Meta learning Framework for Automated Driving

Ahmad El Sallab [1] **Mahmoud Saeed** [*2] **Omar Abdel Tawab** [*3] **Mohammed Abdou** [4]

Abstract

The success of automated driving deployment is highly depending on the ability to develop an efficient and safe driving policy. The problem is well formulated under the framework of optimal control as a cost optimization problem. Model based solutions using traditional planning are efficient, but require the knowledge of the environment model. On the other hand, model free solutions suffer sample inefficiency and require too many interactions with the environment, which is infeasible in practice. Methods under the Reinforcement Learning framework usually require the notion of a reward function, which is not available in the real world. Imitation learning helps in improving sample efficiency by introducing prior knowledge obtained from the demonstrated behavior, on the risk of exact behavior cloning without generalizing to unseen environments. In this paper we propose a Meta learning framework, based on data set aggregation, to improve generalization of imitation learning algorithms. Under the proposed framework, we propose MetaDAgger, a novel algorithm which tackles the generalization issues in traditional imitation learning. We use The Open Race Car Simulator (TORCS) to test our algorithm. Results on unseen test tracks show significant improvement over traditional imitation learning algorithms, improving the learning time and sample efficiency in the same time. The results are also supported by visualization of the learnt features to prove generalization of the captured details.

[*]Equal contribution [1]Ahmad El Sallab is a Senior Expert at Valeo, ahmad.el-sallab@valeo.com [2]Mahmoud Saeed is an Intern at Valeo, mahmoud.saeed.ext@valeo.com [3]Omar Abdel Tawab is an Intern at Valeo, omar.abdeltawab.ext@valeo.com [4]Mohammed Abdou is a Researcher at Valeo, mohammed.abdou@valeo.com. Correspondence to: Ahmad El Sallab <ahmad.el-sallab@valeo.com>.

Proceedings of the 34^{th} International Conference on Machine Learning, Sydney, Australia, PMLR 70, 2017.

1. Introduction

Automated driving development has radically changed during the past few years, driven by advances in Artificial Intelligence (AI), and specifically Deep Learning (DL). Developing an efficient and safe driving policy is in the heart of reaching high level of autonomy in a robot car. Traditional methods are driven with rule based approaches (Le-Anh & De Koster, 2006)(Pasquier et al., 2001), while significant advancement in the field is driven by learning approaches (Sallab et al., 2016)(El Sallab et al., 2017) (Bojarski et al., 2016). Reinforcement learning is the arm of AI which is concerned with solving the control problem based on learning while interacting with the environment.

Our motivation is to take the work in (El Sallab et al., 2017) and (Sallab et al., 2016) a step further towards real car deployment. The constraints of such task are: 1) No damage caused by interaction with real environment (which was possible in the game engines world), 2) Sample efficiency, so learning time has to be reasonable 3) Generalization, where the learnt policy of the driving agent should be able to capture the basics of the intuitive driving when deployed in unseen environments.

There are several approaches to find an efficient driving policy. Optimal control methods based on traditional planning require knowledge of an environment model. While model based approaches are gaining more popularity (Polydoros & Nalpantidis, 2017), it is still hard to develop an efficient environment model especially for complex ones like urban and city scenarios. On the other hand, model free approaches (Watkins, 1989) (Sallab et al., 2016) require the definition of a reward function, which only exist in simulated environments and game engines but not in real world. Imitation learning is another successful approach, which might suffer the risk of de-generalization to unseen environments other than trained on. The most popular algorithms are based on data aggregation, which could be done through some hacks as in (Bojarski et al., 2016), or formally as in (Ross et al., 2011).

In this paper we propose a Meta learning framework for automated driving, to enforce transfer learning from one environment to another. Our hypothesis is that Meta learning will help capturing generic features without memorizing one specific environment. One proof of generalization

is the visualization of which features the agent has learnt. If the learnt features are quite specific to the training environment, then it means that such agent will fail if deployed in another environment with different features. Under the Meta framework, we propose a novel algorithm; MetaDAgger, which is based on a Meta learning framework to aggregate data across different environments. To ensure generalization, we split the learning agent into two: 1) Meta learner, whose objective is to capture generic features not specific to an environment and 2) Low level learner, whose objective is to drive in a specific environment. The role of Meta learning is to smooth the learning performance across different environments.

