

Learning to drive based on multiple sensor cues in The Open Racing Car Simulator (TORCS)

Tim Bicker and Nimar Blume

Abstract—To implement an autonomous driver in The Open Racing Car Simulator (TORCS) based on a deep neural network (DNN) and spiking neural network (SNN) multiple sensor cues are used. Specifically, the DNN predicts the current car displacement and angle to the road centre from a driver's view image. Based on the two values a SNN generates driving commands for the car. Subsequently, the car is put onto a new track and the driving performance is evaluated. The DNN is based on a Convolutional Neural Network and after training the mean absolute error for the displacement is XXXX and for the angle is XXX on an unseen test track.

Index Terms—deep learning, TORCS, convolutional neural network, spiking neural network, autonomous driving

I. INTRODUCTION

AUTONOMOUS driving is currently a controversial subject, especially regarding feasibility and performance. To explore the details of autonomous driving, an abstract prototype is build focussing on the implementation of deriving location data from images and using sensor inputs to drive a car. In this paper The Open Racing Car Simulator (TORCS) is used to simulate a car on a race track and collect relevant sensor data. The goal is to use multiple sensor cues to drive the car safely around the defined test tracks. Through the Robot Operating System (ROS) TODO: ADD CITATION the following sensors can be read from TORCS TODO: ADD ROS TORCS CITATION:

- 1) range finder
- 2) driver point of view (POV) image
- 3) displacement (distance from road centre to car centre)
- 4) angle (rotation angle between car and road)

Then, a deep neural network will be trained on the ground truth data to infer the angle and displacement from the provided POV image. Subsequently, a controller based on spiking neural networks (SNN) is trained to generate driving commands based on range finder, displacement and angle sensor inputs. Finally, the controller and the DNN are connected. So the overall model works only based on the POV image and the range finder data, which are directly taken from TORCS.

As a competing group is working on the same task, a race will be carried out at the end and the winner will be determined. Thus, the goal is to drive the car around the track as quickly as possible while maintaining a safe driving style.

Authors: Tim Bicker (12345678, tim.bicker@tum.de) and Nimar Blume (03638934, nimar.blume@tum.de) **Course:** Practical course: Computational Neuro Engineering Winter Semester 2017/2018 **Submitted:** January 4, 2018 **Supervisor:** Florian Mirus, Neuroscientific System Theory (Prof. Dr. Jörg Conradt), Technische Universität München, Arcisstraße 21, 80333 München, Germany.

II. A DEEP NEURAL NETWORK FOR REGRESSION

The implementation of the deep neural network is done in python version 3.6.3. To facilitate development, the library keras [5] is used with the backend tensorflow [3]. To manipulate image data openCV3 [4] is used and finally to load and manipulate data the library numpy [12] is utilised.

A. The testing deep neural network

To be able to chose hyperparameters for the convolutional neural network (CNN) at an early point, a basic CNN network architecture was chosen which has been proven before. The criteria for choosing the network architecture was, that it has to provide reasonable performance while being quick to train. The focus was rather on fast training performance, as the available resources are limited to us. Thus, after studying the architectures' training performances as seen here [6], AlexNet [8] was chosen as testing DNN.

AlexNet is a network designed for image classification tasks, such as the ImageNet challenge. Therefore, the last layer of AlexNet was altered to use it for multidimensional regression problems such as guessing the angle and displacement from an image. To achieve that, the number of output neurons of the last fully connected layer (FCL) was reduced from 1000 to 2, as there are two numbers to guess. Furthermore, the last FCL used a rectified linear unit (ReLU) as activation function.

$$f(x) = \max(x, 0) \quad (1)$$

To prevent the activation function from cropping the output to values greater than zero, a linear activation function is used instead: $f(x) = x$.

B. Data set splitting

TORCS provides 19 tracks of which all are relevant. Therefore, the recorded data set is split into three parts:

- 1) Training set
- 2) Validation set
- 3) Test set

First, the two tracks TODO:FIX TEST TRACKS were determined to be used for testing later on. As the goal is to train a general model, a prerequisite is that data recorded on either of the two test tracks is not used during training. In total XXX data points were recorded on the two test tracks.

