Learning to Drive in a Day

Alex Kendall Jeffrey Hawke David Janz Przemyslaw Mazur Daniele Reda John-Mark Allen Vinh-Dieu Lam Alex Bewley Amar Shah

Wayve

research@wayve.ai

Abstract: We demonstrate the first application of deep reinforcement learning to autonomous driving. From randomly initialised parameters, our model is able to learn a policy for lane following in a handful of training episodes using a single monocular image as input. We provide a general and easy to obtain reward: the distance travelled by the vehicle without the safety driver taking control. We use a continuous, model-free deep reinforcement learning algorithm, with all exploration and optimisation performed on-vehicle. This demonstrates a new framework for autonomous driving which moves away from reliance on defined logical rules, mapping, and direct supervision. We discuss the challenges and opportunities to scale this approach to a broader range of autonomous driving tasks.

Keywords: Deep Reinforcement Learning, Autonomous Vehicles

1 Introduction

Autonomous driving is a topic that has gathered a great deal of attention from both the research community and companies, due to its potential to radically change mobility and transport. Broadly, most approaches to date focus on formal logic which define driving behaviour in annotated 3D geometric maps. This can be difficult to scale, as it relies heavily on external mapping infrastructure rather than primarily using an understanding of the local scene.

In order to make autonomous driving a truly ubiquitous technology, we advocate for robotic systems which address the ability to drive and navigate in absence of maps and explicit rules, relying - just like humans - on a comprehensive understanding of the immediate environment [1] while following simple higher level directions (e.g., turn-by-turn route commands). Recent work in this area has demonstrated that this is possible on rural country roads, using GPS for coarse localisation and LIDAR to understand the local scene [2].

In the recent years, reinforcement learning (RL) – a machine learning subfield focused on solving Markov Decision Problems (MDP) [3] where an agent learns to select actions in an environment in an attempt to maximise some reward function – has shown an ability to achieve super-human results at games such as Go [4] or chess [5], a great deal of potential in simulated environments like computer games [6], and on simple tasks with robotic manipulators [7]. We argue that the generality of reinforcement learning makes it a useful framework to apply to autonomous driving. Most importantly, it provides a corrective mechanism to improve learned autonomous driving behaviour.

To this end, in this paper we:

1. pose autonomous driving as an MDP, explain how to design the various elements of this problem to make it simpler to solve, whilst keeping it general and extensible,
2. show that a canonical RL algorithm (deep deterministic policy gradients) can rapidly learn a simple autonomous driving task in a simulation environment,
3. discuss the system set-up required to make learning to drive possible on a real-life vehicle,
4. learn to drive a real vehicle in a few episodes with a continuous deep reinforcement learning algorithm, using only on-board computation.

We therefore present the first demonstration of a reinforcement learning agent driving a real car.

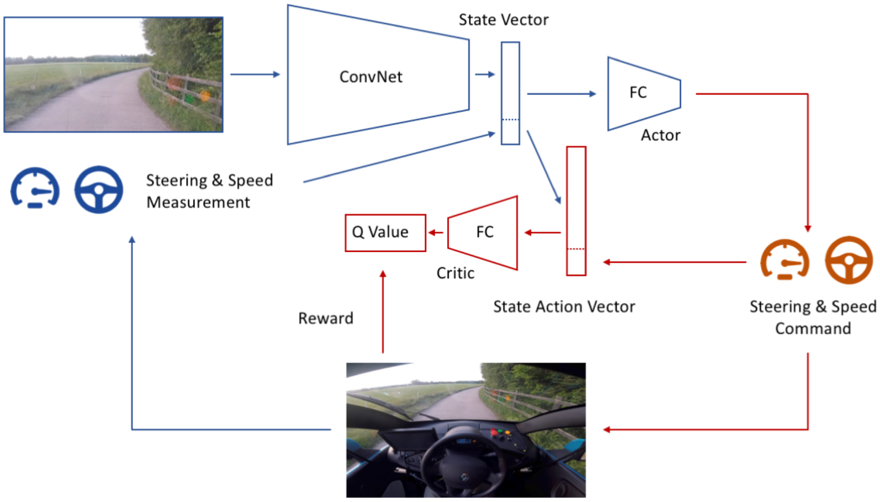


Figure 1: We design a deep reinforcement learning algorithm for autonomous driving. This figure illustrates the actor-critic algorithm which we use to learn a policy and value function for driving.

