Deep Reinforcement Learning for Simulated Autonomous Driving

Adithya Ganesh
Stanford University
`acganesh@stanford.edu`

Joe Charalel
Stanford University
`jcharale@stanford.edu`

Matthew Das Sarma
Stanford University
`madassar@stanford.edu`

Nancy Xu
Stanford University
`xnancy@stanford.edu`

Abstract

This research explores deep Q-learning for autonomous driving in The Open Racing Car Simulator (TORCS). Using the TensorFlow and Keras software frameworks, we train fully-connected deep neural networks that are able to autonomously drive across a diverse range of track geometries. An initial proof-of-concept of classical Q-learning was implemented in Flappy Bird. A reward function promoting longitudinal velocity while penalizing transverse velocity and divergence from the track center is used to train the agent. To validate learning, the research analyzes the reward function parameters of the models over two validation tracks and qualitatively assesses driving stability. A video of the learned agent driving in TORCS can be found online: `https://www.dropbox.com/sh/b4623soznsjmp12/AAA1UD8_oaa94FgFC6eyxReya?dl=0`.

1 Background and Introduction

Our research objective is to apply reinforcement learning to train an agent that can autonomously race in TORCS (The Open Racing Car Simulator) [1, 2]. TORCS is a modern simulation platform used for research in control systems and autonomous driving. Training an autonomous driving system in simulation offers a number of advantages, as applying supervised learning on real-world training data can be expensive and requires substantial amounts of labor for driving and labeling. Furthermore, a simulation is a safe, efficient, and cost-effective way to identify and test failure cases (e.g. collision events) of safety-critical control systems, without having to sacrifice physical hardware. Accurate simulation platforms provide robust environments for training reinforcement learning models which can then be applied to real-world settings through transfer learning.

Deep reinforcement learning has been applied with great success to a wide variety of game play scenarios. These scenarios are notable for their high-dimensional state spaces that border on real-world complexity. In particular, the deep Q-network (DQN) algorithm introduced by Google's DeepMind team in 2015 has been shown to successfully learn policies for agents relying on complex input spaces [3]. Prior to the introduction of DQN, applicability of reinforcement learning agents was limited to hand-crafted feature selection or low-dimensional state spaces. To successfully apply reinforcement learning to situations with real-world complexity, agents must derive efficient representations of high-dimensional sensory inputs and use these features to to generalize past observations to future samples. The DQN algorithm was shown to surpass the performance of all previous algorithms and achieve a performance level comparable to that of a professional games tester across 49 Atari games.

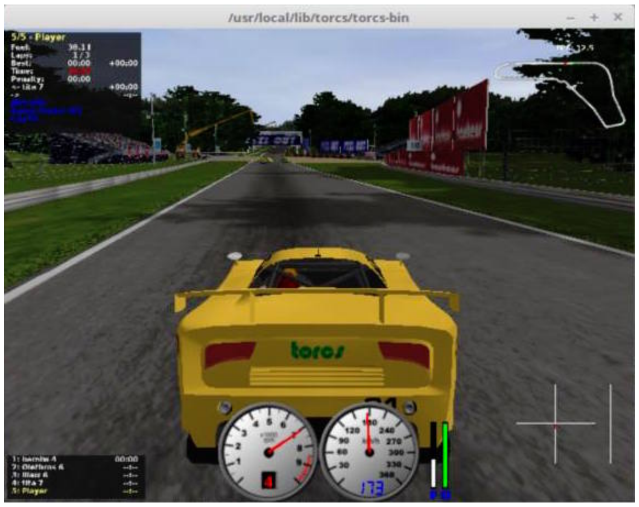


Figure 1: TORCS simulation environment

1.1 Relevant Work

There are two notable, distinct past approaches to training autonomous driving agents in TORCS. In 2015, the DeepDriving model applied deep supervised learning and convolutional neural networks (CNN) to learn a driving policy [4]. Other research, such as that done by DeepMind, focuses on reinforcement learning with CNNs [5]. The research by DeepMind demonstrates the wide applicability of actor-critic architectures, which use a pair of neural networks to address deep reinforcement learning, to continuous control problems.

