Imitációs tanulás a DuckieTown környezetben DAgger felhasználásával

Csapat:

Kovács Boldizsár - GVJY8E

Schneider Marcell - DBGYVI

Talpos Norbert - Q2H4XB

Virág József Ádám - U7KC0P

Imitation Learning in the DuckieTown environment with DAgger

Team:

Kovács Boldizsár - GVJY8E

Schneider Marcell - DBGYVI

Talpos Norbert - Q2H4XB

Virág József Ádám - U7KC0P

Kivonat

2021-ben alig-alig lehet olyan mérnöki területet találni, amin a deep learning a megszokott paradigmákkal szembe menve ne tudna javulást elérni. Az egyik legjellemzőbb ilyen terület az autóipar. A számítási teljesítmény növekedése és a deep learning fejlődése révén lehetőség nyílt az önvezető autók rohamos fejlődésére. Ezek fejlesztésében alapvetően két fő irányzat létezik: a Reinforcement Learning, amikor a modell egy szimulált környezetben tanul a reward function-t próbálja maximalizálni, illetve az Imitation Learning, amikor egy expert viselkedését próbálja megtanulni, és azt általánosítani. Ezutóbbihoz szükség van a szimulált környezetben az expertre, illetve szükséges kiegészíteni a DAgger algoritmussal, ami azt teszi lehetővé, hogy a modell olyan helyzeteket is megtanulhasson az experttől, amiket egyébként nem látna tőle (például hogyan kell visszajönni az útra). A házi feladatunkban az Imitation Learning+DAgger algoritmust dolgoztuk ki a DuckieTown környezetben.

Abstract

Nowadays one can hardly find a field in engineering where deep learning couldn't make an improvement opposed to the regular techniques. One of the most well-known field is the automotive industry. The increase in computation power and the advances in deep learning made possible a breakthrough in making self-driving cars. There are two main branches of development in this field: Reinforcement Learning, in which the model learns by trying to optimize a reward function in the simulation environment, and Imitation Learning, whose goal is to try to clone the behaviour of an expert in the environment. To make Imitation Learning a viable alternative, we need to complement it with the DAgger algorithm, which makes it possible to not only teach the model the optimal situations the expert would normally encounter, but the recoveries too. In this homework we implemented the Imitation Learning+DAgger algorithm in the DuckieTown environment.

Introduction

In our project, we worked in a Duckietown environment to construct an AI model in the field of autonomous driving and imitation learning. At first, we preprocessed images by cropping and resizing them and then we constructed a model consisting of three convolutional layers. The parameters of the layers have benn hyperparameter-optimized and we also used the Dagger-algorithm (an algorithm highly recommended for imitational training) to train our model through a Dagger expert (teacher). Our model will be able to determine the ideal steering (in radian) and speed (e.g. a two-element vector as the output) of the driven car according to the current observed image.

Review of field and previous solutions

There are various research projects conducted in the field of autonomous driving. Some of them include the thrilling environment of DuckieTown. Our project is set in the DuckieTown environment and make use of the imitation learning technique.

The field of imitation learning means learning algorithms designed to observe proper operation of functioning and thus educate the model. In other words, the imitation learning means the presence of a supervised agent providing the learner with data.

In the field of our project, the teaching means showing our model some training data consisting of proper driving pictures and corresponding driving actions, and then the main objective of our model is to predict an appropriate driving action.

The utilised methodology of our project contains the DAgger (data aggregation) algorithm. It trains the model on all the previous data collected by the agent. The driving data is created by the DAgger-expert. Of course, there are some other algorithms like Behavioral Cloning, GAIL and DAgger-extensions such as Ensemble DAgger and Reward-Guided DAgger.

During our examination of self-driving cars in DuckieTown environment, enormous advantage had been taken of a Scientific Students' Associations (TDK) research paper written by Zoltán Lőrincz, so that article is to be considered to be of paramount importance of the project.

