the data. In [12], the authors use a NN to filter Monte Carlo noise from images.

In this paper, we continue the research started in [1, 2, 4] and examine the robustness of the NN when learning the data for this problem and how the variation of the average error with respect to the structure of the NN's hidden layers. Then we examine how to take advantage of symmetry in the data. Finally, we look at ways to visualize the amount of noise in the data and how the noise measure can be used as a tool to explain the average error of the NN.

2 Autonomous Car Driving and NN

The research in this paper is a continuation of the projects described in [1], [2], and [4]. These previous papers describe a Fuzzy-Logic car pilot called Epic that was capable of learning by hill-climbing techniques. Next, a system called Gazelle improved on various aspects of Epic, added more learning components, and implemented a unit to deal with opponents.

Moreover, in Gazelle we introduced the idea of collecting driving data from the procedural pilot and training a NN with it. The experiments showed that the road performance of the NN is not necessarily better than that of the procedural pilot, but it can improve some aspects such as road stability while reducing damage.

Both Gazelle and Epic drive a car procedurally in a car race simulation environment called TORCS. This software provides several pilots available for racing in a graphical environment and allows the user to compete with them. The user has the option to either by driving a car manually, or to program their own pilot and integrate it in the system. For the latter option, TORCS provides a `CarState` class with road information such as the free distance ahead, position of the car on the road, angle with the centerline, opponents present in a 100m radius around the car, and others. The program must

respond with changes to the steering wheel, the gas or brakes, and the gear. All of them are assembled into an object of the `CarControl` class, which is returned to the TORCS race server. Figure 1 shows a racing track available in TORCS and the controlled car driving on it.

Fig. 1. The Alpine2 track (left) and a snapshot of a driving car on it (right)

The procedural pilot that we wrote for Epic and subsequently improved for Gazelle contains several control units. We start by determining the target angle for steering, which is the most important element of driving, as it establishes the trajectory. An opponent modifier unit may change this target angle based on opponents' presence. Next, a target speed is chosen so that the car can achieve a turn by the target angle, while also trying to maximize the speed based on a given limit. This speed change is translated into acceleration or braking and can also be changed by the opponent modifier unit. The gear is adjusted in the last place.

These pilots contain other units to deal with special situations. A module is taking care of the car being outside of the road, and strives to get it back on track. Another module drives the car in reverse when it is stuck facing the wrong way on the road. A learning unit can adjust the speed from one lapse of the track to another based on the amount of damage experienced so far. Another learning unit remembers trouble spots such as sharp turns in the road so that the speed can be adjusted the next time we approach them.

The current research continues our previous work in several directions. First, we investigate the best settings for the NN that can allow it to learn the function governing the steering angle. Second, we look at a better way to exploit the available data by using properties of symmetry. Third, we analyze the connection between the noise in the data and the average error of the trained NN.

In the model we used to apply the NN to this problem, we chose 5 input variables that have significant influence on the steering decisions. The output value is the steering angle. In [2], data was collected by running the procedural driver on five tracks chosen from the available ones. A filtering process was also used to ensure a balanced distribution of the data in terms of output values.

Figure 2 shows an example of the filtering process: unfiltered and unbalanced data distribution in the upper part, and more balanced data distribution in the lower part. The intervals represent a discretization of the target output value in intervals of 0.1 length, going from -0.6 radians at the bottom to 0.6 radians at the top, and the number 8 marking a special interval (-0.01, 0.01]. The filtering process was intended to insure that the NN is not over-trained on some regions of the output target value range. For example many data points of the initial collection were very close to the value 0, because those would occur on any stretches of the road that are almost straight.

