2019 IEEE Intelligent Transportation Systems Conference (ITSC)
Auckland, NZ, October 27-30, 2019

End-to-end Reinforcement Learning for Autonomous Longitudinal Control Using Advantage Actor Critic with Temporal Context

Sampo Kuutti, Richard Bowden, Harita Joshi, Robert de Temple, Saber Fallah

Abstract— **Reinforcement learning has been used widely for autonomous longitudinal control algorithms. However, many existing algorithms suffer from sample inefficiency in reinforcement learning as well as the jerky driving behaviour of the learned systems. In this paper, we propose a reinforcement learning algorithm and a training framework to address these two disadvantages of previous algorithms proposed in this field. The proposed system uses an Advantage Actor Critic (A2C) learning system with recurrent layers to introduce temporal context within the network. This allows the learned system to evaluate continuous control actions based on previous states and actions in addition to current states. Moreover, slow training of the algorithm caused by its sample inefficiency is addressed by utilising another neural network to approximate the vehicle dynamics. Using a neural network as a proxy for the simulator has significant benefit to training as it reduces the requirement for reinforcement learning to query the simulation (which is a major bottleneck) in learning and as both reinforcement learning network and proxy network can be deployed on the same GPU, learning speed is considerably improved. Simulation results from testing in IPG CarMaker show the effectiveness of our recurrent A2C algorithm, compared to an A2C without recurrent layers.**

I. Introduction

Autonomous vehicles have been identified as the solution to many problems in current transportation systems. Reduced pollution, greatly increased safety, improved traffic flow and passenger comfort have been reported as the main advantages of autonomous vehicles [1]–[3]. Early autonomous vehicles attempted to control the car with rule-based systems, where the perception, planning, and control are achieved separately [4]–[6]. However, such approaches require time intensive hand-tuning of parameters and often fail to generalise to the wide variety of operational conditions the vehicle might face on the road. Currently, deep learning techniques have gained significant interest due to the ability of deep neural networks to generalise previously learned rules to new scenarios [7].

Reinforcement learning algorithms have been favoured in autonomous longitudinal control systems, due to their ability to generalise to new scenarios. State-of-the-art results have been achieved in this domain using actor-critic algorithms. Zhao *et al.* [8] proposed a reinforcement learning algorithm for longitudinal control that trained an actor-critic network using ranging sensor readings as input to the network to output the vehicle's desired acceleration, which was then achieved with a low level controller. Advantage actor critic algorithms with recurrent network architectures have also been used for end-to-end reinforcement learning for race driving with discrete action spaces by Mnih *et al.* [9] in a TORCS simulator and Jaritz *et al.* [10] in the World Rally Championship 6 game. While these works demonstrate that reinforcement learning can learn to drive in an end-to-end manner, for a system to be potentially deployable in a real autonomous vehicle, continuous action values should be leveraged for better performance.

However, a common problem with reinforcement learning in longitudinal control, is that the learned policy results to jerky behaviour [11]. Without any temporal context given, the agent has to decide the current action without any consideration for the previous actions, sometimes leading to rapid switching between the throttle and brake pedals. This occurs as the physics of the simulator effectively smooths the input control signals to produce smooth vehicle motion. However, in reality this type of behaviour would pose a problem both in terms of safety as well as occupant comfort if used in an autonomous vehicle, not to mention it may be damaging to the vehicle. Therefore, we propose to use a recurrent network architectures to solve this problem, and obtain smoother pedal behaviour. Introducing temporal context into the reinforcement learning model allows the agent to consider past actions and states when estimating the action at the current time step. The combination of continuous action spaces and temporal knowledge then allows the network to predict smoother and safer control outputs.

Another downside with many current reinforcement learning algorithms is their sample inefficiency [12], which leads to long training times. This makes training and optimising reinforcement learning algorithms more time intensive and can limit testing different parameters or approaches. The use of recurrent architecture exacerbates the problem. Here, we propose a novel solution to mitigate this and speed up training significantly by training another neural network with supervised learning to estimate the vehicle response to control actions from the longitudinal control model. The supervised learning network was trained with 45 hours of collected driving data from IPG CarMaker [13], a popular vehicle dynamics simulation software. By using the supervised network as a proxy for the simulator during training, costly access to the simulator can be eliminated during reinforcement learning. Furthermore, as the proxy simulator network can be deployed on the same GPU as the Advantage Actor Critic (A2C) network, significant speed benefits can be

Sampo Kuutti and Saber Fallah are with the Connected and Autonomous Vehicles Lab, University of Surrey, Guildford, GU2 7XH, UK. Email: {s.j.kuutti, s.fallah}@surrey.ac.uk

Richard Bowden is with the Centre for Vision, Speech and Signal Processing, University of Surrey, GU2 7XH, UK. Email: r.bowden@surrey.ac.uk

Harita Joshi and Robert de Temple are with Jaguar Land Rover Limited, Coventry, CV3 4LF, UK. Email: {hjoshi3, rdetempl}@jaguarlandrover.com

978-1-5386-7024-8/19/$31.00 ©2019 IEEE

obtained during training.

