The Thirty-Second AAAI Conference
on Artificial Intelligence (AAAI-18)

Toward Deep Reinforcement Learning without a Simulator: An Autonomous Steering Example

Bar Hilleli, Ran El-Yaniv
Department of Computer Science
Technion – Israel Institute of Technology
Haifa, Israel
rani@cs.technion.ac.il
barh@campus.technion.ac.il

Abstract

We propose a scheme for training a computerized agent to perform complex human tasks such as highway steering. The scheme is designed to follow a natural learning process wherein a human instructor teaches a computerized trainee. It enables leveraging the weak supervision abilities of a (human) instructor who – while unable to perform the required task well herself – can provide coherent and learnable instantaneous reward signals to the computerized trainee. The learning process consists of three supervised elements followed by reinforcement learning. The supervised learning stages are: (i) supervised imitation learning; (ii) supervised reward induction; and (iii) supervised safety module construction. We implemented this scheme using deep convolutional networks and successfully created a computerized agent capable of autonomous highway steering in the well-known racing game *Assetto Corsa*. We demonstrate that the use of all components is essential to effectively carry out reinforcement learning of the steering task using vision alone, without access to a driving simulator internals, and operating in wall-clock time.

1 Introduction

Consider the task of designing a robot capable of performing a complex human task such as dishwashing, driving or ironing clothes. Although these tasks are natural for adult humans, designing a hard-coded algorithm for such a robot can be a daunting challenge. Obstacles include difficulties in accurately modeling the robot and its interaction with the environment, creating hand-crafted features from high-dimensional sensor data, and guaranteeing that the robot is able to adapt to new situations, to name just a few. In this paper, we propose a general scheme that combines several learning techniques to tackle such challenges. As a proof a concept, we implemented the scheme and applied it to the challenging problem of autonomous highway steering. To this end, we use one of the most popular computer racing games (Assetto Corsa)[1], and attempt to create a self-steering car in the sense that given raw image pixels (of the car racing game screen), we wish to output correct steering control commands for the steering wheel. We use Convolutional Neural Networks (CNNs) as mapping functions from high-dimensional data to control actions and to instantaneous reward signals. The choice of CNNs is based on their proven ability to extract informative features from images in the context of classification and control tasks (Mnih et al. 2015; Krizhevsky, Sutskever, and Hinton 2012), thus obviating the exhausting task of manually defining features. For example, in the work of Mnih et al. (2015), a CNN was successfully trained to predict desired control actions in the Atari 2600 domain, given high-dimensional pixel data.

Two of the most promising approaches for robot training are *Imitation Learning* (IL) and *Reinforcement Learning* (RL). In IL, a human demonstrator performs the desired task with the goal of teaching a (robotic) agent to mimic her actions (Argall et al. 2009). The demonstrations are used to learn a mapping from a given world state, s, received via sensors, to a desired action, a, consisting of instructions to the agent's controllers. In RL, the goal is to enable the agent to find its own policy, one that maximizes a value function defined in terms of certain guidance reward signals received during its interaction with the environment. IL and RL can be naturally combined, as was recently proposed by Taylor, Suay, and Chernova (2011). The idea is to initiate the reinforcement learning process with a policy learned during a preceding IL stage. This combined approach can significantly accelerate the RL learning process and minimize costly agent–environment interactions. In addition, deploying the RL algorithm after the IL stage compensates for the noisy demonstrations from which IL can suffer and extends the imitation strategy to previously unseen areas of the state space.

Noisy demonstrations in the IL stage typically result from inconsistent strategies adopted by the human demonstrator. Consequently, the performance of an IL agent is limited by the quality of the observed demonstration. Another drawback of IL is its susceptibility to 'distributional divergences', where a sequence of minor errors in the agent's behavior is compounded and leads to a state that is completely outside the scope of the training sample. The agent's behavior in such states can be arbitrary and possibly dangerous. RL has it flaws as well: it either requires a realistic simulation of the agent's interaction with the environment or requires operating the agent in a real-world environment, which can be quite costly. Moreover, the sample complexity of RL can be

[1] At the time of writing, this game is considered to be one of the most realistic computer racing environments.

large and the modeling of an effective reward function is often quite challenging, requiring expert insight and domain knowledge. The current state of affairs in robotic design using any technique clearly leaves much to be desired, where ultimate goals such as domestic robot housekeepers remain futuristic.

In safe RL (Garcıa and Fernández 2015), the learning agent seeks a policy that maximizes the long-term future reward received from the environment while respecting some safety constraints (e.g., damage avoidance). Two of the main approaches for integrating the safety concept into RL algorithms are: (i) transforming the optimization criteria to include a notion of risk (e.g., variance of return, worst-outcome); and (ii) modifying the agent's exploration process while utilizing prior knowledge of the task to avoid risky states. Our proposed method is related to the second approach. By incorporating a *safety module* into the RL algorithm, we are able to significantly reduce the number of accidents (mostly in the initial stage of the RL). Furthermore, the integration of the safety module into the RL allows us to improve the agent's exploration of the state space and speed up the learning. This enables the agent to avoid the pitfalls of random exploration policies such as ϵ-greedy, where the agent wastes time on unimportant regions of the state space that the optimal policy would never have encountered.