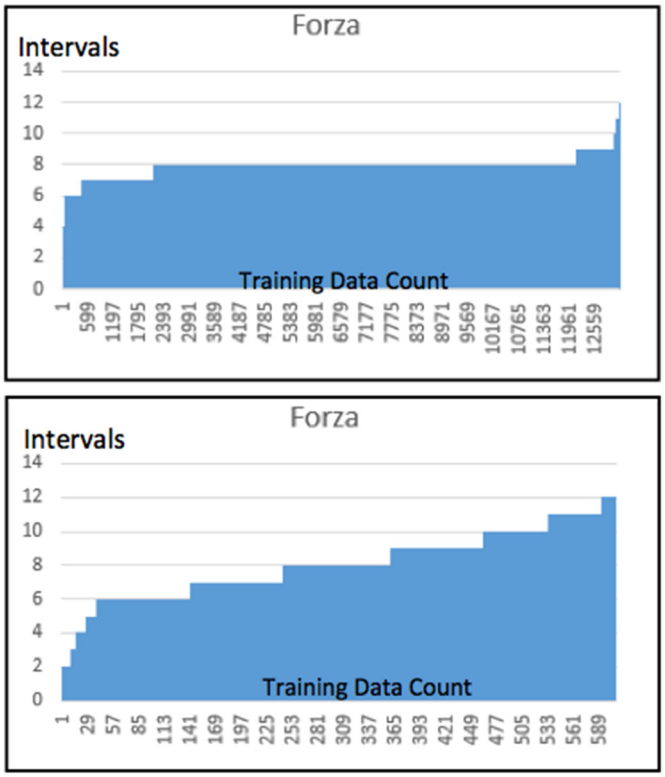

Fig. 2. Filtering the training data for the Forza track

Figure 3 shows a car with the road borders outlined and the input values that we have chosen to feed the NN, labeled from 1 to 5. The first, labeled 1, is the current angle of the car with the road centerline. Variable 2 is the lateral track position of the car and varies from -1 on the left border to 1 on the right border. Values outside of this range signify that the car has exited the road. Variable 3 represents the free distance ahead of the car. Variables 4 and 5 are the differences between the distance ahead and the free distance at 10^{o} angles left and right from the car direction. In this figure, the parameter 5 should have a negative value.

We selected three tracks that the data was collected for in [2] to perform our current experiments on, Alpine2, ETrack5, and Forza. We selected a fourth track, ERoad, to test the trained NN with new data that was not used for training.

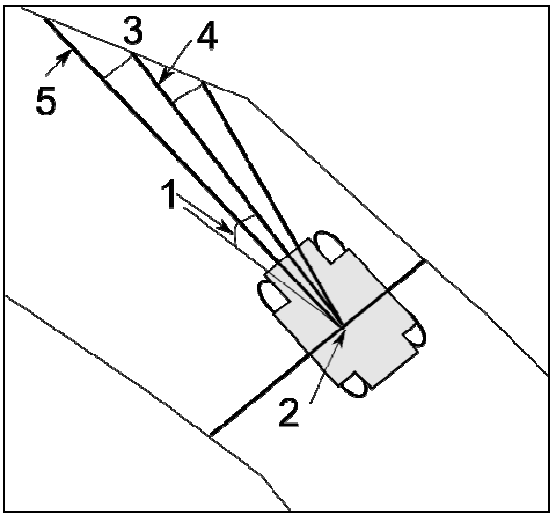

Fig. 3. Car state and five NN input variables

3 Using Problem Symmetry

Some functions by their very nature present some intrinsic symmetries. Depending on the way the data collection is done, these may or may not be present in the training data.

In our case the problem we are trying to solve is spatially symmetrical. Figure 4 shows a mirror of Figure 3 along the vertical axis. If the situation was thus reversed, the target output value for steering the car should be the negative of the target output value in Figure 3.

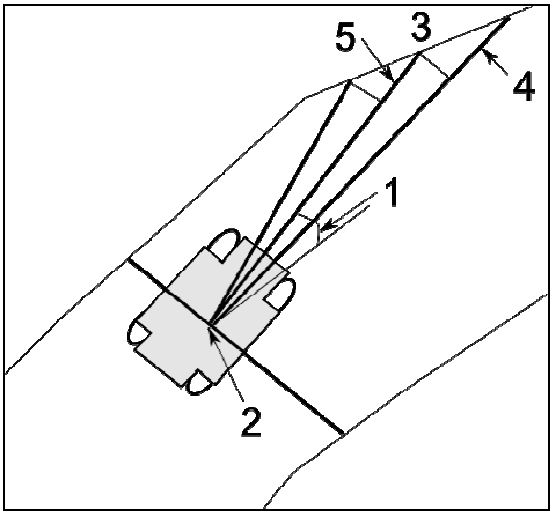

Fig. 4. Mirror situation of Figure 3

Thus, to mirror the problem, we need to multiply the first two variables and the target output value by -1. Then the values of the 4th and 5th variables must be swapped. The value of the third variable remains the same. By performing this operation on each data point that was collected, we can train the NN with data that accurately reflects the symmetrical nature of the problem, in hopes to achieve better driving performance.