2 Related Work

We believe this is the first work to show that reinforcement learning is a viable approach to autonomous driving. We are motivated by its potential to scale beyond that of imitation learning, and hope the research community examines autonomous driving from a reinforcement learning perspective more closely. The closest work in the current literature can be categorised as either imitation learning or classical approaches relying on mapping.

Mapping approaches. Since early examples [8, 9], autonomous vehicle systems have been designed to navigate safely through complex environments using advanced sensing and control algorithms [10, 11, 12]. These systems are traditionally composed of many specific independently engineered components, such as perception, state estimation, mapping, planning and control [13]. However, because each component needs to be individually specified and tuned, this can be difficult to scale to more difficult driving scenarios due to complex interdependencies.

Significant effort has been focused on computer vision components for this modular approach. Localisation such as [14] facilitates control of the vehicle [15] within the mapped environment, while perception methods such as semantic segmentation [1] enable the robot to interpret the scene. These modular tasks are supported by benchmarks such as [16] and [17].

These modular mapping approaches are largely the focus of commercial efforts to develop autonomous driving systems; however, they present an incredibly complex systems engineering challenge, which has yet to be solved.

Imitation learning approaches. A more recent approach to some driving tasks is imitation learning [18, 19], which aims to learn a control policy by observing expert demonstrations. One important advantage of this approach is that it can use end-to-end deep learning, optimising all parameters of a model jointly with respect to an end goal thus reducing the effort of tuning of each component. However, imitation learning is also challenging to scale. It is impossible to obtain expert examples to imitate for every potential scenario an agent may encounter, and it is challenging to deal with distributions of demonstrated policies (e.g., driving in each lane).

Reinforcement Learning. Reinforcement learning is a broad class of algorithms for solving Markov Decision Problems (MDPs). An MDP consists of:

- a set $\mathcal{S}$ of states,
- a set $\mathcal{A}$ of actions,

- a transition probability function $p\colon \mathcal{S} \times \mathcal{A} \to \mathcal{P}(\mathcal{S})$, which to every pair $(s, a) \in \mathcal{S} \times \mathcal{A}$ assigns a probability distribution $p(\cdot|s, a)$ representing the probability of entering a state from state s using action a,
- a reward function $R\colon \mathcal{S} \times \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, which describes the reward $R(s_{t+1}, s_t, a_t)$ associated with entering state s_{t+1} from state s_t using action a_t,
- a future discount factor $\gamma \in [0, 1]$ representing how much we care about future rewards.

The solution of an MDP is a policy $\pi\colon \mathcal{S} \to \mathcal{A}$ that for every $s_0 \in \mathcal{S}$ maximises:

$$V_\pi(s_0) = \mathbb{E}\left(\sum_{t=0}^{\infty} \gamma^t R(s_{t+1}, s_t, \pi(s_t))\right), \tag{1}$$

where the expectation is taken over states s_{t+1} sampled according to $p(s_{t+1}|s_t, \pi(s_t))$.

In our setting, we use a finite time horizon T in place of infinity in the above formula. This is equivalent to one of the states being terminal, i.e. it cannot be escaped and any action at that state gives zero reward.

Rearranging the above equation into a recurrent form, we get one of the two Bellman equations:

$$V_\pi(s_0) = \mathbb{E}\Big(R(s_1, s_0, \pi(s_0)) + \gamma V_\pi(s_1)\Big). \tag{2}$$

Here the expectation is taken only over s_1 sampled according to $p(s_1|s_0, \pi(s_0))$. For reference, let us present the other Bellman equation:

$$Q_\pi(s_0, a_0) = \mathbb{E}\Big(R(s_1, s_0, a_0) + \gamma Q_\pi(s_1, \pi(s_1))\Big), \tag{3}$$

where $Q_\pi(s_0, a_0)$ is the expected cumulative discounted reward received while starting from state s_0 with action a_0 and following policy π thereafter. Again the expectation is taken over s_1 sampled according to $p(s_1|s_0, a_0)$

In other words, reinforcement learning algorithms aim to learn a policy π that obtains a high cumulative reward. They are generally split into two categories: model based and model-free reinforcement learning. In the former approach, explicit models for the transition and reward functions are learnt, and then used to find a policy that maximises cumulative reward under those estimated functions. In the latter, we directly estimate the value $Q(s, a)$ of taking action a in state s, and then follow a policy that selects the action with the highest estimated value in each state.