The proposed approach somewhat resembles the natural teaching–learning procedure used by humans to teach each other complex tasks. In the first stage, the student mimics the actions demonstrated by the instructor, after which the instructor provides real-time feedback and the student improves performance by both optimizing a policy as well as learning the feedback function itself. Then, the student continues to teach herself (without the instructor), using both the reward function previously induced by the instructor and reward signals from the environment.

There is a vast body of literature on IL (Argall et al. 2009) and RL (Kober, Bagnell, and Peters 2013), which we can not survey due to lack of space. The two closest works to ours are Taylor, Suay, and Chernova (2011) and Daniel et al. (2014). In the first, the authors showed that a preceding demonstration learning stage can significantly expedite the reinforcement learning process and improve the final policy performance in a simulated robot soccer domain. In the second, the authors proposed to learn a reward model in a supervised manner and used alternating steps of reward and reinforcement learning to continually improve the reward model and the agent's policy in a robotic arm grasping task. We believe our work is the first to propose and demonstrate a successful integration of all four learning components.

We propose and apply a learning scheme for complex tasks that combines four components: IL, Reward Induction (RI), Safety Network learning, and RL (see supplementary video). First, by performing imitation learning, we get a reasonable initial performance by the agent. In this stage, both the actions of a human demonstrator and the corresponding game images are recorded while she plays the game. A policy network, denoted P_θ, is trained in an SL manner using this data. The goal of this step is to generate an agent capable of operating in the environment without too much risk (e.g., without damaging itself or the environment). Second, we *learn* a reward network, denoted R_θ, (to be used later by the RL procedure) from instructor feedback generated while observing the agent operating in the environment using the initial IL policy. The reward network receives an image and outputs a number in the range $[-1, 1]$ that indicates the instantaneous reward for being in that state. We call this method *reward induction* since the reward function is induced (in an SL manner) from a finite set of labeled examples obtained from a human instructor. Third, we learn a *safety network*, denoted S_θ, to be integrated in the RL procedure. Finally, in the RL stage, the reward network and the safety network are used to teach the agent a policy without any human supervision. In this stage, the Double Deep Q-learning (Hasselt 2010; Hasselt, Guez, and Silver 2016) (DDQN) RL algorithm is used to train a Q-network, denoted Q_θ. The reward signal used in the RL procedure is constructed from the reward network's output *only*. The learned policy network's parameters from the imitation part are used to initialize the Q-network's parameters.

We emphasize that our entire system is implemented *without* any access to the internal state of the game simulator (which is a purchased executable code). This is in contrast to most previously published works on computerized autonomous driving, which were conducted in a simulation environment, allowing access to the internal states of the simulator (e.g., TORCS, Wymann et al. 2000). These states contain valuable parameters such as the car's distance from the roadside or its angle with respect to the road. Thus, when an open simulator is available, such parameters can be extracted and utilized in the learning process, as can other reward information (Zhang and Cho 2016; Loiacono et al. 2010; Munoz, Gutierrez, and Sanchis 2009; Chen et al. 2015). More importantly, the lack of an open simulator in our setting means that all our learning procedures, including reinforcement learning, must be executed slowly in *wall-clock time* (as in real driving), as opposed to the super-fast learning that can typically be achieved using a simulator. The latter issue is one of the most significant limitations when applying RL for real-world tasks. Quoting Yann LeCun, "But it [RL] doesn't really work in the real world ... the main reason for this is ... you cannot run it faster than real-time" (LeCun 2017). Our work aims at bridging this gap by demonstrating that non-trivial RL *without* any access to the internal state of the game simulator can be effectively implemented when supported with an appropriate foundation of supervised learning.

2 Imitation learning

Imitation learning, also known as behavioral cloning or learning from demonstrations, aims at finding a mapping $f^\star : s \rightarrow a$ from a given world state, s, to a desired action a. This mapping is typically termed "policy." Robot learning using mimicry is an old idea, conceived decades ago (Hayes and Demiris 1994; Argall et al. 2009). While being an excellent technique for achieving reasonable performance, this approach by itself is limited. Clearly, the perfor-

mance of an agent trained in this manner is upper bounded by the level of instruction. Moreover, if the training sample is not sufficiently diverse/representative, the agent will not be exposed to unexpected difficult states, and can perform very poorly and unpredictably when such states are encountered in production. Finally, labeled samples obtained from human demonstrations are prone to labeling noise.

Noting these limitations, we use imitation learning in our scheme only to achieve a basic performance level, which will allow the agent to perform the required task without damaging itself or the environment. In our setting, world states are game images, and actions are keyboard keys pressed by a human demonstrator while playing the game. The Assetto Corsa game offers the option of connecting a steering wheel controller; we have not utilized this option and note that the use of such a wheel controller should improve the driving performance of the learning agent.