The contributions of this paper are two-fold. Firstly, we present an implementation of an autonomous longitudinal control system, using actor-critic reinforcement learning with a recurrent network architecture introducing temporal context in the model during learning. The system learns autonomous vehicle longitudinal control in an end-to-end fashion mapping ranging sensor readings and host vehicle states to pedal signals in continuous action space, maintaining a safe distance from the lead vehicle. The performance is compared with and without the recurrent layer, demonstrating an improvement in performance when temporal context is used. Secondly, we show that we can train our agent using a supervised network, which has been trained to estimate vehicle dynamics, thereby accelerating the training process of the reinforcement learning algorithm. The performance of the trained agent is then validated in IPG CarMaker.

The remainder of the paper is structured as follows. Section II introduces the necessary background theory on reinforcement learning and actor critic algorithms. Section III describes the reinforcement learning algorithm used for the longitudinal control of an autonomous vehicle as well as the training framework. Section IV presents the simulation results of the trained algorithm. Finally, concluding remarks are presented in Section V.

II. Reinforcement Learning

In reinforcement learning, an agent interacts with an environment and aims to learn from its own actions. At each time-step t, the agent observes a set of states s_t, takes an action a_t from a possible set of actions $\mathcal{A}$ according to its policy $\pi(s_t)$. The agent then observes a new set of states s_{t+1} and receives a reward r_t according to a reward function $\mathcal{R}$. The agent then tries to maximise the total accumulated return R_t. Reinforcement learning algorithms can generally be divided into three groups [14]: value based, policy gradient, and actor-critic methods. Value based methods learn a state-action value function $Q(s_t, a_t) = \mathbb{E}[R_t|s_t = s, a]$ which maps each possible action to a value for each state. For continuous states and actions, this is typically approximated. The action is then chosen by deterministic policy maximising the state-action value function as

$$\pi(s_t) = \underset{a}{\operatorname{argmax}}\, Q(s_t, a_t) \tag{1}$$

The downside is that there is no guarantee on the optimality of the resulting policy [15]–[17]. On the other hand, policy gradient methods do not estimate the value function. Instead, policy gradient methods parametrise the policy $\pi(s)$ by the parameter vector θ, and calculate the gradient of the cost function with respect to θ and a cost function J as

$$\bigtriangledown_\theta J = \frac{\partial J}{\partial \pi_\theta} \frac{\partial \pi_\theta}{\partial \theta} \tag{2}$$

and the parameters are then updated in the direction of the gradient. Policy gradient methods generally have the advantage of improved convergence, but have the disadvantage of high variance in the policy gradient [18], [19].

Actor-Critic methods are hybrid methods combining both value based and policy based methods, in order to utilise the advantages of both techniques. An Actor-Critic algorithm uses two neural networks. The Critic network estimates how good being in any given state is, i.e. the value function $V(s) = \mathbb{E}[R_t|s_t = s]$. The Actor network estimates the optimal policy function $\pi^*(s_t)$, which maps each observed state to an action. Estimating the value function is useful, since we can not know the actual value of the actions taken (i.e. total rewards for an episode) until the episode has finished. The critic network allows the algorithm to estimate the value of the actions taken during training. Therefore, unlike policy gradient methods where total rewards are calculated at the end of the episode so that network weights can be updated, estimating the value function lets us update the network weights before the episode is finished, such that we can update the weights multiple times each episode [9]. During training, the agent interacts with the environment, and based on the actions chosen in each state, the agent is given a reward. The estimated value function and the reward are then used to update the weights of both networks as shown in Fig. 1.

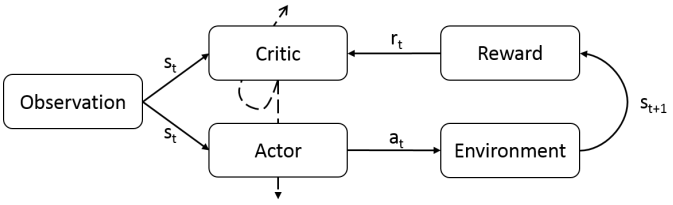

Fig. 1. An Actor-Critic agent interacting with the environment. Dashed lines represent a parameter update.

III. Longitudinal Control with Reinforcement Learning

A. A2C Algorithm

The A2C [9] algorithm uses an advantage function instead of the value function for updating the weights, improving the stability during training. The advantage function estimates how much better the chosen action for a given state is compared to the average action at that state.

$$A(s_t, a_t) = Q(s_t, a_t) - V(s_t) \tag{3}$$

Due to difficulty in estimating the state-action value function $Q(s_t, a_t)$, the advantage function is typically estimated by some advantage estimator, such as the temporal-difference (TD) error [20] used here:

$$A(s_t, a_t) \approx TD = \sum_{k}^{n-1} \gamma^k r_{t+k} + \gamma^n V(s_{t+n}) - V(s_t) \tag{4}$$

where $\gamma \in$ [0,1) is the discount factor used to prioritise immediate rewards over future rewards.