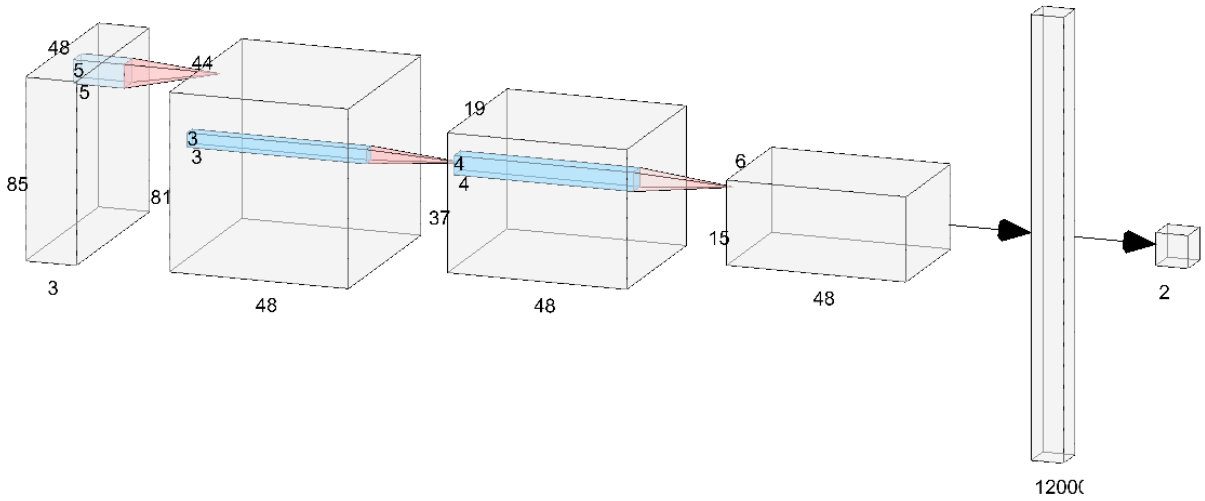


Figure 1: The architecture of the network

Network Architecture

We use a 2d convolutional network with 3 convolutional layers, Leaky ReLU activation function, batch normalization, and max pooling. After that we flatten the output from the last convolutional layer and put the inputs through a dense layer with 12000 neurons. This layer uses ReLU activation, regularization and dropout. The output is a 2 element vector, whose entries are the velocity and the steering angle.

Obtaining and preprocessing data

For the training, data were obtained in two ways. We used the DuckieTown environment's auto driver/expert, and we also drove a car ourselves using a keyboard. From these drives we extracted 3 things at each step. A picture, and the forward and sideways acceleration of the car. With this data we planned to teach.

Most of the information in the pictures was not considered useful, so we processed it in advance to make the teaching easier to work with. We used HSV image processing, which allowed us to separate the different colours from each other, and to minimise noise in the process. This made the asphalt almost completely black, as well as the lane divider yellow, and the road edge grey.

In addition, we have cropped the top of the images because we didn't think the sky was important, and we have also reduced the size of the images because they show the terrain well in a smaller image, but this makes teaching much faster as the size of the images is

reduced.

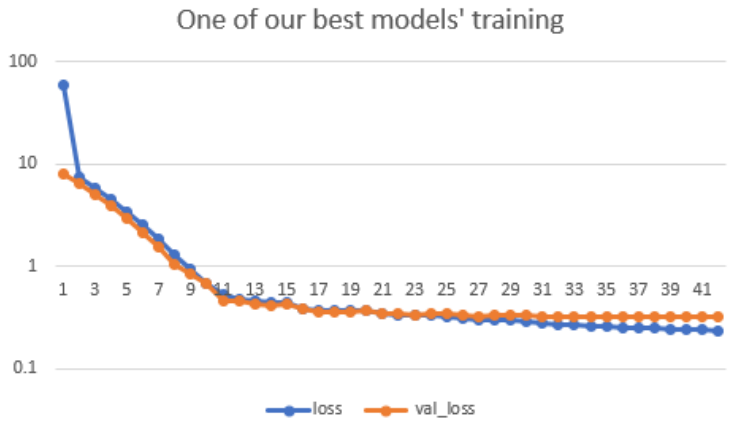
This also helped the scanning of the data, as tens of thousands of images take up less than 40 MB.

We made a level editor, as we need different tracks to get a variety of images. Since we planned to make a lot of tracks, it was faster and more manageable to do this. It became easy to use, although not entirely ergonomic. The way it works is that you can use the arrow keys to increase or decrease the size of the map.The T button lets you step through which track element to drop. The T button lets you step through which track element to drop. R button to rotate them 90 degrees. Left-clicking on any field will place that block where you clicked, and right-clicking will convert the block in that location back to a field. Pressing the D key switches the editor to dropping duckies. You can drop them anywhere you like by clicking somewhere with the left mouse button and rotating the scroll wheel before you do so. If you want to delete the last duckie you put down, you can do this with the right mouse button. When the track is finished, you have to press the S key, then the track name you entered in the console will save a .yaml file, which can be imported one by one into the Duckie environment. We created 40+ tracks using this method. Each one is different from the other. The names of the tracks from 10 to 49 indicate their difficulty and complexity. To begin with, there are relatively no two curved track elements in consecutive order, and no two straight paths side by side. For the forties, this leads to a zig-zag route, preferably as compact as possible. These were hoped to produce a variety of images.