In this stage of the scheme we train a policy network that maps raw image pixels to steering commands of left/right. The policy network, which is composed of four convolutional layers and two fully connected layers, is trained and evaluated. The last layer of the network has three nodes that correspond to the actions: 'No Action', 'Left' and 'Right'. A softmax nonlinearity is applied to the last layer. A full description of the network's architecture is given in the supplementary material.

A training sample of state–action pairs, $D = \{(s_i, a_i)\}_i$, is recorded while a human expert plays the game. A detailed explanation of the recording process is given in Section 5 (in the experiments described below, D contained approximately 70,000 samples, equivalent to two hours of human driving). We then train a policy network using D, denoted P_θ. The negative log-likelihood is used as a loss function for training P_θ,

$$NLL(\theta, \mathcal{D}) = -\frac{1}{|\mathcal{D}|} \sum_{i=0}^{|\mathcal{D}|} log P_\theta \left(a_i | s_i\right),$$

where P_θ receives the state s and outputs a probability distribution on the optional actions, with parameters θ.

A detailed explanation of the performance evaluation procedure is also given in Section 5, The parameters of P_θ were used to initialize the Q-network and the reward network.

3 Deep reward induction

The problem of designing suitable reward functions to guide an agent to successfully learn a desired task is known as *reward shaping* (Laud 2004; Ng, Harada, and Russell 1999). The idea is to define supplemental reward signals so as to make an RL problem easier to learn. The handcrafting of a reward function can be a complicated task, requiring experience and a fair amount of specific domain knowledge. Therefore, other methods for designing a reward function without the domain expertise requirement have been investigated. In *Inverse RL* (IRL), a reward function is learned from expert demonstration (Abbeel and Ng 2004). IRL algorithms rely on the fact that an expert demonstration implicitly encodes the reward function of the task at hand, and their goal is to recover a reward function that best explains the expert's behavior.

Daniel et al. (2014) proposed to learn a reward model in a supervised manner and use iterations between reward learning and reinforcement learning to continually improve the reward model and the agent's policy. Their approach, which is based on manual feature generation by experts, has been applied in a robotic arm grasping task. The reward function is not learned from the raw states visited by the learner, but from some assumed to-be-known, low-dimensional feature representation of them. Constructing such low-dimensional representations usually requires some expert domain knowledge, making the learning of the reward function somewhat less advantageous.

Our reward induction component *learns* the reward function directly from the raw image pixels. Since we do not have access to the internal state of a simulator, defining a reward signal using the car's state parameters (distances from the roadsides, angle with respect to the road, etc.) is impossible without explicit image processing. In this work, we devise and utilize a deep reward network that is learned from a human driving instructor. The reward network, which is implemented with convolutional layers, maps a game state into the instantaneous reward, $r \in [-1, 1]$, corresponding to that state, $R_\theta : s \rightarrow r$. The driving instructor provides binary labeling for each state such that the reward network is a mapping from raw image pixels to { "good", "bad" }. The binary labeling task, which is performed by the human instructor, continues until we are convinced that the reward model is sufficiently accurate (when evaluated on a test set.)

During the imitation learning stage, the human instructor ignored the lane marks on the road (some of the tracks do not contain lane marks at all), but now we consider a more complex application whereby the agent must drive in a designated lane only (the second lane from the right on a four-lane road). Considering this new task, we aim to convey a main concept of our scheme: leveraging the weak supervision abilities of a (human) instructor who – while unable to perform the desired task (driving in a specific lane) herself – can provide informative reward signals to the computerized trainee. To this end, we train a reward network, denoted R_θ^{lane}, to give a high reward only for states where the car was in the specified lane. R_θ^{lane} has an identical architecture to that of the policy network, except for the final fully connected layer that now has one node instead of three. A hyperbolic tangent (tanh) activation function is applied on the last layer to receive output in the range $[-1, 1]$. The reward network is trained with data recorded from a human instructor in a supervised learning procedure. The MSE loss objective is used to train the reward network,

$$\mathbb{E}_{s,l\sim\mathcal{D}} \left[\left(l - R_\theta \left(s\right)\right)^2 \right],$$

where $l \in \{-1, 1\}$ is the corresponding label given by the human instructor. R_θ^{lane} was trained with approximately 30,000 new samples, denoted by D_{lane}, corresponding to roughly one driving hour. States in which the car was in the designated lane were labeled one, and all other states were labeled minus one. A 97% validation accuracy was achieved

with early stopping during the training of R_θ^{lane}. An example of the output of R_θ^{lane} in different states is given in Figure 1, showing very high reward for being in the designated lane.

(a) "Bad" states (b) "Good" states

Figure 1: R_θ^{lane} outputs high reward for driving in the second lane from the right (b), and low reward when in other lanes (a). The reward is the number located in the lower-right corner of each frame.

4 RL using DDQN

In this section we assume basic familiarity with reinforcement learning; see, e.g., Sutton and Barto (1998). In the final RL stage, we utilize the already trained reward network and apply it within a standard RL method. Performance is measured in terms of the learned reward model, and the main goal of this stage is to achieve a significantly better performance level than the one achieved in the mimicry stage by enabling the agent to teach itself. In other words, starting with a policy π_0, we would like to apply an RL algorithm to learn a policy π^*, which is optimal with respect to the expected (discounted) reward received from the reward network.