When used in a discrete action space, the actor network in A2C outputs a probability for each discrete action, where each probability corresponds to how confident the agent is

that the corresponding action is the correct one in its current state. However, when working in a continuous action space a further modification is needed. Here, the actor network has to be modified to generate two outputs which are used to choose the action in each state [9]. The actor network estimates the action value μ and the estimated variance σ^2. This is transformed into a Gaussian probability distribution, where μ is the location (or mean value) and σ is the standard deviation of the distribution. The control action is then sampled from the generated Gaussian distribution.

For the continuous action model, the value and policy losses, $\mathcal{L}_v$ and $\mathcal{L}_\pi$, are then given by

$$\mathcal{L}_v = (A(s_t, a_t))^2 \tag{5}$$

$$\mathcal{L}_\pi = -log(\pi(a_t|s_t))A(s_t, a_t) - \beta H(\pi(s_t)) \tag{6}$$

where β is the entropy coefficient and $H(\pi(s_t))$ is the entropy added to encourage exploration in the policy, calculated as

$$H(\pi(s_t)) = \frac{1}{2}(log(2\pi\sigma^2) + 1) \tag{7}$$

The inputs to the networks are host vehicle velocity v_h, host acceleration $\dot{v}_h$, relative velocity to lead vehicle v_{rel}, and time headway to lead vehicle t_h. The possible actions a_t represent the vehicle pedal values, normalised to a single parameter such that $a_t \in$ [-1, 1], where positive values signal the use of the gas pedal and negative values correspond to the use of the brake pedal.

Both networks are decoupled such that they take the observed states $s_t = [v_h, \dot{v}_h, v_{rel}, t_h]^{\mathrm{T}}$ as an input. This is then followed by three fully connected feed-forward hidden layers in the actor network. The last feed-forward hidden layer is followed by the recurrent layer, which is fully connected to the two outputs, μ and σ^2. In comparison, the critic network has one hidden layer only, fully connected to the output layer, which gives the value estimate $V(s)$. The hidden neurons all use a Relu-6 activation [21], the μ uses a tanh activation, the σ^2 uses a softplus activation, whilst the value estimate has a linear activation.

The recurrent layer consists of Long Short-Term Memory (LSTM) [22] cells, each fully connected to the output layer. The architecture of a basic LSTM cell is shown in figure 2.

The LSTM cell takes the cell state, C_{t-1}, and cell output, h_{t-1}, from previous time-steps as well as the output of the previous layer, x_t, as an input to the cell. The inputs are then connected to three gates within the cell: the forget gate, input gate, and output gate (from left to right in Fig. 2). The outputs of each gate, cell output, and new cell state are then given as shown below.

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \tag{8}$$

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \tag{9}$$

$$\hat{c}_t = tanh(W_c \cdot [h_{t-1}, x_t] + b_c) \tag{10}$$

$$C_t = f_t C_{t-1} + i_t \hat{c}_t \tag{11}$$

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \tag{12}$$

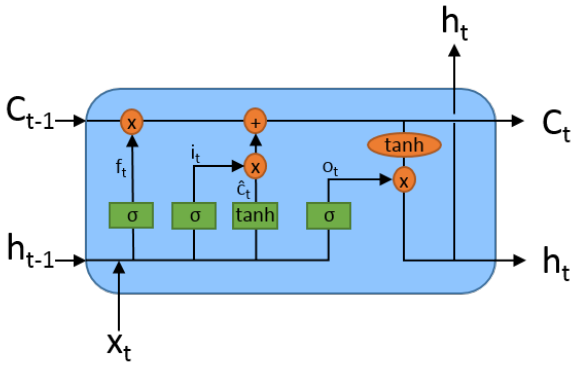

Fig. 2. The architecture of a LSTM cell in the recurrent layer. The orange round connectors represent pointwise operations and green rectangles represent hidden layers. The connections to the left and right represent connections to the previous and next time-steps, respectively.

TABLE I

FINAL NETWORK HYPERPARAMETERS.

Parameter	Value
No. hidden layers (actor)	3
No. neurons per hidden layer (actor)	50
No. of LSTM units (actor)	16
No. hidden layers (critic)	1
No. neurons per hidden layer (critic)	200
Learning rate (actor), η_{actor}	$1x10^{-4}$
Learning rate (critic), η_{critic}	$1x10^{-3}$
Discount factor, γ	0.99
Entropy coefficient, β	$1x10^{-3}$
Minibatch size	64
Trajectory length	80
Trauma ratio, α_{trauma}	1/64
RMSProp ϵ	$1x10^{-10}$
RMSProp decay α	0.9
RMSProp momentum	0.0

$$h_t = o_t tanh(C_t) \tag{13}$$

Where W_x is weight matrix for layer x, b_x is the bias vector for layer x, and $\sigma(x)$ is the sigmoid function.