Model-free reinforcement learning is extremely general. Using it, we can (in theory) learn any task we can imagine, whereas model based algorithms can be only as good as the model learned. On the other hand, model based methods tend to be more data-efficient than model-free ones. For further discussion, see [20].

3 System Architecture

3.1 Driving as a Markov Decision Process

A key focus of this paper is the set-up of driving as an MDP. Our goal is that of autonomous driving, and the exact definition of the state space $\mathcal{S}$, action space $\mathcal{A}$ and reward function R are free for us to be defined. The transition model is implicitly fixed once a state and action representation is fixed, with the remaining degrees of freedom – the transitions themselves – dictated by the mechanics of the simulator/vehicle used.

State space. Key to defining the state space is the definition of the observations O_t that the algorithm receives at each time step. Many sensors have been developed in order to provide sophisticated observations for driving algorithms, not limited to LIDAR, IMUs, GPS units and IR depth sensors; an endless budget could be spent on advanced sensing technology. In this paper, we show that for simple driving tasks it is sufficient to use a monocular camera image, together with the observed vehicle speed and steering angle. Theoretically, state s_t is to be a Markov representation of all previous observations. An approximation a fixed length approximately Markov state could be obtained by, for example, using a Recurrent Neural Network to recursively combine observations. However, for the tasks we consider, the observation itself serves as a good enough approximation of the state.

A second consideration is how to treat the image itself: the raw image could be fed directly into the reinforcement learning algorithm through a series of convolutions [21]; alternatively, a small compressed representation of the image, using, for example, a Variational Autoencoder [22] [23], could be used. We compare the performance of reinforcement learning using these two approaches in Section 4.

Action space. Driving itself has what one might think are a natural set of actions: throttle, brake, signals etc. But what domain should the output of the reinforcement learning algorithm be? The throttle itself can be described as discrete, either on or off, or continuous, in a range isometric to [0, 1]. An alternative is to reparameterise the throttle in terms of a speed set-point, with throttle output by a classical controller in an attempt to match the set-point. Overall, experiments on a simple simulator (Section 4.1) showed that continuous actions, whilst somewhat harder to learn, provide for a smoother controller. We found it important to limit the steering angle requested from the vehicle.

Reward function. Design of reward functions can approach supervised learning – given a lane classification system, a reward to learn lane-following can be set up in terms of minimising the predicted distance from centre of lane, the approach taken in [24]. This approach is limited in scale: the system can only be as good as the human intuition behind the hand-crafted reward. We do not take this approach. Instead, we define the reward as forward speed and terminate an episode upon an infraction of traffic rules – thus the value of a given state $V(s_t)$ corresponds to the average distance travelled before an infraction. A fault that may be identified is that the agent may choose to avoid more difficult manoeuvres, e.g. turning right in the UK (left in US). Command conditional rewards may be utilised in future work to avoid this.

3.2 Reinforcement Learning Algorithm – Deep Deterministic Policy Gradients

We selected a simple continuous action domain model-free reinforcement learning algorithm: deep deterministic policy gradients (DDPG) [24], to show that an off-the-shelf reinforcement learning algorithm with no task-specific adaptation is capable of solving the MDP posed in Section 3.1.