We use a variant of the Q-learning algorithm (Watkins and Dayan 1992), which aims to find the *optimal action-value function*, denoted by $Q^\star(s,a)$, and defined as the expected (discounted) reward after taking action a in state s and following the optimal policy $\pi^\star$ thereafter. The true value of taking action a in state s under policy π is

$$Q^\pi(s,a) = \mathbb{E}\big[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + ... | s_t = s, a_t = a; \pi\big],$$

where $\gamma \in [0,1]$ is a fixed discount factor and r is the guidance reward signal. Given the optimal action-value function, which is defined to be $Q^\star(s,a) = \max_\pi Q^\pi(s,a)$, the optimal policy $\pi^\star$ can be derived simply by taking the action with the highest action-value function in each state. Dealing with a very large state space of images, we approximate the optimal action-value function using a *deep Q-network* (DQN) with parameters θ:

$$Q_\theta(s,a) \approx Q^\star(s,a).$$

A deep Q-network is a neural network, which for a given state, outputs a vector of action values.

We used the DDQN algorithm with replay memory and target values calculated from parameters of the previous iteration, as in the work of Mnih et al. (2015). The loss function we used is, therefore,

$$L(\theta) = \mathop{\mathbb{E}}_{s,a,r,s' \sim \mathcal{B}} \left[\left(r + \gamma Q_{\tilde{\theta}}(s', \operatorname{argmax}_{a'} Q_\theta(s',a')) - Q_\theta(s,a) \right)^2 \right],$$

where (s,a,r,s') are samples taken from the replay memory buffer $\mathcal{B}$, and $\tilde{\theta}$ are the Q-network's parameters from the previous iteration. In our applications, the reward received after taking action a in state s is the instantaneous reward obtained from the reward network for state s (see Section 3). We set γ to be 0.9 and used the ADAM (Kingma and Ba 2014) method for stochastic gradient optimization.

We trained a Q-network, denoted Q_θ, which has the same architecture as P_θ except for the final softmax nonlinearity, which was removed. The Q-network's parameters were initialized from those of the policy network except for the final layer's parameters, which were initialized from a uniform distribution $[-1 \times 10^{-3}, 1 \times 10^3]$. We wished, on the one hand, to maintain the final convolutional layer of the policy network, which contains an informative state representation. Yet, on the other hand, we also had to allow the Q-network to output values compatible with Q-learning temporal difference targets. Therefore, an initial policy evaluation step was performed, where we let P_θ drive, and updated the parameters of only the last two fully-connected layers of Q_θ while fixing all its other parameters. This step can be viewed as learning a critic (and more precisely, only the last two layers of the critic's network) to estimate an action-value function while fixing the acting policy. After this policy evaluation step, we started the RL algorithm using Q_θ, allowing all its parameters to be updated. We refer to this type of initialization as *IL initialization*. Full details of the experiments are given in Section 5.

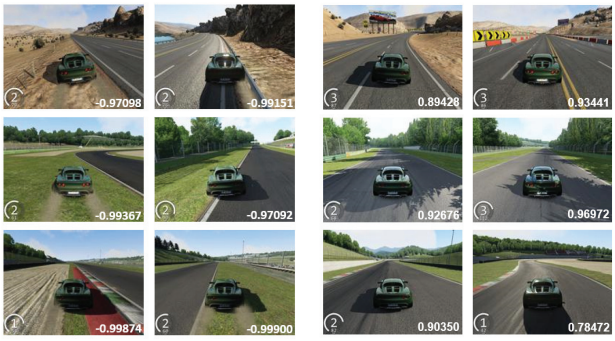
Safety Network for Safe RL

We incorporate the notion of safety into the RL process using a *safety module* that is composed of: (i) a *safety network*, denoted S_θ, whose purpose is to classify the state space into two classes: safe and unsafe; (ii) a safe driving policy, denoted P_θ^{safety}, which takes control of the agent's controller when it is in an unsafe state. S_θ maps a game state into a real number in the range $[-1,1]$, corresponding to the amount of risk in that state, where the value 1 is assigned to the safest possible state and the value -1 is assigned to the riskiest possible state. States are classified as "safe" and "unsafe" according to the safety network's output; if it is above a predefined threshold, the state is labeled "safe." Otherwise it is labeled "unsafe." When unsafe states are encountered, P_θ^{safety} takes control of the car until it is out of danger. Specifically,

$$\text{driving policy} = \begin{cases} P_\theta^{safety}, & \text{if } S_\theta < threshold \\ Q_\theta, & \text{if } S_\theta > threshold. \end{cases}$$