After tuning the network hyperparameters using a grid search, the final network architecture uses the hyperparameters presented in Table I.

The network parameters θ are then updated during training by an RMSProp optimiser [23] as

$$\bar{g}^2_{t+1} = \alpha \bar{g}^2_{t-1} + (1 - \alpha) g^2_t \tag{14}$$

$$\theta_{t+1} = \theta_t - \eta \frac{g_t}{\sqrt{\bar{g}^2_{t+1} + \epsilon}} \tag{15}$$

where g is the gradient, $\bar{g}$ is the running mean of the gradient, η is the learning rate, α is the decay rate, and ϵ is a small value added to avoid division by zero. The update steps for the actor and critic networks were also decoupled, such that the actor network is updated using policy loss $\mathcal{L}_\pi$ and learning rate η_{actor}, whilst the critic network is updated using the value loss $\mathcal{L}_v$ and learning rate η_{critic}.

To encourage the agent to maintain a 2s time headway, a reward function based on time headway and time headway

derivative was defined. This reward function is based on the similar function used in [11], where the time headway term encourages the agent to maintain a headway close to 2s, while the headway derivative term rewards the agent for taking actions which bring it closer to the ideal headway, as shown in Figure 3. A large negative reward is applied when a crash occurs to discourage the agent from crashing into the lead vehicle, and a large positive reward is given when the agent is close to the ideal time headway.

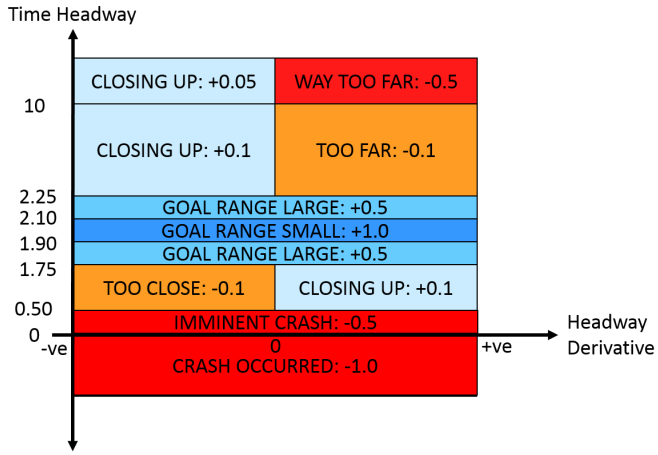

Fig. 3. Reward function for longitudinal control. The function consists of two terms, time headway and time headway derivative; the former encourages the agent to maintain the ideal time headway of 2s, whilst the latter encourages actions which bring it closer to the ideal time headway.

To improve stability of training and avoid catastrophic forgetting [24], [25], experience replay was used to update network weights in addition to the on-policy updates. During training, the agent records, at each time step, experiences $e_t = [s_t, a_t, r_t, s_{t+1}]$ in the replay memory $\mathcal{D}$. When updating the weights, a minibatch of experiences are then uniformly sampled from $\mathcal{D}$. Since the network uses LSTMs for temporal context, these minibatches are sampled as sequenced trajectories of experiences. To avoid mismatches with cell states, the cell state is initialised as a zero state at the beginning of the update step [26]. In addition to the standard experience replay, a second set of experiences called trauma memory [27] was used to store trajectories which lead to collisions. The trauma memory was used to ensure the network learns to avoid collisions, since such events occur rarely during training, using only random sampling from the experience replay would rarely include these events in the minibatches. When picking experiences for the minibatch, the ratio α_{trauma} describes the ratio of trauma memory samples to experience replay samples, which was tuned to $1/64$ during the hyperparameter optimisation.

B. Training

1) Lead Vehicle and Environment Set-up: The training was broken down into 5-minute training episodes, where the episode ends after the 5 minutes or if a crash occurred. At the start of each episode a road friction coefficient value between 0.4 and 1.0 (with increments of 0.025) was chosen. The lead vehicle performed randomly chosen manoeuvrers, where its velocity was in the range $v_{\text{lead}} \in$ [17, 40]m/s, and the acceleration was limited to $\dot{v}_{lead} \in$ [-2, 2]m/s^2. The exception to this was emergency braking manoeuvrers, which the lead vehicle performed, on average, once an hour. During emergency braking the deceleration was chosen between $\dot{v}_{lead} \in$ [-6, -3]m/s^2. The aim of the agent was to maintain a 2s time headway from the vehicle in front, whilst the lead vehicle varied its velocity under different manoeuvrers.