The remaining tracks are used to train the DNN and to determine its hyperparameters. Therefore, the data set is randomly split into train and val (validation) at a ratio of 90% to 10%. The validation set data itself was thus not used for training, but it is recorded on tracks which are included in the training tracks.

C. Input images

The image data is acquired from TORCS [13] using ROS via the TORCS-ROS node [9].

1) *Choosing a camera angle:* TORCS provides several camera perspectives:

- 1) Driver's view with hood
- 2) Driver's view without hood
- 3) Third person perspective: far
- 4) Third person perspective: close

All perspectives are evaluated based on the DNN described in subsection II-A and as final perspective the driver's view without hood is chosen. The basis for that decision can be seen in Figure 1, which shows the mean absolute error for each view, with the driver's view without hood being the lowest. TODO: INCLUDE EXAMPLE IMAGES



Figure 1. Mean absolute error of the same DNN trained with multiple camera angles

2) *Choosing the image size:* The images are provided by TORCS ROS [9] at a rate of 10 frames per second (fps) at a resolution of 640 px \times 480 px. Because that image size is too large to train a reasonably deep network in a reasonable time, the images are down-scaled prior to training as well as in the final application. To determine the image providing the best result, four different sizes were evaluated with the DNN described in subsection II-A:

- 1) 320 px \times 240 px
- 2) 160 px \times 120 px
- 3) 80 px \times 60 px

First, the training images were collected from TORCS ROS at a resolution of 640 px \times 480 px. Upon loading the images, openCV based downscaling was applied using the bilinear interpolation algorithm. Second, the testing DNN was trained with the images at the mentioned resolutions and the mean absolute error for the validation set was recorded, which is visualised in Figure 2.

Therefore, the image size 80 px \times 60 px is used for the DNN as it provides the smallest mean absolute error and additionally reduces the training time due to the small image size.

D. Choosing the optimiser

A plethora of different optimisers are available for use in the backpropagation step when training DNNs. The largest differences are in their abilities to exit large local maxima, training performance and accuracy. As the currently used DNN is manageable in size, the training performance is not yet a critical property. Therefore, the optimiser was selected based on its ability to minimise the mean absolute error of the validation set. As illustrated in Figure 3, the adamax optimiser provides the best result and is thus chosen for future

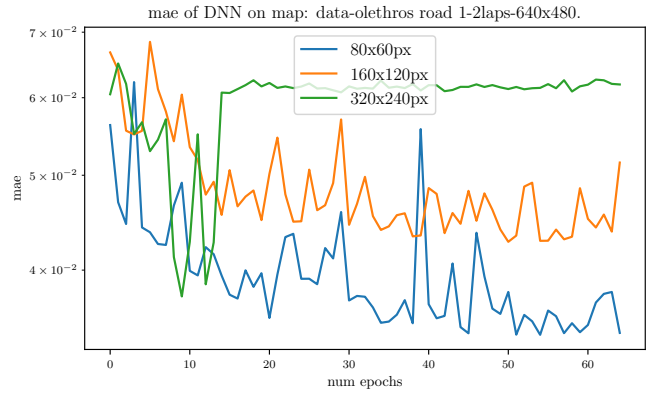


Figure 2. Mean absolute error of the same DNN trained with multiple image resolutions

application. adamax is similar to the adam optimiser while being more stable in certain conditions, see [7].

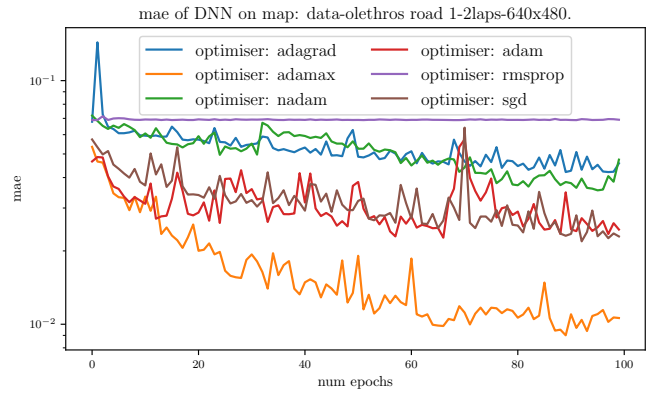


Figure 3. Optimiser comparison in the test DNN using with an image size of 80 px \times 60 px