S_θ has the same architecture as R_θ^{lane}, and is trained the same, but with a different dataset (including labeling). Training data for S_θ were gathered in the following manner. A human instructor drove the car while intentionally alternating between edge conditions (i.e., driving on, or even outside road boundaries) and safe driving; examples are provided in a supplementary video. "Safe" states, where the car is on the road facing the correct direction, as demonstrated by the human instructor, are labeled one. All other states are labeled minus one. The training of S_θ continues until a statistical test

indicates that this network provides accurate results over test data. The safe driving policy is the learned policy from the IL stage, P_θ. We note that the decision to use the IL policy as a safe policy is specific to our application and very plausibly, it may not be the best choice in the general case. For example, it is conceivable that in a physical environment, a safe policy will consist of a predefined sequence of actions that cause the car to come to a complete stop, then disengage and yield control to a human driver or to an automaton that will return the car to a safe position or action. By using the proposed safety module, we were able to train our agent faster and with fewer accidents, as can be seen in Figures 3 and 6, and Table 1. An example of the output of S_θ in different states is given in Figure 2. We also note that it is possible to train a single reward network to predict a graded reward, which includes both our standard reward as well as extremely low values corresponding to unsafe states. Since in our setting, however, the reward labeling is given by the human instructor in real-time (while the agent is driving the car), we found it easier to provide binary labels than graded reward feedback.



(a) "Unsafe" states (b) "Safe" states

Figure 2: S_θ outputs a high score for "safe" states according to the instructor (b), and a low score for "unsafe" states (a). The output of the safety network is the number located in the lower-right corner of each frame.

5 Experiments and results

RL experiments. For the RL experiments we used R_θ^{lane} on the "Black Cat County" track (see supplementary material). The Q-network's parameters were initialized in two different ways: (i) from the policy network's parameters, as explained in Section 4; and (ii) with random initialization for a baseline. The agent operated with an ϵ-greedy policy ($\epsilon = 0.05$). The agent was trained for a total of 3.5 million frames (that is, around 4 days of game experience) with a replay memory of the $5,000$ most recent frames. Figure 3 and Table 1 show that initializing the Q-network's parameters with those learned in the IL stage increased the learning rate at the beginning of the RL, and significantly reduced the number of critical driving mistakes made by the agent (compared to randomly initializing the Q-network).

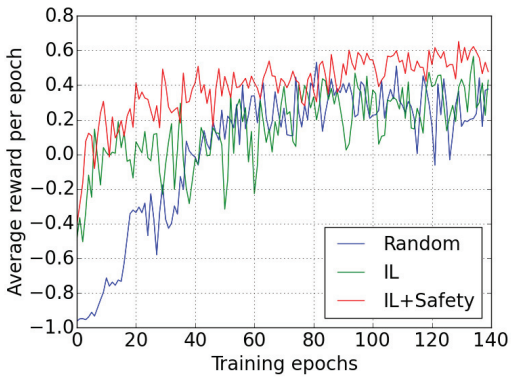
By incorporating the safety module into the RL process, we could further expedite the agent's learning rate and better circumvent critical driving mistakes. This is evident in Figure 3, where the reward and action-value curves of **IL + Safety** consistently dominates the corresponding **IL** curves. Figure 6 further emphasizes the advantage of the safety module, now comparing randomly initialized RL with and without the safety module. Figure 4 presents the number of safe policy takeovers during the RL process; as time goes by, the agent's driving skills improve and it is less likely to enter dangerous states. Even though the following comparison is not exactly "apples to apples", we note that the final RL agent outperformed the human instructor's initial demonstration; namely, it received higher rewards than the imitation policy P_θ (reminder: the RL agent was trained to drive in a designated lane, as opposed to the imitation policy).

Table 1: Averaged number of accidents per epoch throughout the agent's RL training. The training is divided into four different episodes as follows: epochs 0–3, epochs 3–12, epochs 12–39 and epochs 39–120. An epoch is defined as $15,000$ sample frames. The first row corresponds to a random initialization of the Q-network. In the second and third rows, the Q-network is initialized with the IL policy parameters. In the third row, the safety module is incorporated into the reinforcement learning. The number of accidents in the last episode is averaged over 81 epochs, but in the final epochs, both the **IL** and **IL + Safety** agents can complete tracks without accidents.

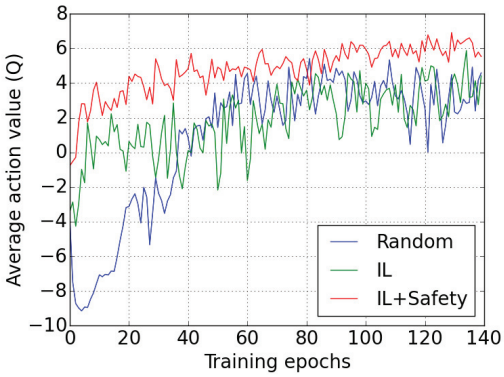
	0-3	**3-12**	**12-39**	**39-120**
RANDOM	116.6	104.7	68.5	41
IL	67	65.3	68.1	44.5
IL + SAFETY	47	46.1	36.1	23.3

A high quality reward signal is critical to the agent's success during the RL stage. In the proposed scheme, the reward signal is learned from human instructions. Therefore, the learned reward network must be able to generalize the human instructions for unseen states. To investigate the effect of the reward signal's quality on the agent's overall performance during the RL stage, we conducted the following experiment. An additional reward network, denoted $R_\theta^{0.2\,lane}$, was trained using only 20% of the dataset D_{lane}, and achieved 66% validation accuracy. We then trained an RL agent with a reward signal given by $R_\theta^{0.2\,lane}$ and compared its performance to the RL agent trained using R_θ^{lane}. The results, presented in Figure 5, emphasize the necessity of a reward function that is able to correctly generalize the human instructor's knowledge to unseen states.