The two learners are trained using Convolutional Neural Networks (ConvNets) on two different data sets. The meta learner is trained on a data set of environments; where a set of environments are kept for training, while another independent set of environments are kept for testing and are not allowed to alter the learning process. One the other hand, the Low learner is trained on a data set aggregated from each specific environment, in the form of state-action pairs. The link between both learners is done through continual lifelong learning, where the Low learner communicates its learning to the Meta learner whenever a switch from an environment to the other is undergone. In this way the learning is preserved across different environments, which enable the Meta learner to capture general features as proved by the features visualizations. It is worth noting that in (Sallab et al., 2016) an interesting conclusion is that continual learning highly improves the learning curve, which is in line with the proposed metal learning framework proposed in this work.

The experimental setup is based on The Open Race Car Simulator (TORCS) (Wymann et al., 2000). The Meta dataset is formed of 19 tracks, with 10 training tracks and 9 testing tracks. Low learners are based on Convolutional Neural Networks (ConvNets) for policy learning. The results show significant advantage of MetaDAgger over DAgger on both training and testing tracks. MetaDAgger is not only improving generalization, but also significantly improving the sample efficiency and learning time over DAgger. The objective in each low level task is to keep the central lane. The demonstration in all tracks comes from a tradition PID controller, with access to the position of the ego car with respect to the left and right lanes. The control actions is applied to the steering wheel, with continuous actions output. The input states in all experiments are taken as the raw pixels information as provided by TORCS.

The rest of the paper is organized as follows; first we discuss the related work, then the Meta learning framework is described followed by the MetaDAgger algorithm. Then the experimental setup is described followed by the discussion of results and visualizations, and finally we conclude.

2. Related Work

The problem of developing a driving policy is essentially formulated under the framework of optimal control. The solution of the optimization problem is easily found under the traditional planning framework only if an environment model exists, a requirement which is difficult to achieve in the real world of high way, urban or city driving. Model based approach is a wide topic that has been recently tackled in (Polydoros & Nalpantidis, 2017).

On the other hand, model free approaches have undergone a huge advancement in the area of human level control, the Deep Q Net is a famous example of which (Mnih et al., 2015) (Mnih et al., 2013). The success of model free Reinforcement Learning (Sutton, 1988) (Watkins, 1989) has reached the area Automated Driving (Karavolos, 2013) (El Sallab et al., 2017) (Sallab et al., 2016). However, model free approaches require interacting with an environment through actions and rewards scheme. Such interaction usually requires a controlled environment such as simulators and game engines. The reward function is relatively easy acquired in such configuration of game engines. However, it is not clear how such reward function can be obtained in real world without causing too much damage. Moreover, model free approaches are known of their sample inefficiency, which is compensated by huge number of interactions with the simulated environment, which is again infeasible in real world.

Another learning approach based on human demonstrations has been successfully deployed in automated driving (Bojarski et al., 2016). Apprenticeship learning has been tackled in (Abbeel & Ng, 2004) (Ng et al., 2000), under the framework of Inverse Reinforcement Learning (IRL). IRL goal is to deduce the reward function the coach has been trying to achieve or maximize throughout the course of apprenticeship. Such hard goal is achieved on the expense of high complexity algorithm, involving two nested RL loops.

In order to tackle the problem of sample efficiency without knowing the environment models, supervised learning models are employed, where learning is based on demonstrated behavior to imitate (imitation learning). Learning from demonstrations has been approached as a supervised learning problem for automated driving in (Bojarski et al., 2016). The risk of such an approach is the lack of generalization, where supervised learning schemes could converge to exact behavior cloning. The challenge of getting such approaches to work lies in the ability to enrich the training data by introducing new unseen states from the demonstration. One solution is data aggregation, where the agent is able to extend his actions to further states than the demon-

strated ones. Some hacks have been proposed in (Bojarski et al., 2016) to achieve data aggregation, where two cameras has been added in addition to the central camera to augment the training states with different situations supported with the correct action to take in each. A more formal approach has been formulated in the Dataset aggregation (DAgger) algorithm (Ross et al., 2011). DAgger algorithm is also extensible to include safety constraints, as in SafeDAgger (Zhang & Cho, 2016).