2) Proxy Network for Vehicle Dynamics: To integrate the Python based Tensorflow [28] framework used for reinforcement learning with the IPG CarMaker simulation platform, a C-based library for communicating with CarMaker's Applications Online (APO) communication service was developed. The functionality of the library was compiled into a dynamic link library and and an import library for use by external applications, which were then imported to Python using the ctypes module. The implemented integration allows communication with CarMaker through the APO service over TCP and UDP sockets, allowing the reinforcement learning agent to observe states and control the host vehicle in CarMaker, while all learning is handled in Tensorflow. However, this creates some delays in the communication pipeline. The agent has to query states from CarMaker, choose an action based on the learned policy, send the control command to CarMaker, and then read the new states through the APO communication service. This leads to 40ms of simulated time between each control output when the simulator is running at real time. However, at maximum simulation speed ($\sim$20x real time), up to 300ms of simulated time occurs between control commands received by the simulator. Further delays can exist when the network weights are updated using large batches. Therefore, for training, the simulator was constrained to running at real time. This poses a problem for training, since due to the sample inefficiency of actor-critic reinforcement learning, the network needs a relatively large amount of experiences to converge to an optimal policy.

Similar problems have been overcome in robotics applications by training a Gaussian process as a simulator proxy to estimate the simulator using data sampled from the Gaussian process [29], [30]. However, Gaussian processes scale poorly to large datasets [31]. Another potential solution would be to use a simplified vehicle model to speed up simulation, but this would reduce the model accuracy in certain scenarios. Instead, a second neural network, the proxy network, was trained using supervised learning to estimate the vehicle response to the control actions of the A2C algorithm. This has the benefit that the neural network can take advantage of the GPU computing to speed up the training process. The proxy network was trained on 45 hours of collected driving data from IPG CarMaker, resulting in a dataset of 2,364,041 time steps, of which 1,418,424 are in the training set.

The network has 3 hidden layers, with 100 neurons each. It takes host vehicle velocity v, acceleration $\dot{v}$, road coefficient of friction CoF, and pedal actions from current and last time-step, y_t and y_{t-1}, as inputs. The output of the network

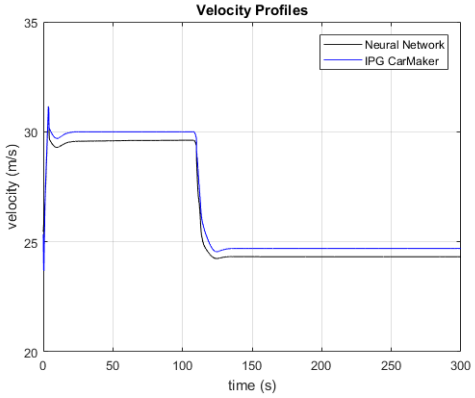

Fig. 4. Vehicle state estimation with IPG CarMaker (blue) and proxy network (black).

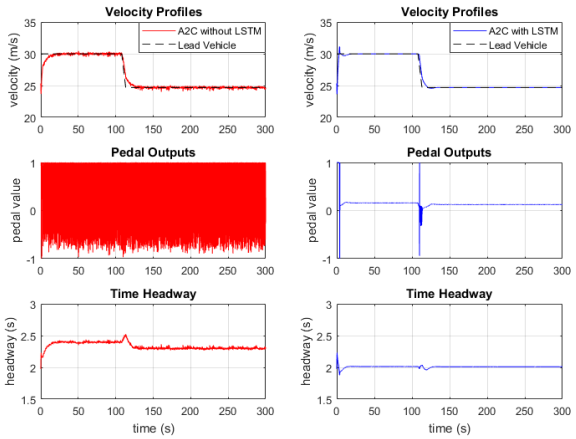

Fig. 5. A2C controllers without LSTM (left) and with LSTM (right) following a lead vehicle with constant velocities and a deceleration at CoF = 1.0.

is the host vehicle velocity at the next time-step, v_{t+1}, using time-steps of 40ms. The network was then trained using supervised learning with the training data collected from IPG CarMaker for 1,000,000 training steps with a batch size of 1000 and a learning rate of $1\text{x}10^{-6}$. The model with the lowest validation loss was then chosen, resulting in a model with mean absolute percentage error of 0.1893% and 0.2248% for velocity and acceleration estimations, respectively, as measured on the validation data. A typical highway driving scenario can be seen in Fig. 4, with a comparison of vehicle states simulated in IPG CarMaker and estimated with the trained proxy network, using the same pedal values as input. Here, it can be seen that the neural network can estimate the vehicle response from the pedal values, although a slight underestimation can be seen at the start of the episode, which leads to the maximum absolute velocity error of 0.405m/s. Therefore, the neural network can estimate the vehicle response to the pedal actions sufficiently well to be used for training the A2C network, and final validation can then be done in IPG CarMaker to ensure the validity of the results.

Once the supervised network was trained to work as an estimation of the IPG CarMaker simulation environment, it was used to train the A2C algorithm. The A2C agent was trained for 2,500 episodes which equals to over 200 hours of driving. The training was completed using the proxy network in under 19 hours, with both the proxy network and A2C network deployed on the same GPU. Once trained, the performance of the trained A2C agent was then tested in IPG CarMaker for which the results can be seen in the next section.