DDPG consists of two function approximators: a critic $Q\colon \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, which estimates the value $Q(s, a)$ of the expected cumulative discounted reward upon using action a in state s, trained to satisfy the Bellman equation

$$Q(s_t, a_t) = r_{t+1} + \gamma(1 - d_t)Q(s_{t+1}, \pi(s_{t+1})),$$

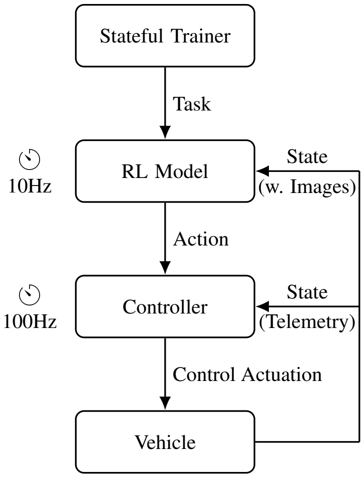
under a policy given by the actor $\pi\colon \mathcal{S} \to \mathcal{A}$, which attempts to estimate a Q-optimal policy $\pi(s) = \mathrm{argmax}_a Q(s, a)$; here $(s_t, a_t, r_{t+1}, d_{t+1}, s_{t+1})$ is an experience tuple, a transition from state s_t to s_{t+1} using action a_t and receiving reward r_{t+1} and "done" flag d_{t+1}, selected from a buffer of past experiences. The error in the Bellman equality, which the critic attempts to minimise, is termed the temporal difference (TD) error. Many variants of actor-critic methods exist, see e.g. [25, 26].

DDPG training is done online. Beyond the infrastructure of setting up such a buffer for use on a real vehicle (which requires it to be tolerant of missing/faulty episodes and any-time stoppable), reinforcement learning can be sped up by selecting the most "informative" examples from the replay buffer. We do so using a commonly established method called prioritised experience replay [27]: we sample experience tuples with probability proportional to the TD error made by the critic. The weights used for this sampling are updated upon each optimisation step with minimal overhead; new samples are given infinite weight to ensure all samples are seen at least once.

DDPG is an off-policy learning algorithm, meaning that actions performed during training come from a policy distinct from the learn optimal policy by the actor. This happens in order to gain diverse state-action data outside of the narrow distribution that would be seen by the optimal policy, and thus increase robustness. We use a standard method of achieving this in the context of continuous reinforcement learning methods: our exploration policy is formed by adding discrete

```
while True do
    Request task
    Waiting for environment reset
    if task is train then
        Run episode with noisy policy
        if exploration time is over then
            Optimise model
        end if
    else if task is test then
        Run episode with optimal policy
    else if task is undo then
        Revert previous train/test task
    else if task is done then
        Exit experiment
    end if
end while
```

(a) Task-based workflow for on-vehicle training



(b) Policy execution architecture, used to run episodes during model training (or run continuously under test).

Figure 2: Outline of the workflow and the architecture of the training algorithm.

Ornstein-Uhlenbeck process noise [28] to the optimal policy. Therefore, at each step we add to optimal actions noise x_t given by:

$$x_{t+1} = x_t + \theta(\mu - x_t) + \sigma\epsilon_t, \tag{4}$$

where θ, μ, σ are hyperparameters and $\{\epsilon_t\}_t$ are i.i.d. random variables sampled from the normal distribution $N(0, 1)$. These parameters need to be tuned carefully, as there is a direct trade-off between noise utility and comfort of the safety driver. Strongly mean reverting noise with lower variance is easier to anticipate, whilst higher variance noise provides better state-action space coverage.

3.3 Task-based Training Architecture

Deployment of a reinforcement learning algorithm on a full-sized robotic vehicle running in a real world environment requires adjustment of common training procedures, to account for both driver intervention and external variables affecting the training.

We structure the architecture of the algorithm as a simple state machine, outlined in Figure 2a, in which the safety driver is in control of the different tasks. We define four tasks: train, test, undo and done. The definition of these tasks allows the system to be both interactive and stateful, favouring an on-demand execution of episodes instead of an a priori fixed schedule.

The train and test tasks allow us to interact with the vehicle in autonomous mode, executing the current policy. The difference between the two tasks consists in noise being added to the model output and the model being optimised in training tasks, whereas test tasks run directly the model output actions. During early episodes, we skip optimisation to favour exploration of the state space.

Each episode is executed until the system detects that automation is lost (i.e. the driver intervened). In a real world environment, the system can not reset automatically between episodes, unlike agents in simulation or in a constrained environment. We require a human driver to reset the vehicle to a valid starting state. Upon episode termination, while the safety driver performs this reset, the model is being optimised, minimising the time spent between episodes.