Further advances have made it possible to apply Deep Q-Learning to learn robust policies for continuous action spaces. The deep deterministic policy gradient (DDPG) method [5] presents an actor-critic, model-free algorithm based on the deterministic policy gradient. The algorithm is designed for physical control problems over high-dimensional, continuous state and action spaces. As such, it has potential applications in numerous physical control tasks. This paper continues to explore deep Q-learning for continuous control within the context of The Open Racing Car Simulator (TORCS).

2 Classical Q-learning experiments

Q-learning models the dynamics of a control process as a Markov Decision Process through iterative updates of an approximation of the expected total reward of state-action pairs with respect to the optimal control policy [6]. In Q-learning, the estimated Q-value of a state-action pair is given by the function $Q : S \times A \to \mathbb{R}$. We apply the standard update rule for a state-actor pair (s, a) as follows:

$$Q(s,a) := Q(s,a) + \alpha \left(R + \gamma \max_{a' \in A} Q(s', a') - Q(s,a) \right)$$

As a proof of concept, our initial experiments applied classical Q-learning to Flappy Bird by discretizing the continuous state space of object location and velocity. We characterized the game state by the vertical and horizontal separation of the bird and the next obstacle and the vertical velocity of the bird. We compress this state into $S \subset \mathbb{Z}^3$:

$$S = \{(\Delta y, \Delta x, v_y) \mid |\Delta y| \leq 50,\ 0 \leq \Delta x \leq 40,\ |v_y| \leq 9\}$$

Our action is a binary decision of whether of not to jump. In each frame, we reward staying alive with $R = 0.01$ and penalize crashing with $R = -10$. The discretization and compression cause the MDP to be stochastic, so we set $\alpha = .1$ as a means for permitting gradual learning of the appropriate policy. Additionally, we set $\gamma = .99$ to assign significant value to future reward. In order to fairly explore the state-action space $S \times A$, we bias the agent towards selecting the better move during training, while selecting a random move with probability $\epsilon = 0.15$ for exploration on each iteration. We also considered the Boltzmann softmax function as a scheme for biasing the agent towards actions with a greater estimated value; however, it is difficult to stipulate a cooling rule that causes Q to converge at an appropriate rate.

After roughly 35,000,000 iterations, the policy converged to a nearly perfectly playing computer agent. However, this demonstrates that classical Q-learning is infeasible for more complex control problems, since each additional dimensions contributes exponentially to the size of the state space. This "curse of dimensionality" makes it unlikely that Q-learning applied to discretized state spaces will be tractable for TORCS without significant preprocessing of the high-dimensional state space.

3 Methods

3.1 Deep Q-Learning Overview

Consider a deterministic policy $\mu_\theta : \mathcal{S} \to \mathcal{A}$ parametrized by $\theta \in \mathbb{R}^n$. Let r_t^γ denote the total discounted reward from time-step t onwards:

$$r_t^\gamma = \sum_{k=t}^{\infty} \gamma^{k-t} r(s_k, a_k),$$

where $0 < \gamma < 1$. We can define a performance objective $J(\mu_\theta)$ as the total discounted reward for a given policy, while the Q-function calculates the total discounted reward given a starting state-action pair:

$$J(\mu) = \mathbb{E}[r_1^\gamma | \mu]; \quad Q^\mu(s,a) = \mathbb{E}[r_1^\gamma | S_1 = s, A_1 = a; \mu].$$

If ρ^μ is the discounted state distribution, the deterministic policy gradient theorem [7] states that:

$$\begin{aligned}\nabla_\theta J(\mu_\theta) &= \int_{\mathcal{S}} \rho^\mu(s) \nabla_\theta \mu_\theta(s) \nabla_a Q^\mu(s,a)|_{a=\mu_\theta(s)} \mathrm{d}s \\ &= \mathbb{E}_{s \sim \rho^\mu}[\nabla_\theta \mu_\theta(s) \nabla_a Q^\mu(s,a)|_{a=\mu_\theta(s)}].\end{aligned}$$

This formulation allows us to obtain the optimal policy applying a gradient ascent update rule on the objective $J(\mu_\theta)$.