Transfer of experience is an essential component of the human learning process. Transfer of knowledge could be in the form of external coaching, demonstrations or apprenticeship, or it could take the form of self accumulated experience through lifelong, continual learning (urgen Schmidhuber et al., 1996). Meta learning (Thrun & Pratt, 2012) (urgen Schmidhuber et al., 1996) is another name of continual and transfer learning. Meta learning has also been used to search the best hyper parameters settings (Hochreiter et al., 2001) (Andrychowicz et al., 2016). Continual learning fosters the transfer of experience across the life time of the agent. In addition, Meta learning provides a framework to transfer the knowledge acquired in one task to other tasks as well thanks to the captured information from task to task in the high level or Meta learner (Thrun & Pratt, 2012).

3. Meta Learning Framework for Automated Driving

In this section the Meta Learning framework is presented $Figure\ 1$. The goal of the framework is to reach smooth and generalized performance over a set of environments$\{E_1, E_2, .., E_N\}$. The performance is measured with the ability to take driving actions (e.g. steering) in a correct manner defined by the demonstrated behavior.

To ensure and test generalization, the set of environments are split into training and testing environments; E_{train} and E_{test}. The learning algorithm is allowed to access the ground truth actions set of training environments E_{train} , while it is not allowed to access the ground truth actions of the test environments E_{test} . The learning is said to be generalized if it achieves the desired performance on the test environments E_{test}. The architecture is based on two learners:

3.1. Low Level Learner (L)

The scope of the Low learner L is to capture the specific features of each environment. The environment in operation E is selected from a pool of training environments E_{train}. The low level learner is associated with a training data set L_{train} which is formed due to interaction with an environment in operation E. The training data set L_{train} is formed of a list of state-action pairs $\{s, a\}^t$, where t is the index of the interaction time stamp, with t=$\{0, 1, ..., T\}$, where T is the end of interaction episode. An interaction episode is terminated with the L reaching the goal, made a terminating mistake or after certain defined number of steps. The state s is the measurement obtained by probing the environment E. The action a is obtained from a demonstration, either from Human or from a reference algorithm.

Under the framework of Automated Driving, L is trained using supervised learning imitating the demonstrated behavior.

3.2. Meta Learner (M)

The goal of the Meta Learner M is to capture general features across all environments $\{E_1, E_2, .., E_N\}$. With each interaction episode, M communicates the initialization parameters to the Low learner L to start interaction with the in operation environment E. After the Low learner finishes an episode, the learnt parameters of L are communicated back to the M to ensure continual learning.

The Meta Learner M parameters are updated based on aggregated data over different interactions with the training environments E_{train}. The result of data set aggregation is called M_{train}. The members of M_{train} have the same format of state-action pairs $\{s, a\}^t$ as L_{train}. The training of M is also following a supervised learning scheme using the training data in M_{train}.

The driving policy π^M is obtained as the parameters of the Meta Learner M, such that an action at time t is obtained as . In the context of Neural Networks, π^M is parameterized by the weights parameters of the network. The performance of π^M is tested against a test data set M_{test}, which is formed from the set of test environments E_{test}.

4. MetaDAgger Algorithm

In this section the MetaDAgger algorithm is described as previous, in the light of the Meta learning framework described in $Figure\ 1$.

The algorithm is based on two main steps:

4.1. Data Collection

During data collection the reference demonstration is interacting with the environment to collect reference data. The reference demonstration could be a human or a reference algorithm. The collected data take the shape of state-action $\{s, a_{ref}\}^t$ pairs at each interaction time step t, where a_{ref} is the action provided by the demonstration. This data is aggregated into M_{train} to train the initial model parameters M, which could be a ConvNet fitting the supervised data.