The undo and done tasks depict the key differences in the architecture. The system may terminate an episode for a variety of valid reasons other than failing to drive correctly: these episodes can not be considered for the purposes of training. The undo task is introduced for this reason, as it allows us to undo the episode and restore the model as it was before running that episode. A common example in our experiments is other drivers seeking to use the road being used as the environment. The done task allows us to gracefully exit the experiment at any given moment, and is helpful since the procedure is interactive and it doesn't run for a fixed number of episodes.

4 Experiments

The main task we use to showcase the vehicle is that of lane-following; this is the same task as addressed in [24], however done on a real vehicle as well as on simulation, and done from image input, without knowledge of lane position. It is a task core to driving, and was the cornerstone of the seminal ALVINN [18]. We first accomplish this task in simulation in Section 4.1, and then use these results and knowledge of appropriate hyperparameters to demonstrate a solution on a real vehicle in Section 4.2.

For both simulation and real-world experiments we use a small convolutional neural network. Our model has four convolutional layers, with 3×3 kernels, stride of 2 and 16 feature dimensions, shared between the actor and critic models. We then flatten the encoded state and concatenate the vector the scalar state for the actor, additionally concatenating the actions for the critic network. For both networks we then apply one fully-connected layer with feature size 8 before regressing to the output. For the VAE experiments, a decoder of the same size as the encoder is used, replacing strided convolution with transposed convolution to upsample the features. A graphical depiction is shown in Figure 1.

4.1 Simulation

To test reinforcement learning algorithms in the context of lane following from image inputs we developed a 3D driving simulator, using Unreal Engine 4. It contains a generative model for country roads, supports varied weather conditions and road textures, and will in the future support more complex environments (see Figure 3 for game screenshots).

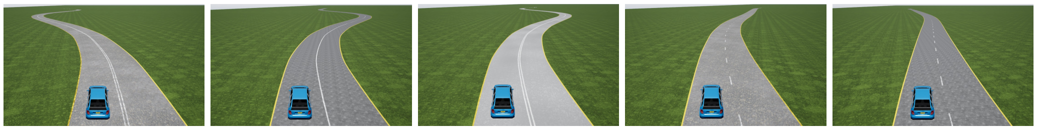


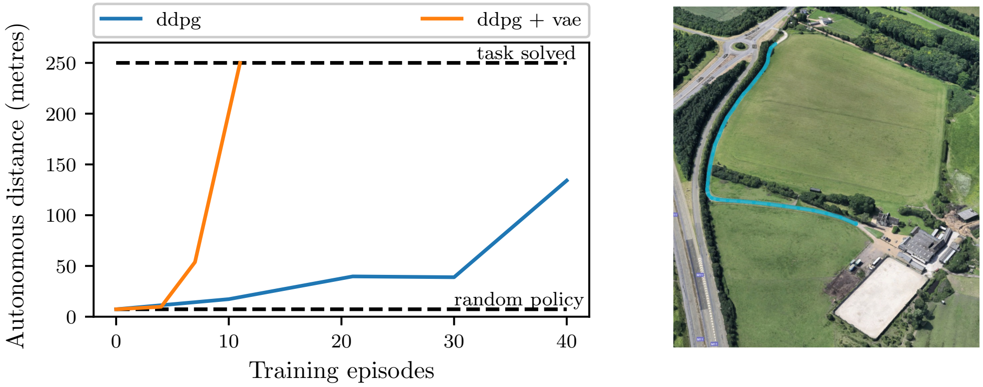
Figure 3: Examples of different road environments randomly generated for each episode in our lane following simulator. We use procedural generation to randomly vary road texture, lane markings and road topology each episode. We train using a forward facing driver-view image as input.

The simulator proved essential for tuning reinforcement learning parameters including: learning rates, number of gradient steps to take following each training episode and the correct termination procedure – conservative termination leads to a better policy. It confirmed a continuous action space is preferable – discrete led to a jerky policy – and that DDPG is a suitable reinforcement learning algorithm. As described in the environment setup in Section 3.1, reward granted in the simulator corresponded to the distance travelled before exiting lane, with new episodes resetting the car to the centre of the lane.