4 NN Experimental Results

In this section we present the experimental results of training the NN with the data from each of the three tracks, with and without symmetry. For each track, we present the resulting average error both during the training process and for testing the trained NN on the data obtained from the fourth track, ERoad. The error is defined as the average difference between the NN's output and the target value for each data point, as follow:

$$error = \sqrt{\sum_{x\ in\ data}(output(x) - target(x))^2 / \#data}$$

Tables 1, 2, and 3 summarize the results for the tracks Alpine2, ETrack5, and Forza respectively. In all three tables, the training and testing results represent 1000 training iterations where the NN learns all the data points in the file by back-propagation. For all the tables, the testing data was obtained using data from the ERoad track. In all three tables, the NN contains four layers. The first one is the input layer and always contains 5 neurons. The number of layers in the second and third layers, denoted by HL1 and HL2 (for hidden layer), varies in the tables and is marked in the first two columns. The last layer is the output one and always contains a single neuron.

Table 1. Training and testing results for the Alpine2 data in 1000 iterations

		No Symmetry		Symmetry	
HL1	HL2	Train Error	Test Error	Train Error	Test Error
8	3	0.154	0.139	0.164	0.143
10	5	0.135	0.179	0.156	0.206
12	5	0.136	0.181	0.156	0.141
15	6	0.140	0.198	0.156	0.185

Table 2. Training and testing results for the ETrack5 data in 1000 iterations

		No Symmetry		Symmetry	
HL1	HL2	Train Error	Test Error	Train Error	Test Error
8	3	0.082	0.200	0.110	0.177
10	5	0.081	0.174	0.088	0.156
12	5	0.089	0.207	0.087	0.159
15	6	0.109	0.233	0.111	0.182

Table 3. Training and testing results for the Forza data in 1000 iterations

		No Symmetry		Symmetry	
HL1	HL2	Train Error	Test Error	Train Error	Test Error
8	3	0.046	0.149	0.042	0.145
10	5	0.044	0.154	0.045	0.149
12	5	0.044	0.161	0.049	0.142
15	6	0.057	0.165	0.046	0.143

From Table 1 we can see that for the Alpine2 track, the best configuration for the training data both with and without symmetry is that of 10 + 5 hidden neurons. For the test data, the best configuration is that of 8 + 3 hidden neurons without symmetry. The track ERoad is not a very symmetrical one in terms of turns, so for future research we will be looking for a track that has both left and right turns for more accurate testing.

Even though the ERoad track is not symmetrical, we can see that for a larger number of neurons, 12 and 15 on HL1, using data symmetry has resulted in a better average error on the test data.

From Table 2 we can see that for the ETrack5 data, the models of 10 + 5 and 12 + 5 neurons provide the best training error without and with the use of symmetry respectively. For the test data, the best result is shown by the 10 + 5 neurons configuration with the use of symmetry. In this case, using symmetry is consistently better for the test data. This can be explained by the fact that the ETrack5 track contains less turns in the road in both directions than the Alpine2 track.

It is interesting to note that the error in the test data is generally higher for the ETrack5 than it is for Alpine2, even though the error on the training data is lower. This can be explained by the fact that the Alpine2 track being more complex, it provides more opportunities for the NN to train well to calibrate its output for a larger variety of situations.

Looking at Table 3, it seems that the training error is substantially lower than for Alpine2 and for ETrack5. The best result in this case was obtained by the 8 + 3 neurons model with symmetry. For the test error the best result was obtained by the 12 + 5 neurons model, also with symmetry. Since this is a simpler track than the first two, the overall error is lower and the symmetry seems to help lower the error consistently, especially on the test data.