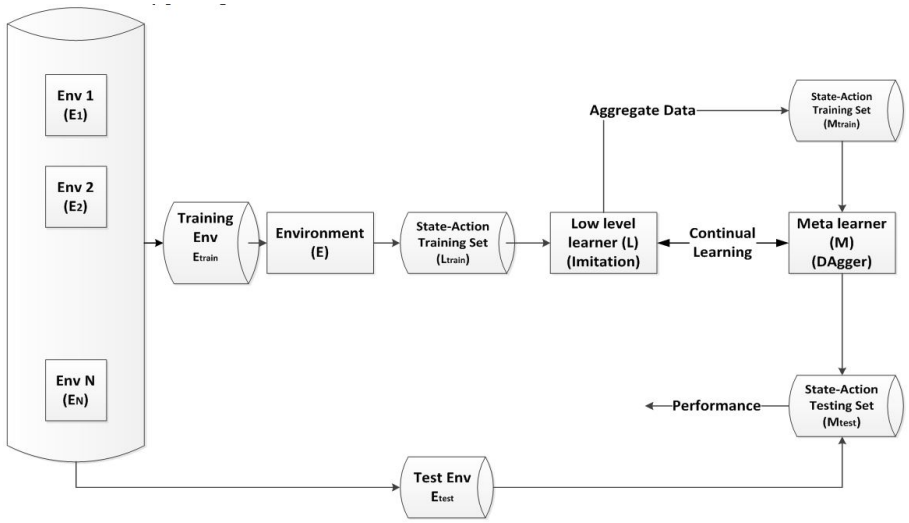

Figure 1. Meta Learning Framework for Dataset Aggregation

Algorithm 1 MetaDAgger

Loop on E_{train}**:** // Data Collection Step
Collect L_{train} += $\{s, a_{ref}\}^t$ // a_{ref} is obtained from a demonstration or a reference algorithm
Aggregate M_{train} += L_{train}
Fit a Meta model M based on M_{train}

Loop until N**-iter is reached** // Data Aggregation Step
for each E in E_{train} **do**
Initialize L = M
Loop for N_{steps}
L interacts with E to measure current state s
Execute a_L = π^L(s), where π^L is obtained using the parameters of L.
if $a_L != a_{ref}$: //Aggregate incorrect actions
Aggregate L_{train} += $\{s, a_{ref}\}^t$
Fit a Low model L based on L_{train}
Save back M=L
Return M, π^M

4.2. Data Aggregation

Data aggregation is repeated for a number of iterations N-iter $> |E_{train}|$, which means that the aggregation is repeated more than once over all the training environments. Every iteration, L interacts with E, starting from the parameters captures in M, which ensures continual learning. The parameters of L define the interaction policy π that will be executed. The interaction happens for N-steps, during which data aggregation takes place. The action to be executed is obtained from the policy parameters π^L based on the measured state st. The aggregated data L_{train} is then used to retrain L. At the end of each episode, the model is saved back to M to ensure lifelong and continual learning for the next environment.

In order to improve sample efficiency, only the incorrect actions are saved. Hence, only actions that are different from the reference actions are saved. The difference is taken within certain tolerance (say 40% error). This is important in two aspects: 1) the aggregated data set is focused on mistakes only, which need to be corrected when L is re-trained, and 2) in real world, we have no access to the reference action a_{ref}, however, it is easier to ask a human supervisor to attend and correct only the algorithm mistakes, even before it causes damage, in which case the corrected state-action pair is aggregated and retraining happens to correct it.

5. Experimental Setup

5.1. Environment

MetaDAgger is designed to be ready for deployment in real car. The algorithm steps ensure data sampling efficiency and zero damage to the experimental car. Moreover, aggregation of incorrect actions ensures easy human supervision requiring the least effort.

As a first step towards deploying MetaDAgger in real car, we perform our experiments under TORCS game engine. We divide the 19 tracks in TORCS into 10 training tracks (E_{train}), and 9 test tracks (E_{test}). We use the Gym-TORCS environment (GYM, 2016), with the visual image input (Vis, 2016). Hence, the input state is the raw image pixels (64x64) as provided by the visual TORCS client as shown in $Figure\ 2$. The actions provided by MetaDAgger are the continuous steering angle values. The supervision signal is obtained using a reference PID controller

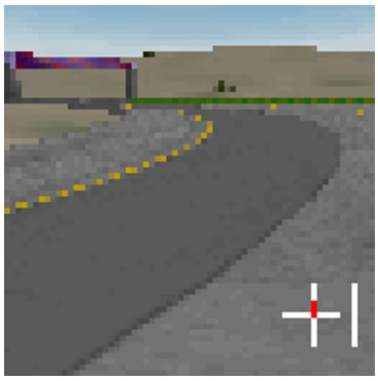

Figure 2. TORCS visual input state

that can access the position of the car from the two side lanes, which is provided by the TORCS game engine. Although we could use this supervision signal to aggregate all encountered states in the data aggregation step, however, we keep aggregating only the incorrect ones. Since the actions are continuous, mapping the absolute action value to a steering angle is sensitive to small differences. Because of this, we relax the exact matching between the reference and algorithm actions to a certain tolerance, empirically set to 40%.