E. Building a deep neural network with keras

Keras is a python library hello keras [11]

III. NENGO CONTROLLER

The controller of the car is designed based on Spiking Neural Networks (SNN) which are implemented with Nengo [2], a python framework for building large-scale neural systems that is based on the Neural Engineering Framework (NEF) [10]. The controller is divided in several modular parts: acceleration, braking, steering, gear changing and clutching. We choose the modular approach, because it allows to test different modules independently from each other and also mix hard coded solutions with learned ones. In order to discuss the architecture of the nengo model, we choose to first explain TODO: the setup?

a) *Signals:* The available input signals are as follows: The already mentioned lateral displacement, rotation angle relative to the street center line and the 19-dimensional range finder signals are used. Furthermore, we calculate the car speed

155 v as $v = \sqrt{v_x^2 + v_y^2}$, with v_x and v_y as the car speed in the
156 front and side directions relative to the car.

157 As control signals an acceleration, brake and clutch signal, as
158 well as a steering and a gear signal are given.

159 A. Controller Training

160 In this work a supervised learning approach for training
161 the controller is chosen. This leads to several variables, that
162 strongly influence controller performance. Those variables are
163 discussed in the following.

164 B. Data Sources

165 a) *TORCS-ROS default driver*: The first and easiest to
166 acquire is the default TORCS-ROS driver [9]. The advan-
167 tage of this driver is that he drives a maximum speed of
168 $149 \text{ km} \cdot \text{h}^{-1}$ and drives very carefully. Which means, that he
169 always tries to stick to the middle lane of the road and brakes
170 heavily when he comes close to a turn. We believe that this is
171 an easy to learn and robust driving style. However, this driver
172 is very slow, as can be seen in Table I.

173 b) *TORCS drivers*: Furthermore, there are several drivers
174 that are implemented in TORCS. Those drivers have no speed
175 limit and their driving style approximates that of a real race.
176 The cars drive very close to the edge of the road and drive
177 very fast through turns. We think that this is a hard to learn
178 and error-prone driving style, because small errors lead to
179 disastrous effects.

180 c) *driving manually*: Finally, there is the option to drive
181 manually. As it turns out this is really difficult with a keyboard.
182 Because fractions of seconds on the left and right arrows lead
183 to very strong steering behavior and almost every time lead to
184 a full turn-around or a far deviation of the track. This leads to
185 a manual driving style that comes close to the default driver:
186 very slow through turns and sticking close to the road center.
187 However, because it is possible to drive as fast as one wants
188 and in general one achieves a smoother driving of the car, we
189 achieve a faster time than the default driver.

Because of the above mentioned characteristics of the differ-

TORCS-ROS default	85s
TORCS	43s
manual	69s

Table I

THIS TABLE COMPARES THE LAP TIMES OF DIFFERENT DRIVER TYPES.

190 ent driver styles, it does make sense to first learn the TORCS-
191 ROS default driver, because he seems to be the most robust
192 one and data generation is simple.

194 C. learning methods

195 Besides the functionality of Nengo to directly encode and
196 decode hardcoded functions according to the NEF [10], Nengo
197 provides two different learning strategies: offline and online
198 learning. With Nengo deep learning [1], an extension to
199 implement classic deep learning methods exists as well.

200 a) *offline learning*: can be used for the classical super-
201 vised learning approach. Feature data and corresponding labels
202 are given and Nengo uses a least squares approximation to
203 solve for the neuron weights [10].

204 b) *online learning*: iteratively improves the initial neuron
205 configuration in a supervised manner during runtime. How-
206 ever, this requires that for every timestep during runtime a
207 goal value has to be known so Nengo can solve for it. In our
208 problem this is not the case. Therefore online learning is not
209 further considered in this work.

210 c) *Nengo deep learning*: allows to train classical deep
211 learning methods e.g. feedforward neural networks with back-
212 propagation and then implement the learned nets into Nengo.
213 This work focuses on a mixture between hard coded function-
214 ality and offline learning.

215 D. Evaluation metric

216 As a evaluation metric for the controller performance the
217 following is used: First, it is evaluated if the controller is able
218 to finish one lap in a reasonable amount of time. Second, the
219 time necessary to complete one lap is considered. Both criteria
220 are first evaluated on the trained tracks and in case the first
221 criteria is successful, it is then measured how well the driver
222 generalizes to unseen tracks.