IV. Simulation Results

Once the training phase with the proxy network was complete, the performance of the A2C network was validated in various driving scenarios. All simulation experiments presented here were performed in the IPG CarMaker simulation environment to utilise its superior accuracy and ensure the validity of the results, by combining the IPG CarMaker platform with the A2C algorithm in Tensorflow as described previously in Section III-B. The A2C algorithm was tested for 10 hours on highway driving with and without the LSTM layer, for a total of 20 hours. Both networks used the same parameters as shown previously in Table I (with the exception of the LSTM layer) and were trained for 2,500 episodes. As an additional baseline, the default driver in CarMaker, IPGDriver, is used to compare the A2C networks' performance against a rule-based control strategy. The section starts with discussing example episodes to analyse the individual performance of both network architectures, followed by discussion on the overall performance of each 10 hour testing run.

A typical highway driving scenario with the lead vehicle decelerating to a lower velocity in dry road conditions can be seen in Fig. 5. The subplots on the left show the performance without the LSTM layer and the subplots on the right with the LSTM. From these results it is clear that the LSTM layer helps the agent use smoother pedal values to control the vehicle. While both networks appear to have learned a reasonable driving policy, the LSTM architecture shows a significantly smoother driving style and maintains a closer headway to the 2s target, with a mean t_h of 2.0114s compared to 2.3280s without the LSTM. A highway driving scenario in wet conditions can be seen in Fig. 6 where the lead vehicle periodically accelerates and decelerates. Again, the network with LSTM shows better performance in terms of headway error and significantly smoother driving style.

As an example of safety critical scenarios tested in the simulations, an emergency braking scenario in wet road conditions can be seen in Fig. 7. The lead vehicle starts the scenario with a constant velocity of 26m/s, and at t = 17s begins to decelerate at 5m/s^2 to a velocity of 12m/s. To demonstrate the performance against a rule-based control strategy, the behaviour of the IPGDriver is also shown here. All controllers successfully brake in time to avoid a collision with the lead vehicle. In comparison to the IPGDriver, the networks maintain a significantly safer distance and headway

TABLE II

A2C PERFORMANCE WITH 10 HOURS OF TESTING.

Parameter	IPGDriver	A2C without LSTM	A2C with LSTM
min. x_{rel}	10.737 m	6.851 m	7.780 m
mean x_{rel}	75.16 m	68.89 m	58.01 m
max. v_{rel}	13.90 m/s	8.053 m/s	7.891 m/s
mean v_{rel}	0.187 m/s	0.0656 m/s	0.0289 m/s
min. t_h	1.046 s	0.9089 s	1.114 s
mean t_h	2.5471 s	2.379 s	2.007 s
collisions	0	0	0

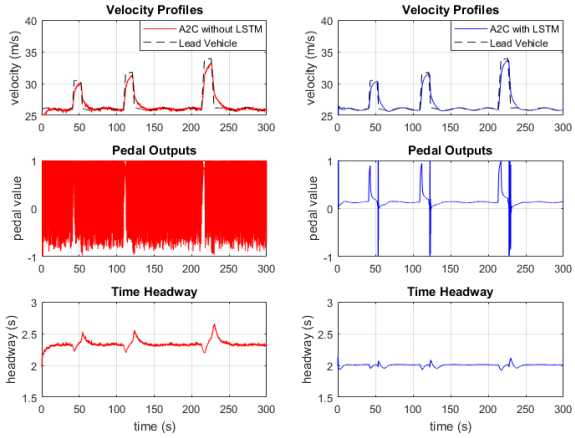

Fig. 6. A2C controllers without LSTM (left) and with LSTM (right) following a lead vehicle with varying velocity profile at CoF = 0.55.

errors. More importantly, the results show the network with LSTM maintains a safer distance from the lead vehicle. Minimum distances and time headways for the LSTM network are 20.744m and 1.727s compared to 19.476m and 1.558s without LSTM.

The results for the complete 10 hour highway driving runs can be seen in Table II for IPG Driver and both networks with and without the LSTM layer. The results show both networks have learned to control the vehicle and maintain a reasonable distance from the lead vehicle. These results demonstrate that even the simple network architecture without temporal context offers benefits over the rule-based controller. However, it can be seen that the LSTM layer has significantly improved the performance of the controller. The aim of the agent was to maintain a 2s time headway from the lead vehicle, and with the LSTM the mean headway was 2.007s compared to 2.379s without the LSTM, showing significant improvement using the recurrent layer. Moreover, previous results show that the LSTM layer helps the agent plan a smoother control policy using temporal context, reducing the jerkiness of the vehicle. Although some rare rapid changes from the gas pedal to the brake pedal could still be seen, this type of behaviour was significantly reduced compared to the non-LSTM network architecture. This would improve the comfort of any occupants and the overall safety of the autonomous vehicle significantly.