5.2. Network Architecture and Hyper Parameters

For the Low learner L and the Meta learner M we fit the same ConvNet model shown in $Figure\ 3$, which simplifies copying models back and forth between M and L for continual learning. The kernel sizes are kept to a small size (3x3) due to the small input image size (64x64). Batch normalization is found to significantly improve the learning time (Ioffe & Szegedy, 2015). Xaviar initialization (Glorot & Bengio, 2010) is used to initialize all weights. Dropout of 0.25 is used in the convolution layers, followed by Dropout of 0.5 in the fully connected ones. ReLU activation is used in all layers, except for the output, where linear activation to produce the continuous output. The loss criterion is Mean Squared Error (MSE) minimization, since our task is a regression one.

5.3. Results

We first evaluate the generalization performance of MetaDAgger. The evaluation is done over the 9 test tracks M_{test}. Results are shown in $Figure\ 4$. The horizontal axis represents the number of data aggregation iterations (N-iter), while the vertical axis is the number of steps the car moved without exiting the lane. We should note that 1000 steps mean one complete lap.

We compare the performance of MetaDAgger against traditional DAgger (Ross et al., 2011). Results are shown in $Figure\ 5$, where the vertical axis represents the number of steps without collision (one complete lap equals 1000 steps), and the horizontal axis represents the test track number. For some of the test tracks, MetaDAgger is able to complete one or two laps. For other tracks, performance is better than DAgger, although no laps are completed.

We analyze the learnt features in case of DAgger and MetaDAgger. We use Grad CAM (Selvaraju et al., 2016) visualization technique of the gradient of the last neuron (linear activation for action output), projected back to the input image. The result is a heat map showing which part of the image contributes more to the output. In case of DAgger, the learnt features are more memorizing and capturing the details of the track it is trained on, as shown in $Figure\ 6$. For example, we can see the most important part is the horizon or the place with the mountain; this is because the theme of this track is a desert one. It is then understood why it is hard for such model to generalize to other tracks with different themes.

On the other hand, when we visualize the filters of MetaDAgger in $Figure\ 7$, we see that the learnt features are more representing the relevant features to the driving task, like the positions of the lanes. This proves the generality of such model.

We further evaluate the sample efficiency of MetaDAgger versus DAgger in $Figure\ 8$. Here we evaluate how many data aggregation iterations are needed in order for the algorithm to complete a lap. An iteration terminates after certain number of steps, a complete lap or a termination condition. In our experiments the number of steps are taken to be 1000 (complete lap), while the termination condition is met when the car is out of the track. In case of DAgger, it takes 4 iterations, while for MetaDAgger it takes only two. Moreover, just after the data collection and behavior cloning step, the algorithm is able to complete 80% of the track.

6. Conclusion

In this paper we presented a framework for generalized imitation learning, based on the principles of Meta learning and data set aggregation. The proposed algorithm MetaDAgger is shown to be able to generalize on unseen test tracks, achieving much less training time and better sample efficiency. The results on TORCS show significant improvement on both training and testing tracks, supported by visualizations of the generic learnt features. The proposed algorithm is designed to be ready for real car deployment, where the data aggregation step is only limited to correcting the mistakes the algorithm makes in real world.

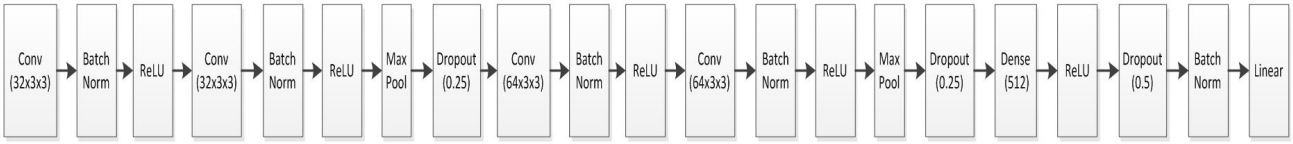

Figure 3. ConvNet architecture for Low and Meta learners

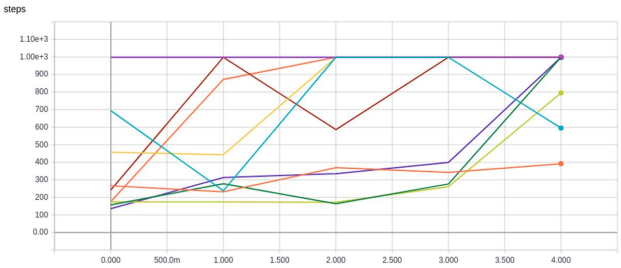

Figure 4. MetaDAgger performance on test tracks. Most of the tracks are completed just after 0 or 1 iterations of data aggregation.