We can also wonder if more training iterations could have led to different results. Figures 5 and 6 show a plot of the different models we used over 5000 iterations for the ETrack5 data with symmetry for the training and testing cases respectively.

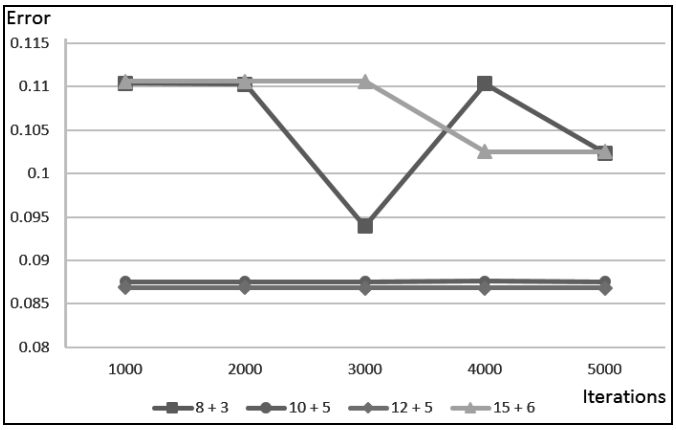

Fig. 5. Training error on the ETrack5 data with symmetry

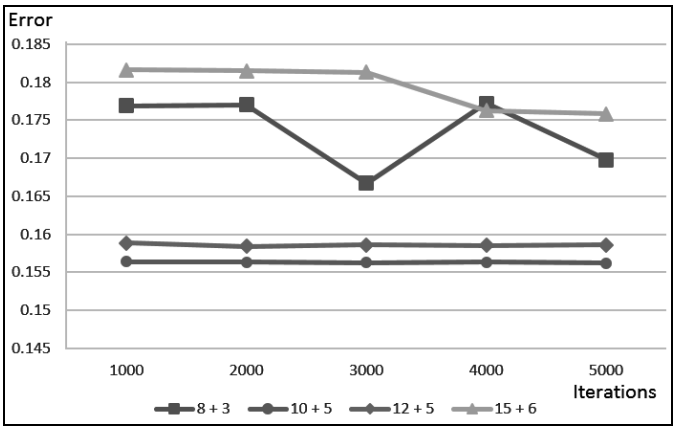

Fig. 6. Testing error on the ETrack5 data with symmetry

From these two figures we can see that for some of the models (10 + 5 and 12 + 5), training past the initial 1000 iterations led to very little change in the error both for training and for testing. For the 8 + 3 model, there is a substantial decrease of the error both in training and testing modes around 3000 iterations, followed by another increase. For the 15 + 6 model, the back-propagation seems to increase its efficiency after 3000 iterations and the error continues to decrease after that, although between 4000 and 5000 iterations the learning process slows down.

These figures show that smaller NNs are likely to learn faster, so 1000 iterations might be sufficient for the model 8 + 3 neurons. Larger NNs are likely to learn more slowly, and in this case a larger number of iterations must be tried to give the back-propagation algorithm a chance to work. However, the overall results do not suggest that larger networks always have a chance to achieve a better error than smaller ones, and experiments can help calibrate the appropriate size of the layers.

5 Noise Analysis

In this section we are analyzing the noise in the collected data for the three tracks we chose for training: Alpine2, ETrack5, and Forza.

5.1 Noise Measurement Function

Noise can be defined in a variety of ways. It is a fuzzy notion that indicates that adjacent data points in an area, mostly of visual or audio nature, differs in random, unpredictable, or unexpected ways. The idea of entropy is often connected to it, for which Boltzmann's and Gibb's equations are available as forms of measurement [3]. These equations are related to the noise measurements ideas of noise figure and noise factor used in acoustics and radio engineering. These do not directly apply to the analysis we are trying to perform. Usually logarithmical averaging methods are used for analyzing large collections of data, but those do not represent the picture we want to look at for our problem.