V. CONCLUDING REMARKS

In this paper, a reinforcement learning algorithm for autonomous vehicle longitudinal control based on the A2C algorithm was presented. The proposed algorithm leverages multiple recent advancements in reinforcement learning and longitudinal control algorithms. The recurrent layers were introduced into the network architectures to provide temporal context and improve performance. Using continuous action spaces together with temporal context the network could output smoother and safer control signals. Sequenced experience replay, together with trauma memory, boosted the stability of training and safety of the learned control policy. The proposed solution was compared to a network without recurrent layers, and shown to improve the performance (in terms of time headway errors) and smoothness of the continuous control actions. The agent was trained to maintain a 2s time headway from the lead vehicle in a highway driving environment. A variety friction coefficients and lead vehicle manoeuvrers were used to increase the complexity of the task. Moreover, the training framework using a proxy simulator was shown to accelerate the training process, mitigating the sample inefficiency problem of reinforcement learning. The A2C network was trained using a supervised learning network trained to estimate the host vehicle's response to pedal actions chosen by the reinforcement learning agent. This sped up the training process. All validation experiments were then performed in a more accurate simulation environment, IPG CarMaker, to ensure the validity of the results. A further advantage of the A2C algorithm is that it also enables distributed training, where multiple agents can collect experiences, further speeding the training.

ACKNOWLEDGMENT

This work was supported by the UK-EPSRC grant EP/R512217/1 and Jaguar Land Rover.

REFERENCES

[1] Department for Transport, "Research on the Impacts of Connected and Autonomous Vehicles (CAVs) on Traffic Flow: Summary Report," 2017. [Online]. Available: https://www.gov.uk/government/uploads/system/uploads/attachment_data/file/530091/impacts-of-connected-and-autonomous-vehicles-on-traffic-flow-summary-report.pdf

[2] U. Montanaro, S. Dixit, S. Fallah, M. Dianati, A. Stevens, D. Oxtoby, and A. Mouzakitis, "Towards connected autonomous driving: review of use-cases," *Vehicle System Dynamics*, pp. 1–36, 2018.

[3] S. Kuutti, S. Fallah, K. Katsaros, M. Dianati, F. Mccullough, and A. Mouzakitis, "A survey of the state-of-the-art localization techniques and their potentials for autonomous vehicle applications," *IEEE Internet of Things Journal*, vol. 5, no. 2, pp. 829–846, 2018.

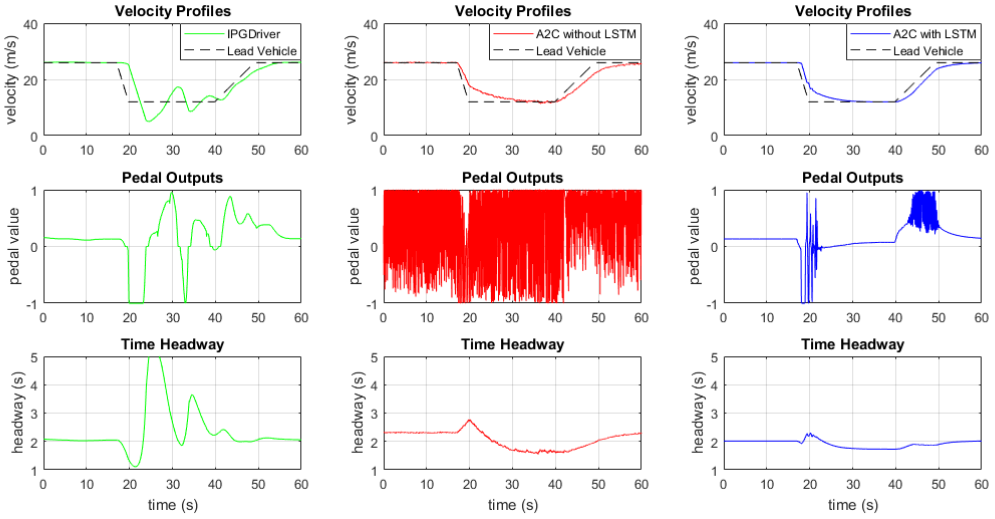

Fig. 7. IPGDriver (left) as well as A2C controllers without LSTM (middle) and with LSTM (right) following a lead vehicle performing an emergency braking manoeuvrer at CoF = 0.5.

[4] A. Sorniotti, P. Barber, and S. De Pinto, "Path tracking for automated driving: A tutorial on control system formulations and ongoing research," in *Automated Driving*. Springer, 2017, pp. 71–140.

[5] C. Urmson, J. Anhalt, D. Bagnell, C. Baker, R. Bittner, M. Clark, J. Dolan, D. Duggins, T. Galatali, C. Geyer *et al.*, "Autonomous driving in urban environments: Boss and the urban challenge," *Journal of Field Robotics*, vol. 25, no. 8, pp. 425–466, 2008.