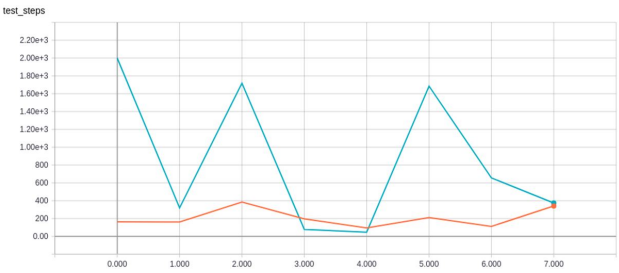

Figure 5. MetaDAgger vs. DAgger. Overall MetaDAgger is better than DAgger. For some tracks, it is still hard even with Meta learning to capture some hard turns.

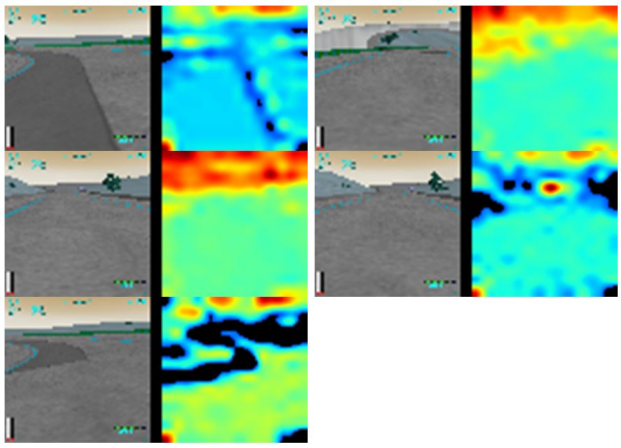

Figure 6. DAgger visualization. Most of the features represent the horizon or the mountain features.

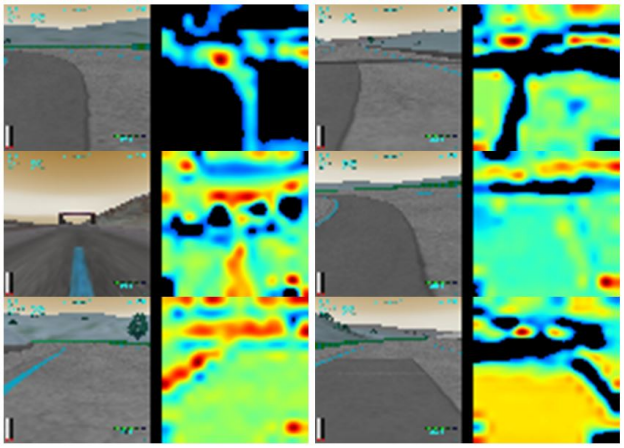

Figure 7. MetaDAgger filter visualization. Lane features are captured in many cases.

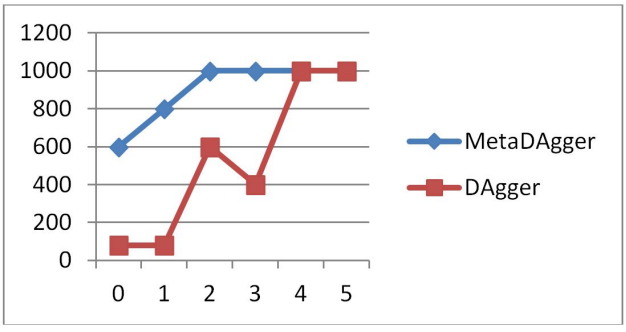

Figure 8. Sample efficiency of MetaDAgger vs. DAgger. After only data collection, MetaDAgger is able to complete 80% of the lap.

References

Gym-torcs. *Software available at https : //github.com/ugo − nama − kun/gym_torcs*, 2016.

Visual-torcs. *Software available at https : //github.com/giuse/vtorcs*, 2016.

Abbeel, Pieter and Ng, Andrew Y. Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the twenty-first international conference on Machine learning*, pp. 1. ACM, 2004.