We found that we could reliably learn to learn follow in simulation from raw images within 10 training episodes. Furthermore, we found little advantage to using a compressed state representation (provided by a Variational Autoencoder). We found the following hyperparameters to be most effective, which we use for our real world experiments: future discount factor of 0.9, noise half-life of 250 episodes, noise parameters of θ of 0.6 and σ of 0.4, 250 optimisation steps between episodes with batch size 64 and gradient clipping of 0.005.

4.2 Real-world driving

Our real world driving experiments mimic in many ways those conducted in simulation. However, executing this experiment in the real world is significantly more challenging. Many environmental factors cannot be controlled, and real-time safety and control systems must be implemented. For these experiments, we use a 250 meter section of road. The car begins at the start of the road to commence training episodes. When the car deviates from the lane and enters an unrecoverable position, the safety driver takes control of the vehicle ending the episode. The vehicle is then returned to the center of the lane to begin the next episode. We use the same hyperparameters that we found to be effective in simulation, with the noise model adjusted to give vehicle behaviour similar to that in simulation under the dynamics of the vehicle itself.



(a) Algorithm results

(b) Route

Figure 4: Using a VAE with DDPG greatly improves data efficiency in training over DDPG from raw pixels, suggesting that state representation is an important consideration for applying reinforcement learning on real systems. The 250m driving route used for our experiments is shown on the right.

	Training			Test	
Model	Episodes	Distance	Time	Meters per Disengagement	# Disengagements
Random Policy	-	-	-	7.35	34
Zero Policy	-	-	-	22.7	11
Deep RL from Pixels	35	298.8 m	37 min	143.2	1
Deep RL from VAE	11	195.5 m	15 min	-	0

Table 1: Reinforcement learning results on an autonomous vehicle over a 250m length of road. We report the best performance for each model. We observe the baseline RL agent can learn to lane follow from scratch, while the VAE variant is much more efficient, learning to succesfully drive the route after only 11 training episodes.

We conduct our experiments using a modified Renault Twizy vehicle, which is a two seater electric vehicle, shown in Figure 1. The vehicle weighs 500kg, has a top speed of 80 km/h and has a range of 100km on a single battery charge. We use a single monocular forward-facing video camera mounted in the centre of the roof at the front of the vehicle. We use retrofitted electric motors to actuate the brake and steering, and electronically emulate the throttle position to regulate torque to the wheels. All computation is done on-board using a single NVIDIA Drive PX2 computer. The vehicle's drive-by-wire automation automatically disengages if the safety driver intervenes, either by using vehicle controls (brake, throttle, or steering), toggling the automation mode, or pressing the emergency stop. An episode would terminate when either speed exceeded 10km/h, or drive-by-wire automation disengaged, indicating the safety driver has intervened. The safety driver would then reset the car to the centre of the road and continue with the next episode.

Table 1 shows the results of these experiments. Here, the major finding is that reinforcement learning can solve this problem in a handful of trials. Using 250 optimisation steps with batch size 64 took approximately 25 seconds, which made the experiment extremely manageable, considering manoeuvring the car to the centre of the lane to commence the next episode takes approximately 10 seconds anyway. We also observe in the real world, where the visual complexity is much more difficult than simulation, a compressed state representation provided by a Variational Autoencoder trained online together with the policy greatly improved reliability of the algorithm. We compare our method to a zero policy (driving straight with constant speed) and random exploration noise, in order to confirm that the trial indeed required a non-trivial policy.

A video of the training process for our vehicle learning to drive the 250m length of private road with the stateful RL training architecture (Section 3.3) is available at:
`https://wayve.ai/blog/learning-to-drive-in-a-day-with-reinforcement-learning`

5 Discussion

This work presents the first application of reinforcement learning to a full sized autonomous vehicle. The experiments demonstrate we are able to learn to lane follow with just thirty minutes of training – all done on on-board computers.

In order to tune hyperparameters, we built a simple simulated driving environment where we experimented with reinforcement learning algorithms, maximising distance before a traffic infraction using DDPG as a canonical algorithm. The parameters found transferred amicably to the real-world, where we rapidly trained a policy to drive a real vehicle on a private road, with a reward signal consisting only of speed and termination upon control driver taking control. Notably, this reward requires no further information or maps of the environment. With more data, vehicles and larger models, this framework is general enough to scale to more complex driving tasks.