The idea in the noise measurement function we propose here is to be able to showcase how the amount of difference in the values of the output varies as a function of the amount of difference in the input. The kind of situation that is most interesting to us is where some input values are very close to each other, but the output values associated with them are quite different. This is likely to cause some difficulties for the NN in the training process. So we define here a function N connecting the difference in the input values to the difference in the output values.

Let x_1 and x_2 be two input data points or vectors. In our case, each of them contains 5 values for the 5 input parameters we chose for training the NN. Let $\Delta x = d(x_1, x_2)$ be the simple Euclidian distance between the two points. Let $y_1 = f(x_1)$ and $y_2=f(x_2)$ be the target output values for each of these input points, and $\Delta y = |y_1 - y_2|$. Then as a measure of noise in the data we chose to examine the shape of the two dimensional function

$$\Delta y = N(\Delta x) \qquad (1)$$

If the initial data collection contains 1000 points, then plotting the function N would result in a surface of 1,000,000 points. We decided to reduce the analysis to those points for which either Δx or Δy is less than $\varepsilon = 0.1$ and we sorted them by Δx. Figure 7 shows the function N plotted this way for the data collection Alpine2.

For a better understanding of this image, we plotted a second figure zooming in on the first part by looking at noise points for which $\Delta x <= 0.5$. Figure 8 shows this second plot.

For the ETrack5 and Forza data, the right side part of the plot is very similar to Alpine2 in Figure 7. Thus, we show here only the zoomed-in plots where $\Delta x <= 0.5$. Figure 9 shows the noise function for the ETrack5 file, while Figure 10 shows the plot for Forza.