Andrychowicz, Marcin, Denil, Misha, Gomez, Sergio, Hoffman, Matthew W, Pfau, David, Schaul, Tom, and de Freitas, Nando. Learning to learn by gradient descent by gradient descent. In *Advances in Neural Information Processing Systems*, pp. 3981–3989, 2016.

Bojarski, Mariusz, Del Testa, Davide, Dworakowski, Daniel, Firner, Bernhard, Flepp, Beat, Goyal, Prasoon, Jackel, Lawrence D, Monfort, Mathew, Muller, Urs, Zhang, Jiakai, et al. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316*, 2016.

El Sallab, Ahmad, Abdou, Mohammed, Perot, Etienne, and Yogamani, Senthil. Deep reinforcement learning framework for autonomous driving. *Autonomous Vehicles and Machines, Electronic Imaging*, 2017.

Glorot, Xavier and Bengio, Yoshua. Understanding the difficulty of training deep feedforward neural networks. In *Aistats*, volume 9, pp. 249–256, 2010.

Hochreiter, Sepp, Younger, A, and Conwell, Peter. Learning to learn using gradient descent. *Artificial Neural NetworksICANN 2001*, pp. 87–94, 2001.

Ioffe, Sergey and Szegedy, Christian. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

Karavolos, Daniel. Q-learning with heuristic exploration in simulated car racing. 2013.

Le-Anh, Tuan and De Koster, MBM. A review of design and control of automated guided vehicle systems. *European Journal of Operational Research*, 171(1):1–23, 2006.

Mnih, Volodymyr, Kavukcuoglu, Koray, Silver, David, Graves, Alex, Antonoglou, Ioannis, Wierstra, Daan, and Riedmiller, Martin. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

Mnih, Volodymyr, Kavukcuoglu, Koray, Silver, David, Rusu, Andrei A, Veness, Joel, Bellemare, Marc G, Graves, Alex, Riedmiller, Martin, Fidjeland, Andreas K, Ostrovski, Georg, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

Ng, Andrew Y, Russell, Stuart J, et al. Algorithms for inverse reinforcement learning. In *Icml*, pp. 663–670, 2000.

Pasquier, Michel, Quek, Chai, and Toh, Mary. Fuzzylot: a novel self-organising fuzzy-neural rule-based pilot system for automated vehicles. *Neural networks*, 14(8): 1099–1112, 2001.

Polydoros, Athanasios S and Nalpantidis, Lazaros. Survey of model-based reinforcement learning: Applications on robotics. *Journal of Intelligent & Robotic Systems*, 86 (2):153–173, 2017.

Ross, Stéphane, Gordon, Geoffrey J, and Bagnell, Drew. A reduction of imitation learning and structured prediction to no-regret online learning. In *AISTATS*, volume 1, pp. 6, 2011.

Sallab, Ahmad El, Abdou, Mohammed, Perot, Etienne, and Yogamani, Senthil. End-to-end deep reinforcement learning for lane keeping assist. *arXiv preprint arXiv:1612.04340*, 2016.

Selvaraju, Ramprasaath R, Das, Abhishek, Vedantam, Ramakrishna, Cogswell, Michael, Parikh, Devi, and Batra, Dhruv. Grad-cam: Why did you say that? visual explanations from deep networks via gradient-based localization. *arXiv preprint arXiv:1610.02391*, 2016.

Sutton, Richard S. Learning to predict by the methods of temporal differences. *Machine learning*, 3(1):9–44, 1988.

Thrun, Sebastian and Pratt, Lorien. *Learning to learn*. Springer Science & Business Media, 2012.

urgen Schmidhuber, J, Zhao, Jieyu, and Wiering, Marco. Simple principles of metalearning. 1996.

Watkins, Christopher John Cornish Hellaby. *Learning from delayed rewards*. PhD thesis, University of Cambridge England, 1989.

Wymann, Bernhard, Espié, Eric, Guionneau, Christophe, Dimitrakakis, Christos, Coulom, Rémi, and Sumner, Andrew. Torcs, the open racing car simulator. *Software available at http://torcs. sourceforge. net*, 2000.

Zhang, Jiakai and Cho, Kyunghyun. Query-efficient imitation learning for end-to-end autonomous driving. *arXiv preprint arXiv:1605.06450*, 2016.