[6] Z. Sun, G. Bebis, and R. Miller, "On-road vehicle detection: A review," *IEEE Transactions on Pattern Analysis & Machine Intelligence*, no. 5, pp. 694–711, 2006.

[7] D. Silver, J. A. Bagnell, and A. Stentz, "Learning Autonomous Driving Styles and Maneuvers from Expert Demonstration," in *Experimental Robotics*. Springer, Heidelberg, 2013, pp. 371–386.

[8] D. Zhao, Z. Xia, and Q. Zhang, "Model-free optimal control based intelligent cruise control with hardware-in-the-loop demonstration [research frontier]," *IEEE Computational Intelligence Magazine*, vol. 12, no. 2, pp. 56–69, 2017.

[9] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *International conference on machine learning*, 2016, pp. 1928–1937.

[10] M. Jaritz, R. De Charette, M. Toromanoff, E. Perot, and F. Nashashibi, "End-to-end race driving with deep reinforcement learning," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 2070–2075.

[11] C. Desjardins and B. Chaib-Draa, "Cooperative adaptive cruise control: A reinforcement learning approach," *IEEE Transactions on intelligent transportation systems*, vol. 12, no. 4, pp. 1248–1260, 2011.

[12] Z. Wang, V. Bapst, N. Heess, V. Mnih, R. Munos, K. Kavukcuoglu, and N. de Freitas, "Sample efficient actor-critic with experience replay," *arXiv preprint arXiv:1611.01224*, 2016.

[13] IPG Automotive GmbH, "Carmaker: Virtual testing of automobiles and light-duty vehicles," 2017. [Online]. Available: https://ipg-automotive.com/products-services/simulation-software/carmaker/

[14] V. R. Konda and J. N. Tsitsiklis, "On actor-critic algorithms," *SIAM journal on Control and Optimization*, vol. 42, no. 4, pp. 1143–1166, 2003.

[15] I. Grondman, L. Busoniu, G. A. Lopes, and R. Babuska, "A survey of actor-critic reinforcement learning: Standard and natural policy gradients," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 42, no. 6, pp. 1291–1307, 2012.

[16] L. Baird, "Residual algorithms: Reinforcement learning with function approximation," in *Machine Learning Proceedings 1995*. Elsevier, 1995, pp. 30–37.

[17] G. J. Gordon, "Stable function approximation in dynamic programming," in *Machine Learning Proceedings 1995*. Elsevier, 1995, pp. 261–268.

[18] 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*, 2000, pp. 1057–1063.

[19] M. Riedmiller, J. Peters, and S. Schaal, "Evaluation of policy gradient methods and variants on the cart-pole benchmark," in *Approximate Dynamic Programming and Reinforcement Learning, 2007. ADPRL 2007. IEEE International Symposium on*. IEEE, 2007, pp. 254–261.

[20] S. Bhatnagar, M. Ghavamzadeh, M. Lee, and R. S. Sutton, "Incremental natural actor-critic algorithms," in *Advances in neural information processing systems*, 2008, pp. 105–112.

[21] A. Krizhevsky and G. Hinton, "Convolutional deep belief networks on cifar-10," *Unpublished manuscript*, vol. 40, no. 7, 2010.

[22] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[23] T. Tieleman and G. Hinton, "Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude," *COURSERA: Neural networks for machine learning*, vol. 4, no. 2, pp. 26–31, 2012.

[24] R. M. French, "Catastrophic forgetting in connectionist networks," *Trends in cognitive sciences*, vol. 3, no. 4, pp. 128–135, 1999.

[25] J. Kirkpatrick, R. Pascanu, N. Rabinowitz, J. Veness, G. Desjardins, A. A. Rusu, K. Milan, J. Quan, T. Ramalho, A. Grabska-Barwinska *et al.*, "Overcoming catastrophic forgetting in neural networks," *Proceedings of the national academy of sciences*, vol. 114, no. 13, pp. 3521–3526, 2017.

[26] M. Hausknecht and P. Stone, "Deep recurrent q-learning for partially observable mdps," 2015. [Online]. Available: https://www.aaai.org/ocs/index.php/FSS/FSS15/paper/view/11673/11503

[27] H. Chae, C. M. Kang, B. Kim, J. Kim, C. C. Chung, and J. W. Choi, "Autonomous braking system via deep reinforcement learning," in *2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC)*. IEEE, 2017, pp. 1–6.

[28] Google, "Tensorflow: An open source machine learning framework for everyone," 2017. [Online]. Available: https://www.tensorflow.org/

[29] 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-11)*, 2011, pp. 465–472.

[30] R. McAllister and C. E. Rasmussen, "Data-efficient reinforcement learning in continuous state-action gaussian-pomdps," in *Advances in Neural Information Processing Systems*, 2017, pp. 2040–2049.

[31] H. Liu, Y.-S. Ong, X. Shen, and J. Cai, "When gaussian process meets big data: A review of scalable gps," *arXiv preprint arXiv:1807.01065*, 2018.