Implementation

Traning and Testing

Several challenges were met during training. The traning part of the pipline is quite long. First, in the above described architecture we wanted to get the learning rate right. It was done by some manual hpyerparameter optimization across multiple training runs with a small sample of 5000 images. It was really helpful to get a grasp of approximately where to place the learning rates' value. We used different flat values and then tried to tune them using a learning rate Scheduler. With the scheduler down below we managed to get down to 0.3 validation MSE loss which is pictured in the diagram.



After realizing that the optimal learning is probably well described by the scheduler we proceeded to turn our focus on the other, architectural hyperparameters. We decided to run the optimization on the sizes of the convolutional filters from 16 to 96 with the step of 16 the sizes of the convolutional kernels from the options of 3 / 5 / 7 On the second dropout rate in the upper layers from 0.1 to 0.9 with the step of 0.2 The learning rate from the options of 0.01/0.001/0.0001,0.0001 to double check our previous assumptions on the learning rate After running the optimizaion for 196 trials we got the following values

Name of the hyperparameter	Value
conv 1 filter	80
conv 1 kernel	7
conv 2 filter	64
conv 2 kernel	7
conv 3 filter	48
conv 3 kernel	7
dropout	0.3
learning rate	0.001

With these values in hand we trained the model on the prepared data. We managed to hit a validation loss of below 0.3 by the end of training but our best working model in the environment has a validation loss of 1.01 for. **For the environment we found that if we multiply the predicted acceleration by 1.6 and the steering by 10.0 the models survivability greatly improves.** For testing we generated some more images with the expert agent and evaluated the model on these images. We got the following results:

	zigzag dists	small loop	canyon
Base MSE for velocity	1.01	1.01	1.02
Base MSE for steering	1.01	1.00	1.02
Adjusted MSE for velocity	1.02	1.00	1.01
Adjusted MSE for velocity	2.02	1.73	2.22

As for the live testing of our agent in the DuckieTown environment, we tested the model on numerous maps. The maps and the results on the map:

Zigzag dists	Our best model completes a full circle on this map
Small loop	Our model goes off track after completing numerous corners
Canyon	Our model manages to follow the lane. The corners are a little tight, but stays on track

Future plans

We encountered a multitude of minor major problems throughout the whole exercise. Initially we wanted to have one of our own from the DuckieTown environment. This took us a while, because we started from the point where we needed a VM and Ubuntu. We installed a bunch of necessary libraries, and then hoped that everything worked out and we could see some kind of simulation. We eventually did, and we started to get to know the environment. We arrived at our solution in a rather bumpy way, but in the end it was the best we could do. We tried to use the lesson materials as best we could, and also looked on the internet for solutions to some problems. Basically, we were very interested in solving each problem. We tried to solve them as best we could, while learning from them. We are relatively satisfied with the results, although we have had a million better and worse sub-results along the way. A model that could only drive well on occasion, but only left and forward. So after each failure, we tried to fine-tune our solution, and checked the code to see if we had mistyped something, or not written it at all.We tried to make the images as varied as possible, and we tried to modify the architecture. We did this with more and less success. It was fun working on this project, we put a lot of hours and effort in it. We have learned that it is possible to get things wrong even on the smallest things. We got on very well as a team, everyone did their part and did it fairly. With our experience, we are guaranteed to simplify future problems in many areas of life. It is a very useful skill to have. Whether it's a self-driving car, or just a simple event in a simulation environment, wherever something rewarding comes up, it's easy to help.

References

Duckietown official website:
https://www.duckietown.org/

Duckietown baseline DAgger algorithm.
https://docs.duckietown.org/DT19/AIDO/out/embodied_il_sim_dagger.html

Duckietown-world.
https://github.com/duckietown/duckietown-world

Ian Goodfellow and Yoshua Bengio and Aaron Courville, „Deep Learning", MIT Press, 2016

Chigozie Enyinna Nwankpa, Winifred Ijomah, Anthony Gachagan, and Stephen Marshall, "Activation Functions: Comparison of Trends in Practice and Research for Deep Learning", CoRR, arXiv: 1811.03378, pp. 8, 2018 Wu Jianxin. "Introduction to convolutional neural networks.", National Key Lab for Novel Software Technology, pp. 11-23., 2017

Keras official website:
https://keras.io/

Keras Tuner tutorial:
https://www.tensorflow.org/tutorials/keras/keras_tuner

Deep Learning Wizard:
https://www.deeplearningwizard.com/

Zoltán Lőrincz, Imitation Learning in the DuckieTown Environment
http://tdk.bme.hu/VIK/DownloadPaper/Imitacios-tanulas-a-Duckietown-kornyezetben