Technical implications of not having access to the simulator's internal state. While performing the RL algorithm, the following operations are executed online: image capturing, image pre-processing, reward prediction by the reward network, action prediction by the Q-network, Q-network parameter updates, and others. On average, the serial execution of these operations takes 70 ms. Therefore, the agent must always act under 70 ms latency (from the time the image is captured to the time the corresponding steering key



(a) Average reward



(b) Average action-value

Figure 3: Agent's training curves. An epoch is defined to be $15,000$ sample frames. The Q-network's parameters were initialized as follows: Blue: Random initialization. Green: IL initialization. Red: IL initialization + safety module. (a) Average reward per epoch. (b) Average action-value per epoch. The average reward achieved by the imitation policy, P_θ, was -0.45.

is pressed). During this gap, the previous key continues to be pressed until a new one is received. This added difficulty can be avoided when using a simulator. Most importantly, our inability to apply the RL stage at a super-fast simulation speed limits the number of training epochs we can afford to conduct.

Image preprocessing. The images were resized from 1024×768 to 192×144 and were not converted to gray scale. The pixel values were scaled to be in the range $[0, 1]$. Pixels corresponding to the speed indicator were set to zero in order to guarantee that the agent does not use the speed information during the learning process. In all our experiments, each input instance consisted of two sequential images with a 0.5-second gap between them.

Work environment. We used the Theano-based Lasagne library for implementing the neural networks. Two GeForce GTX TITAN X GPUs were utilized during the RL experiments; one was used to run the racing game and the other to

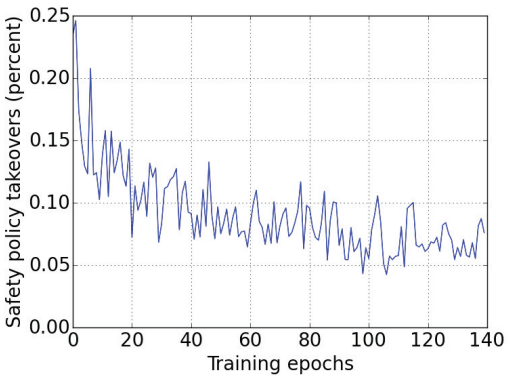


Figure 4: Fraction of safety policy takeovers per decision in each epoch during the RL. An epoch is defined to be $15,000$ sample frames. Each point in the graphs is the relative number of sample frames per epoch in which the safety policy took control (i.e., the fraction of sample frames for which the output of the safety network was 'unsafe'.) The Q-network was initialized with the IL policy network parameters (IL initialization).

train the networks and predict the rewards and actions.

Data generation and network training technicalities. The human demonstrator played the game using a "racing" strategy: driving as fast as possible while ignoring the lane marks and trying to avoid accidents. Training data for the policy and reward networks was collected from the tracks: 'Black Cat County', 'Imola' and 'Nürburgring gp'. The "Black Cat County" track is the only one with lane marks, and it was used for the RL experiments. Sample images of the tracks are given in the supplementary material. Using a Python environment, screen images were recorded every 0.1 seconds while the human demonstrator played the game. Keyboard keys pressed by the demonstrator were also recorded. The same sampling rate was used when gathering data for the reward induction stage. 80% of the recorded samples were used as training data and the remaining 20% as validation data. We used the ADAM stochastic optimization method with dropout for regularization (Srivastava et al. 2014). The network parameters that achieved the best validation accuracy were chosen.

Performance evaluation. Without a natural performance evaluation measure for driving skills and without access to the internal state of the game, our performance evaluation procedure is based mainly on the accumulated average reward achieved by the agent. Better driving means higher (discounted) accumulated average reward and vice versa. Moreover, we also use the agent's number of car accidents as a performance evaluation measure. Car accidents are detected in the following manner: if the difference between two sequential speed frames exceeds a predefined threshold, the event is considered as an accident.

Other technicalities. Game restarts were performed when the agent had zero speed (e.g., stuck against some obstacle) or when it drove in the wrong direction (both of these

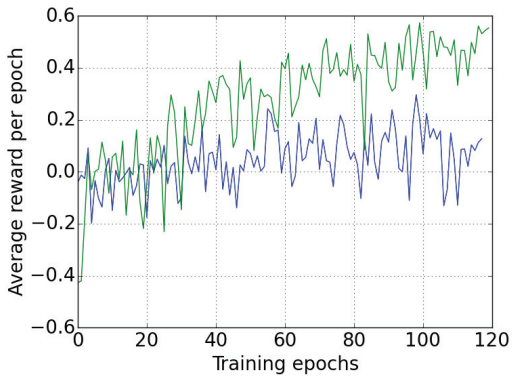


Figure 5: Agent's RL training curves. An epoch is defined to be $30,000$ sample frames. Each point in the graphs is the average reward achieved per epoch from the reward network R_θ^{lane}. Green: reward signal is received from R_θ^{lane}. Blue: reward signal is received from $R_\theta^{0.2\,lane}$.