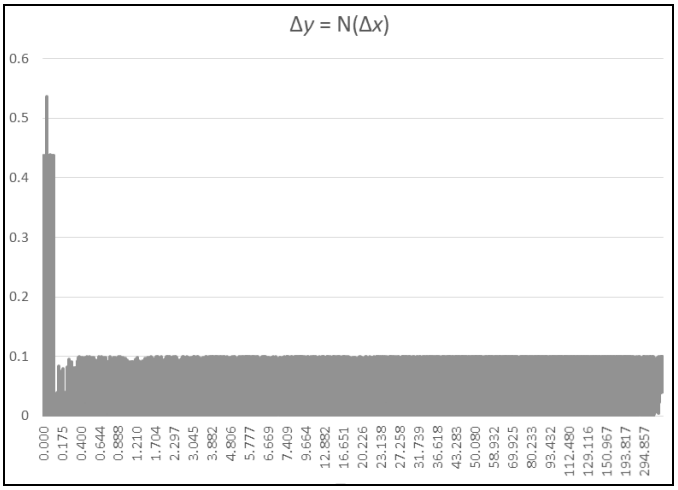

Fig. 7. Noise plot for Alpine2

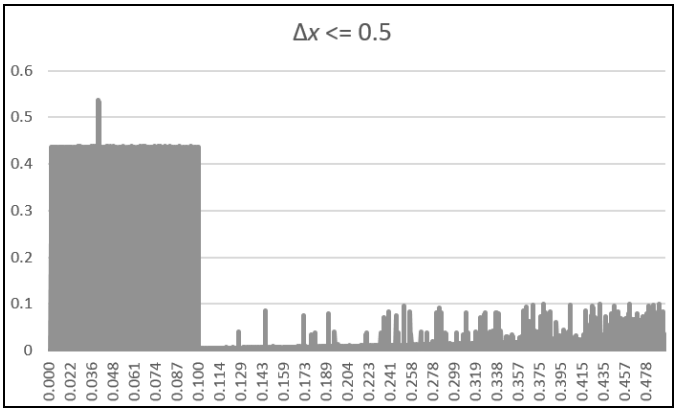

Fig. 8. Noise plot for Alpine2 with $\Delta x <= 0.5$

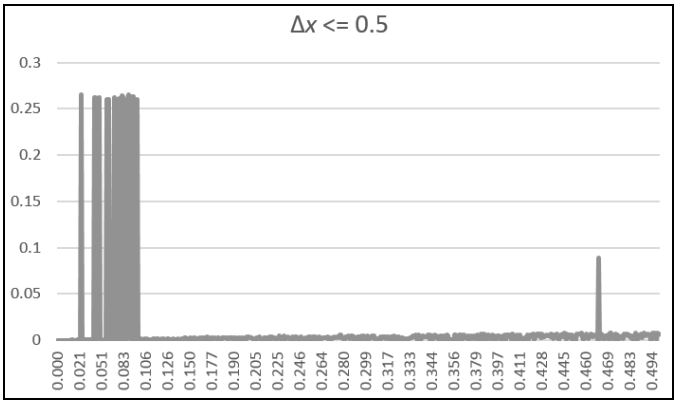

Fig. 9. Noise plot for ETrack5 with $\Delta x <= 0.5$

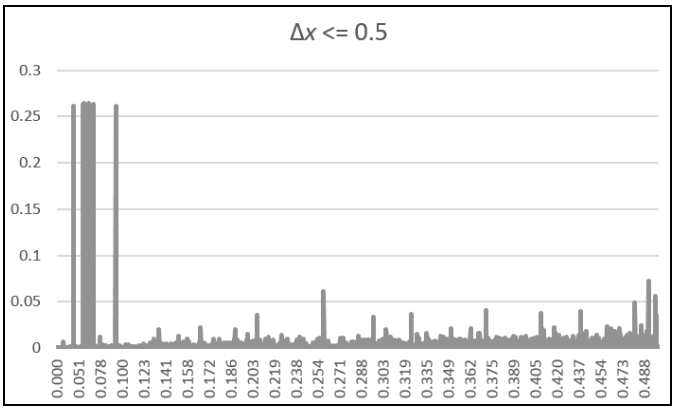

Fig. 10. Noise plot for Forza with $\Delta x <= 0.5$

5.2 Noise Analysis

Looking at Figure 7, we can distinguish two major artifacts, a visible spike on the left-hand side and a long plateau on the right-hand side. Figure 8 zooms in on the first part of the graph to show more details of this spike.

The interpretation of the spike is that we have input data points with a distance less than 0.1 for which the difference in the output is between 0.4 and 0.5. We can even spot some points closer than 0.02 with this kind of difference in the target output value. This spike will make it difficult for the NN to provide an accurate output value for all of these points, and will increase the value of the observed error.

Examining possible sources of this kind of spike, it is conceivable that the data are issued from a noisy source to begin with. If the data were collected from a human driving a car, various factors could have caused him or her to make different steering decisions in similar situations. The level of stress, fatigue, or distraction can be of influence and can cause slightly different reactions. Even when the data are collected from a procedural driver, the variety of algorithms used to address situations on the road can also cause the observed differences.

However, the spike can also mean that the data collection might have been incomplete. It is possible that external parameters that were not observed during the data collection also have an influence on the outcome and should be taken into consideration. If at all possible, the researcher can go back to the input parameters definition to see if more variables can be added. For this study, this is a possible future research track.

In our case, the noise analysis shows that the largest amount of noise, represented by the left-size spike in the three images, is present in the Alpine2 track, due to its complex nature. Looking back at the training error, this track resulted in the largest training error values of the three, between 0.13 and 0.15 in 1000 iterations. The second track in terms of noise is ETrack5, which also happens to be the median track in terms of training error, between 0.08 and 0.11. The track data collection showing the least amount of noise, Forza, is also the one with the smallest value of training error, between 0.04 and 0.06. However, in terms of test error, the best results were shown by the Alpine2 track, probably also due to its complex nature. From this analysis it seems pretty clear that there is a correlation between the amount of noise in the data, such as defined by the function N, and the average error of the trained NN. Thus, noise is a useful tool for understanding the capacity of learning of this method.

Let us examine the second artifact, the long plateau on the right side of Figure 7 for Alpine2. The same plateau is observed in the full plot for the two other tracks. This plateau means that we have data points that are increasingly distant - close to 300 for Alpine 2 - for which the difference in the target output value is less than or equal to 0.1. This occurrence in itself does not constitute a problem for the NN and is not likely to contribute to the average error. However, it can also indicate some improvements that can be made to the data collection. It could be caused by the nature of the function we are trying to learn, such as a periodical function, for example. However, another possibility is that some of the input variables chosen to feed the NN are redundant or do not contribute much to the output value. The researcher can examine the set of variables to see if some reduction of them is possible. Methods such as the ones proposed in [7] can be used for this purpose.