3.2 Reward Function

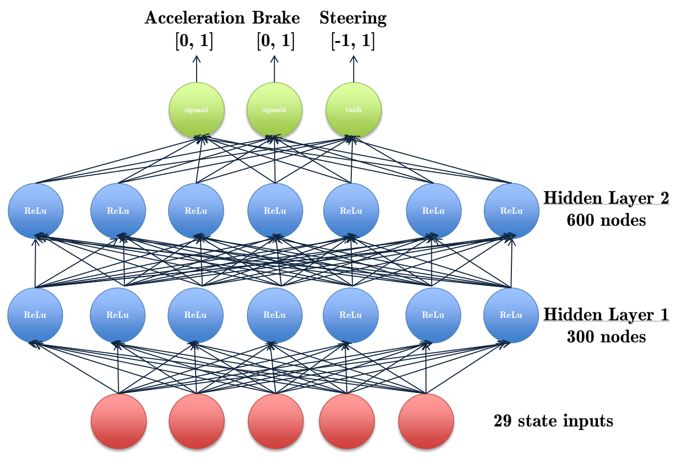
In the TORCS environment (Figure 1), each state observation captures componentwise velocities (e.g. $V_{x,t}$), angle relative to the track θ_t, distances to track center and edges, and wheel rotation speed in double precision. Our action is a triple specifying the steering angle, acceleration, and braking in double precision. We reward longitudinal velocity while penalizing transverse velocity and divergence from the median μ_t of the track:

$$R = \sum_t |V_{x,t} \cos \theta_t| - |V_{x,t} \sin \theta_t| - |V_{x,t}||x_t - \mu_t|$$

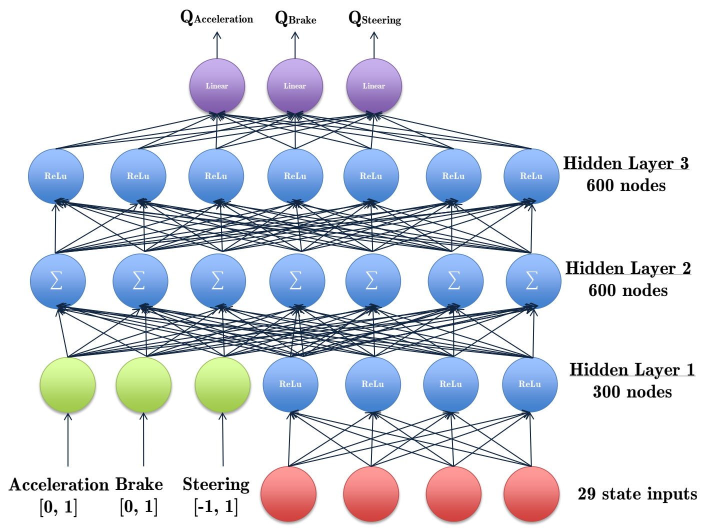
The observation $V_{x,t}$ captures the velocity of the car in the direction of its heading.

3.3 Neural Network Architectures

Architecture A: Fully-Connected Networks. In our actor-critic approach to deep Q learning, the critic model estimates Q-function values while the actor model selects the most optimal actions for each state based on those estimates. This is expressed as the final layer of the actor network which determines acceleration, steering, and brake with fully connected sigmoid, tanh, and sigmoid activated neurons. These bound the respective actions within their domains. In the actor network, both hidden layers are comprised of ReLu activated neurons (Figure 2a). In the critic model, the actions are not made visible until the second hidden layer. The first and third hidden layers are ReLu activated, while the second merging layer computes a point-wise sum of a linear activation computed over the first hidden layer and a linear activation computed over the action inputs (Figure 2b).