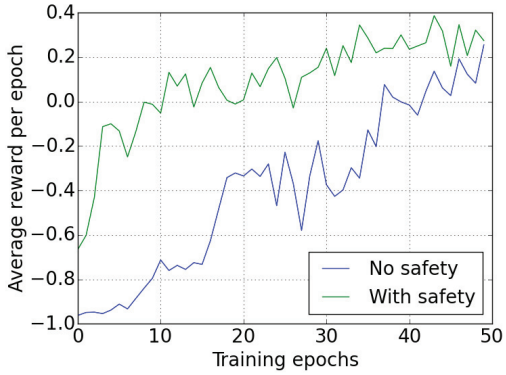


Figure 6: Agent's training curves. An epoch is defined to be $15,000$ sample frames. Each point in the graph is the average reward per epoch. The Q-network was initialized with random parameters in both cases. Blue: no safety module. Green: with safety module.

situations were detected through simple image processing). All the experiments were conducted without any other vehicles on the road. We used the 'Lotus Elise SC' car model in all of our experiments. Focusing only on the steering control problem, we eliminated accelerator/brake control variability by using a "cruise control" behavior whereby the car's speed was set to 50 kmh. This was achieved by extracting the car's speed from the game image and applying a simple handcrafted controller.

6 Concluding remarks

We presented a generic learning framework consisting of three supervised elements that facilitate RL training of a computerized agent, which should perform a complex task based on raw perception. In our setting, the agent uses raw image pixels without any image processing or other handcrafted features. We expect the proposed framework to be useful in various application domains, and have demonstrated its strength on an autonomous highway steering problem. Our solution relies on recent deep representational methods (CNNs) to successfully implement all elements of the proposed framework. We successfully demonstrated the advantage of all four elements, and we note that the proposed implementation is scalable to any computerized (black-box) driving game where the steering is controlled via the keyboard.

A typical real-world RL application often relies on a accurate simulator of the environment, which enables super-fast applications of RL algorithms by accelerating the real-world clock. In our setting, it is impossible to accelerate the real-world clock (as in many other real-world tasks for which constructing an accurate simulator is extremely difficult). This limitation is hard to overcome and requires appropriate and extremely effective RL, as we aim to achieve here. In contrast, in the work of Mnih et al. (2015), where RL is used to predict desired control actions in the Atari 2600 domain, it is possible to train the RL agent (using the open source ALE simulator) 100 times faster than in our setting.

The main strength of our scheme lies in leveraging the weak supervision abilities of a (human) instructor who – while unable to perform the required task well herself – can provide coherent and learnable instantaneous reward signals to the computerized trainee. In other words, the instructor should be able to perform the task herself only at a very basic level (and demonstrate it to the agent), and should be able to provide helpful feedback to the agent if actions taken by the agent are advancing the task or not. This leveraging effect clearly occurred in our self-steering example, where single-lane driving demonstrations (by the instructor) were not included in the imitation stage (and moreover, most of the training tracks do not even include lane marks). Nonetheless, the agent quite easily learned to drive in a single designated lane after a one-hour instruction session followed by self-reinforcement learning (without the instructor).

Many directions have been left open for future research. First, our self-steering example can be improved in various ways. It would be very interesting to extend our solution to include acceleration and breaking control. Then, it would be challenging to train a reward function to promote fast racing style driving. While the proposed safety module proved itself as an extremely effective technique to expedite the RL in our setting, our supervised approach to construct this module was dependent on the particular task we considered. In general, it would be very interesting to develop effective approaches to constructing such modules for arbitrary tasks, perhaps using semi-supervised, single-class classification or even unsupervised techniques (El-Yaniv and Nisenson 2007). Other useful techniques that can be used in learning to detect unsafe states are selective prediction, and concept drift detection (Harel et al. 2014; Wiener and El-Yaniv 2015; Geifman and El-Yaniv 2017).

Whether the proposed approach can be used to train real-world robotic agents in the absence of a realistic simulator is a major open question. To successfully apply our scheme, both effective acquisition of instantaneous rewards from an

instructor and accurate modeling of the reward function are required. While in our driving example, the reward models were easily constructed, creating these models for highly complex tasks is expected to be challenging in terms of both model capacity and the development of effective methodologies for interaction with the instructor. Furthermore, to handle more complex tasks (e.g., cooking, dishwasher loading), the policies must operate a number of controllers (or more complicated controllers with many degrees of freedom). An interesting question in this regard is how to construct appropriate network architectures and training methodologies to jointly handle many related controllers. We anticipate that harnessing the supervision abilities of a (human) instructor, for the purpose of learning an effective reward model will become a critical building block in creating complex robotic agents.