Whilst viable, this approach will require translation of reinforcement learning research advancements, as well as work on core reinforcement learning algorithms if it is to become a leading approach for scaling autonomous driving. The following are some of our thoughts on the future work required.

The first area for development suggested by the results here is a better state representation. Our experiments have shown that a simple Variational Autoencoder greatly improves the performance of DDPG in the context of driving a real vehicle. Beyond pixel-space autoencoders is a wealth of computer vision research addressing effective compression of images: here existing work in areas such as semantic segmentation, depth, egomotion and pixel-flow provide an excellent prior for what is important in driving scenes [29, 1, 30]. This research needs to be integrated with reinforcement learning approaches for real tasks, both model-free and model-based.

However, unsupervised state encoding alone will likely not be sufficient. In order to compress the state in a manner that makes it simple to learn a policy with just a small number of samples, information on which elements of the state (image observation) are important is required. This information should come from the reward and terminal signals. Reward and terminal information can be incorporated in an encoding in many ways, but one difficulty always prevails: credit allocation. Rewards obtained at a specific time step may be related to observations received many time-steps in the past. Thus good models used for this application will contain a temporal component.

Two areas that could greatly improve the availability of data for the application of reinforcement learning to real autonomous driving are semi-supervised learning [30] and domain transfer [31], both of which have recently received much attention. Whilst only a small portion of driving data might have rewards and terminals associated with it, as those are costly to obtain, the image embeddings – and perhaps other aspects of models – could benefit from driving data captured from dashcams in every-day vehicles. These could be used to pre-train the image autoencoder. In the context of a model-based RL system, these could also be used to approximate state transition functions, whilst advances in semi-supervised learning might allow us to utilise this data without reward/terminal labels data. Domain transfer, on the other hand, may allow us to create simulations sufficiently convincing that reward and terminal data from these may be used to train a policy that can be transferred directly onto a real car.

The algorithm used here is intentionally a common canonical approach, chosen to demonstrate the ease with which reinforcement learning may be applied to driving. Many improvements to it have been developed in the wider literature, including the use of natural gradients [32]. Other research has looked at better transformation of observations into states, typically using an RNN [33, 34], as well as methods to perform multi-step planning, as in [35]. It is no question that these could provide superior performance.

New advances in model-based reinforcement provide alternative exciting avenues for autonomous driving research, with work such as [20] showing outstanding performance of models when observing directly the state of a physical system. This could offer significant benefits to an image-based domain. Alternative model-based approaches which may work include [36].

We hope this paper inspires more research into applying reinforcement learning research to autonomous driving, perhaps combining it with elements from other machine learning techniques such as imitation learning and control theory. The method here solved a simple driving task in half an hour – what more could be done in a day?

References

[1] V. Badrinarayanan, A. Kendall, and R. Cipolla. Segnet: A deep convolutional encoder-decoder architecture for scene segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2017.

[2] T. Ort, L. Paull, and D. Rus. Autonomous vehicle navigation in rural environments without detailed prior maps. In *International Conference on Robotics and Automation (ICRA)*, 2018.

[3] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 1998.

[4] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. P. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587): 484–489, 2016.

[5] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, T. P. Lillicrap, K. Simonyan, and D. Hassabis. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *CoRR*, abs/1712.01815, 2017.

[6] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

[7] S. Gu, E. Holly, T. Lillicrap, and S. Levine. Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates. In *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, pages 3389–3396. IEEE, 2017.

[8] T. Kanade, C. Thorpe, and W. Whittaker. Autonomous land vehicle project at cmu. In *Proceedings of the 1986 ACM fourteenth annual conference on Computer science*, pages 71–80. ACM, 1986.

[9] R. S. Wallace, A. Stentz, C. E. Thorpe, H. P. Moravec, W. Whittaker, and T. Kanade. First results in robot road-following. In *IJCAI*, pages 1089–1095. Citeseer, 1985.

[10] M. Montemerlo, J. Becker, S. Bhat, H. Dahlkamp, D. Dolgov, S. Ettinger, D. Haehnel, T. Hilden, G. Hoffmann, B. Huhnke, et al. Junior: The stanford entry in the urban challenge. *Journal of field Robotics*, 25(9):569–597, 2008.