223 E. Nengo design

224 Since we have the possibility to look in the source code
225 of the default driver, we observe that the different modules
226 drive based on different sensor inputs. The steering module is
227 a function based on the car speed v_x , the lateral displacement
228 d_{lat} and the rotation angle α . Both, acceleration and
229 braking module are based on three range sensor signals -5 ,
230 0 and 5 degrees TODO: insert degrees symbol' as well as v_x .
231 We chose to implement the gear changing as a hard coded
232 solution without any learning involved, since it is a simple if
233 else condition. It is further possible to neglect the clutching
234 module, because it did not show to make any difference
235 for driver performance even though it is implemented in the
236 default driver.

237 IV. EXPERIMENTS AND EVALUATION

238 A. Performance comparison between guessing angles vs. 239 guessing displacement

240 For the final application, the same model is used to guess
241 the car's displacement and its angle. During training, the
242 loss function (MSE) and the metrics (MAE) are continueally
243 evaluating the consolidated training performance. However, to
244 determine the performance of predicting the individual values,
245 two models were trained on each either predicting the angle
246 or predicting the displacement of the car. Figure 4 shows
247 the difference between angle and displacement prediction
248 is shown. In training and validation set, the value of the
249 displacement is in the range of -0.9 to 1.1 and the angle is in
250 the range of -0.9 to 1.1 . Thus, it can be concluded that the
251 displacement prediction performance especially still needs to
252 be worked on.

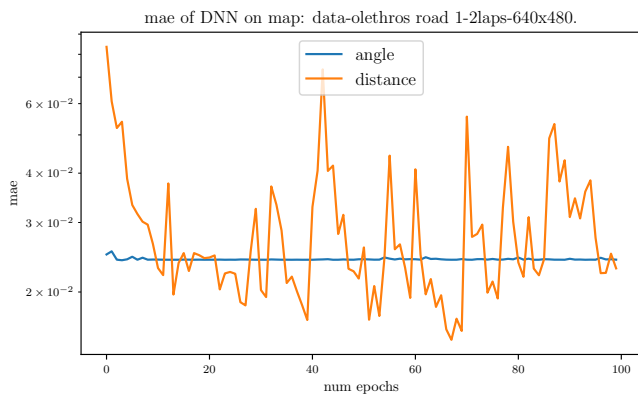


Figure 4. Mean absolute error of two test networks trained only to predict either the displacement or angle of the car

B. Controller performance

For finetuning the results we have basically three tuning possibilities: the neuron sizes, the amount of data and the composition of tracks the data was collected from.

The neuron size..

amount of data ...

track composition..

Final generalisation times..

C. Controller and DNN

V. CONCLUSION

A. Outlook

The network architecture can perhaps still be significantly improved upon, especially with an architecture specifically designed for regression tasks. The controller needs to be improved further and looked at more closely. Finally, the DNN and the controller will be connected in order to work together.

REFERENCES

- [1] Deep learning integratino for nengo. Accessed: 2017-12-13. 3
- [2] The nengo neural simulator. Accessed: 2017-12-10. 2
- [3] M. 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. 1
- [4] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000. 1
- [5] F. Chollet et al. Keras. <https://github.com/fchollet/keras>, 2015. 1
- [6] J. Johnson. Cnn architecture benchmarks. <https://github.com/jcjohnson/cnn-benchmarks>, 2017. 1
- [7] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014. 2
- [8] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012. 1
- [9] F. Mirus. torcs ros. https://github.com/fmirus/torcs_ros, 2017. 2, 3
- [10] T. C. Stewart. A technical overview of the neural engineering framework. Technical report, Centre for Theoretical Neuroscience, 2012. 2, 3
- [11] A. Toshev and C. Szegedy. DeepPose: Human pose estimation via deep neural networks. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2014. 2

- [12] S. van der Walt, S. C. Colbert, and G. Varoquaux. The numpy array: A structure for efficient numerical computation. *Computing in Science Engineering*, 13(2):22–30, March 2011. 1
- [13] B. Wymann, E. Espié, C. Guionneau, C. Dimitrakakis, R. Coulom, and A. Sumner. TORCS, The Open Racing Car Simulator. <http://www.torcs.org>, 2016. 2