Acknowledgments

This research was supported in part by the Israel Science Foundation (grant No. 1890/14).

References

Abbeel, P., and Ng, A. Y. 2004. Apprenticeship Learning via Inverse Reinforcement Learning. In *Proceedings of the Twenty-First International Conference on Machine Learning*.

Argall, B. D.; Chernova, S.; Veloso, M.; and Browning, B. 2009. A survey of robot learning from demonstration. *Robotics and Autonomous Systems* 57(5):469–483.

Chen, C.; Seff, A.; Kornhauser, A.; and Xiao, J. 2015. Deepdriving: Learning Affordance for Direct Perception in Autonomous Driving. In *Proceedings of the IEEE International Conference on Computer Vision*, 2722–2730.

Daniel, C.; Viering, M.; Metz, J.; Kroemer, O.; and Peters, J. 2014. Active Reward Learning. In *Proceedings of Robotics Science & Systems*.

El-Yaniv, R., and Nisenson, M. 2007. Optimal single-class classification strategies. In *Advances in Neural Information Processing Systems*, 377–384.

Garcıa, J., and Fernández, F. 2015. A comprehensive survey on safe reinforcement learning. *Journal of Machine Learning Research* 16(1):1437–1480.

Geifman, Y., and El-Yaniv, R. 2017. Selective classification for deep neural networks. In *Advances in Neural Information Processing Systems*.

Harel, M.; Mannor, S.; El-Yaniv, R.; and Crammer, K. 2014. Concept drift detection through resampling. In *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, 1009–1017.

Hasselt, H. V.; Guez, A.; and Silver, D. 2016. Deep Reinforcement Learning with Double Q-Learning. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*.

Hasselt, H. V. 2010. Double Q-learning. In *Advances in Neural Information Processing Systems*, 2613–2621.

Hayes, G. M., and Demiris, J. 1994. A Robot Controller Using Learning by Imitation. Technical report, University of Edinburgh, Department of Artificial Intelligence.

Kingma, D., and Ba, J. 2014. Adam: A Method for Stochastic Optimization. *arXiv preprint arXiv:1412.6980*.

Kober, J.; Bagnell, J. A.; and Peters, J. 2013. Reinforcement Learning in Robotics: A survey. *The International Journal of Robotics Research*.

Krizhevsky, A.; Sutskever, I.; and Hinton, G. E. 2012. Imagenet Classification with Deep Convolutional Neural Networks. In *Advances in Neural Information Processing Systems*, 1097–1105.

Laud, A. D. 2004. *Theory and Application of Reward Shaping in Reinforcement Learning*. Ph.D. Dissertation, University of Illinois at Urbana-Champaign.

LeCun, Y. 2017. Predictive Learning: the Next Frontier in A.I. https://www.youtube.com/watch?v=qjI8lJn8Mss&app=desktop. Quote appearing in time: 54:38.

Loiacono, D.; Prete, A.; Lanzi, P. L.; and Cardamone, L. 2010. Learning to Overtake in TORCS Using Simple Reinforcement Learning. In *IEEE Congress on Evolutionary Computation*, 1–8.

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A. A.; Veness, J.; Bellemare, M. G.; Graves, A.; Riedmiller, A.; Fidjeland, A. K.; Ostrovski, G.; et al. 2015. Human-level control through deep reinforcement learning. *Nature* 529–533.

Munoz, J.; Gutierrez, G.; and Sanchis, A. 2009. Controller for TORCS created by imitation. In *IEEE Symposium on Computational Intelligence and Games*, 271–278.

Ng, A. Y.; Harada, D.; and Russell, S. J. 1999. Policy invariance under reward transformations: Theory and application to reward shaping. In *International Conference on Machine Learning*, volume 99, 278–287.

Srivastava, N.; Hinton, G.; Krizhevsky, A.; Sutskever, I.; and Salakhutdinov, R. 2014. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research* 15(1):1929–1958.

Sutton, R. S., and Barto, A. G. 1998. *Reinforcement Learning: An Introduction*. MIT press Cambridge.

Taylor, M. E.; Suay, H. B.; and Chernova, S. 2011. Using Human Demonstrations to Improve Reinforcement Learning. In *AAAI Spring Symposium: Help Me Help You: Bridging the Gaps in Human-Agent Collaboration*.

Watkins, C. J. C. H., and Dayan, P. 1992. Q-Learning. *Machine Learning* 279–292.

Wiener, Y., and El-Yaniv, R. 2015. Agnostic pointwise-competitive selective classification. *Journal of Artificial Intelligence Research* 52:171–201.

Wymann, B.; Espié, E.; Guionneau, C.; Dimitrakakis, C.; Coulom, R.; and Sumner, A. 2000. TORCS, the open racing car simulator. *Software available at http://torcs. sourceforge. net*.

Zhang, J., and Cho, K. 2016. Query-Efficient Imitation Learning for End-to-End Autonomous Driving. *arXiv preprint:1605.06450*.