[11] J. Levinson, J. Askeland, J. Becker, J. Dolson, D. Held, S. Kammel, J. Z. Kolter, D. Langer, O. Pink, V. Pratt, et al. Towards fully autonomous driving: Systems and algorithms. In *Intelligent Vehicles Symposium (IV), 2011 IEEE*, pages 163–168. IEEE, 2011.

[12] U. Franke, D. Gavrila, S. Gorzig, F. Lindner, F. Puetzold, and C. Wohler. Autonomous driving goes downtown. *IEEE Intelligent Systems and Their Applications*, 13(6):40–48, 1998.

[13] S. Thrun, W. Burgard, and D. Fox. *Probabilistic robotics*. MIT press, 2005.

[14] C. Linegar, W. Churchill, and P. Newman. Made to measure: Bespoke landmarks for 24-hour, all-weather localisation with a camera. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, Stockholm, Sweden, May 2016.

[15] U. Muller, J. Ben, E. Cosatto, B. Flepp, and Y. L. Cun. Off-road obstacle avoidance through end-to-end learning. In *Advances in neural information processing systems*, pages 739–746, 2006.

[16] A. Geiger, P. Lenz, and R. Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.

[17] W. Maddern, G. Pascoe, C. Linegar, and P. Newman. 1 year, 1000 km: The oxford robotcar dataset. *The International Journal of Robotics Research*, 36(1):3–15, 2017.

[18] D. A. Pomerleau. Alvinn: An autonomous land vehicle in a neural network. In *Advances in neural information processing systems*, pages 305–313, 1989.

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

[20] M. Deisenroth and C. E. Rasmussen. Pilco: A model-based and data-efficient approach to policy search. In *Proceedings of the 28th International Conference on machine learning (ICML)*, pages 465–472, 2011.

[21] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989.

[22] D. P. Kingma and M. Welling. Auto-encoding variational bayes. In *The International Conference on Learning Representations (ICLR)*, 2014.

[23] D. J. Rezende, S. Mohamed, and D. Wierstra. Stochastic backpropagation and approximate inference in deep generative models. In *Proceedings of the 31st International Conference on machine learning (ICML)*, 2014.

[24] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. Continuous control with deep reinforcement learning. In *International Conference on Learning Representations (ICLR)*, 2016.

[25] R. S. Sutton, D. A. McAllester, S. P. Singh, and Y. Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Advances in neural information processing systems*, pages 1057–1063, 2000.

[26] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International Conference on Learning Representations (ICLR)*, 2016.

[27] T. Schaul, J. Quan, I. Antonoglou, and D. Silver. Prioritized experience replay. In *International Conference on Learning Representations (ICLR)*, 2015.

[28] G. E. Uhlenbeck and L. S. Ornstein. On the theory of the brownian motion. *Phys. Rev.*, 36: 823–841, Sep 1930.

[29] A. Kendall, Y. Gal, and R. Cipolla. Multi-task learning using uncertainty to weigh losses for scene geometry and semantics. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[30] T. Zhou, M. Brown, N. Snavely, and D. G. Lowe. Unsupervised learning of depth and ego-motion from video. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[31] K. Bousmalis, N. Silberman, D. Dohan, D. Erhan, and D. Krishnan. Unsupervised pixel-level domain adaptation with generative adversarial networks. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[32] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz. Trust region policy optimization. In *International Conference on Machine Learning*, pages 1889–1897, 2015.

[33] M. Hausknecht and P. Stone. Deep recurrent q-learning for partially observable mdps. *CoRR, abs/1507.06527*, 2015.

[34] M. Igl, L. Zintgraf, T. A. Le, F. Wood, and S. Whiteson. Deep variational reinforcement learning for pomdps. In *Proceedings of the 28th International Conference on machine learning (ICML)*, 2018.

[35] G. Farquhar, T. Rocktäschel, M. Igl, and S. Whiteson. Treeqn and atreec: Differentiable tree planning for deep reinforcement learning. In *International Conference on Learning Representations (ICLR)*, 2018.

[36] D. Ha and J. Schmidhuber. World models. *CoRR*, abs/1803.10122, 2018.