In our case, none of our variables are redundant. It is more likely that extra variables could provide more precision for the NN. However, some of our procedural driving algorithms have a Fuzzy Logic nature. For example, one of the rules says that if the free distance ahead is larger than a given threshold, we can consider the road to be almost straight and keep driving in the same direction. This means that in terms of this input variable, increasing its value above the threshold will not cause an observable difference in the output. This can be a possible cause for the observed plateau.

6 Conclusions

In this paper we presented an application of neural networks to a problem of autonomous car driver, in particular, to determining the steering angle. We used data collected on three tracks to train the NN and data collected from a fourth track for testing. We introduced an idea of using intrinsic symmetry of the problem to improve the NN training. We also proposed a method for visualizing noise issues in the data that can help with better understanding the results.

The experimental results presented in Section 4 show that using the problem's symmetry is beneficial to the training and testing results. The best configuration of the NN for our problem seems to be that of 10 neurons on the first hidden layer and 5 on the second. Even though the training error is generally lower for a simpler track such as Forza, the best error on the training data was obtained by using data from the Alpine2 track, which is more complex and challenging. Our

experiments also show that larger NNs do not always lead to better performance, although training them longer can improve their performance more than for smaller ones.

Section 5 introduced a noise measurement function and a method for visualizing the relevant part of it to study the noise present in the data. We have shown that data collections that present less noise are more likely to lead to lower average error for the NN. We also discussed how the analysis of the noise function can indicate better ways to collect the data for a problem. Thus, the noise analysis is an informative method for a learning algorithm.

7 References

[1] K. Albelihi, D. Vrajitoru, "An Application of Neural Networks to an Autonomous Car Driver", *The 17th International Conference on Artificial Intelligence*, July 27-30, Las Vegas, 716-722, 2015.

[2] K. Albelihi, "*The Gazelle Adaptive Racing Car Pilot*," Master's Thesis, Indiana University South Bend, 2014.

[3] E. T. Jaynes, "Gibbs vs Boltzmann entropies", *American Journal of Physics*, 33, 391-8, 1965.

[4] C. Guse and D. Vrajitoru, "The epic adaptive car pilot," in *Proceedings of the Midwest Artificial Intelligence and Cognitive Science Conference*, South Bend, IN, April 17-18 2010, 30–35.

[5] M. Trompf, "Building Blocks for a Neural Noise Reduction Network for Robust Speech Recognition", *Proceedings of EUSIPCA 1992*, Brussels, Belgium, Aug. 24-27, 1992.

[6] N. Friedman , M. Linial , and I. Nachman, "Using Bayesian networks to analyze expression data", *Journal of Computational Biology*, Vol. 7, 601-620, 2000.

[7] A. Hyvärinen, Survey on Independent Component Analysis, Neural Computing Surveys, Vol. 2, 94-128, 2001.

[8] K. Perlin, Ken, "An Image Synthesizer", *Proceedings of SIGGRAPH Computer Graphics*, 19 (0097-8930): 287-296, July 1985.

[9] W. Satney, A. Carrington, T. Scantlebury-Manning, and A. Als, "Determining Signal Source Integrity Using a Semi-supervised Pattern Classification System", *The 17th International Conference on Artificial Intelligence*, July 27-30, Las Vegas, 629-635, 2015.

[10] M. Olano, "Modified Noise for Evaluation on Graphics Hardware", *ACM SIGGRAPH/Eurographics Graphics Hardware*, 2006.

[11] B. Li and K. Chai Sim, "A spectral masking approach to noise-robust speech recognition using deep neural networks", *IEEE/ACM Transactions on Audio, Speech and Language Processing (TASLP)*, Volume 22 Issue 8, August 2014.

[12] N. Khademi Kalantari, S. Bako, and P. Sen, "A machine learning approach for filtering Monte Carlo noise", *ACM Transactions on Graphics (TOG) - Proceedings of ACM SIGGRAPH 2015*, Volume 34 Issue 4, August 2015.