(a) Actor network with two ReLu activated hidden layers



(b) Critic network with ReLu and linear activated hidden layers

Figure 2: Fully connected actor-critic neural network architectures for deep Q-learning

Architecture B: Recurrent LSTM Networks. A powerful variation on feedforward neural networks is the recurrent neural network (RNN). In this architecture, connections between units form a directed cycle. This allows the network to model dynamic temporal and spatial dependencies from time series driving data. Long short term memory (LSTM) networks are a robust variant of RNNs that capture long term dependencies between states. We implemented an LSTM network using TensorFlow and Keras, but it was found that further hyperparameter tuning is needed to demonstrate robust convergence.

3.4 The Replay Buffer

To diminish the effect of highly correlated training data, we train the neural networks with a large replay buffer D that accumulates 100,000 samples. During training, we update our parameters using samples of experience $(s, a, r, s') \sim U(D)$, drawn uniformly at random.

$$L = \mathbb{E}_{(s,a,r,s') \sim U(D)} \left[\left(r + \gamma \max_{a'} Q(s', a') - Q(s, a) \right)^2 \right]$$

Using nonlinear functions to approximate the Q-function often yields instability or divergence [8]. Here we exploit *experience replay* [3] to increase efficiency and mitigate instability by smoothing over changes in the data distribution. When training RNN models we sample blocks of training data from the same episode with length equal to the buffer network depth. To encourage sensible exploration, we use an Ornstein-Uhlenbeck process [5, 9] to simulate Brownian-motion about the car with respect to its momentum. This process avoids pathologies of other exploration algorithms that frequently cause the car to brake and lose momentum.

4 Results

Using the TensorFlow and Keras software frameworks [10, 11], we trained multi-layer fully connected networks with 300 and 600 respective units in each hidden layer (see Figure 2). The implementation and hyperparameter choices were adapted from [12]. A replay buffer size of 100000 state-action pairs was chosen, along with a discount factor of $\gamma = 0.99$. The Adam optimization algorithm [13] was applied, with a batch size of 32 and learning rates of 0.0001 and 0.001 for the actor and critic respectively. Target networks were trained with gradually updated parameters, as described in [3], which stabilizes convergence. Training was performed using an NVIDIA GeForce GTX 1080 GPU.

With this architecture, the resulting agent was capable of navigating a variety of test tracks of highly variable shape in TORCS. We trained the networks on the challenging Aalborg road track (a. in Figure 4). For validation, we tested the networks on the simpler A-Speedway track (b. in Figure 4), as well as the highly irregular Alpine-1 track (c. in Figure 4). See `https://www.dropbox.com/sh/b4623soznsjmp12/AAA1UD8_oaa94FgFC6eyxReya?dl=0` for a video of the policy in action.

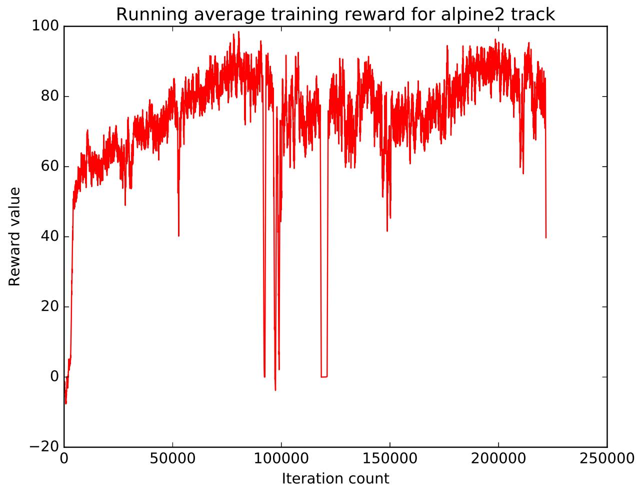


Figure 3: Sliding window average of reward over 500 samples during training of the fully connected network.

We are currently in the process of training the LSTM networks, which contain significantly more hyperparameters than the full-connected feed forward network. As a result, the LSTM requires additional tuning on to ensure convergence of the network. We discuss this further in the Future Work section.

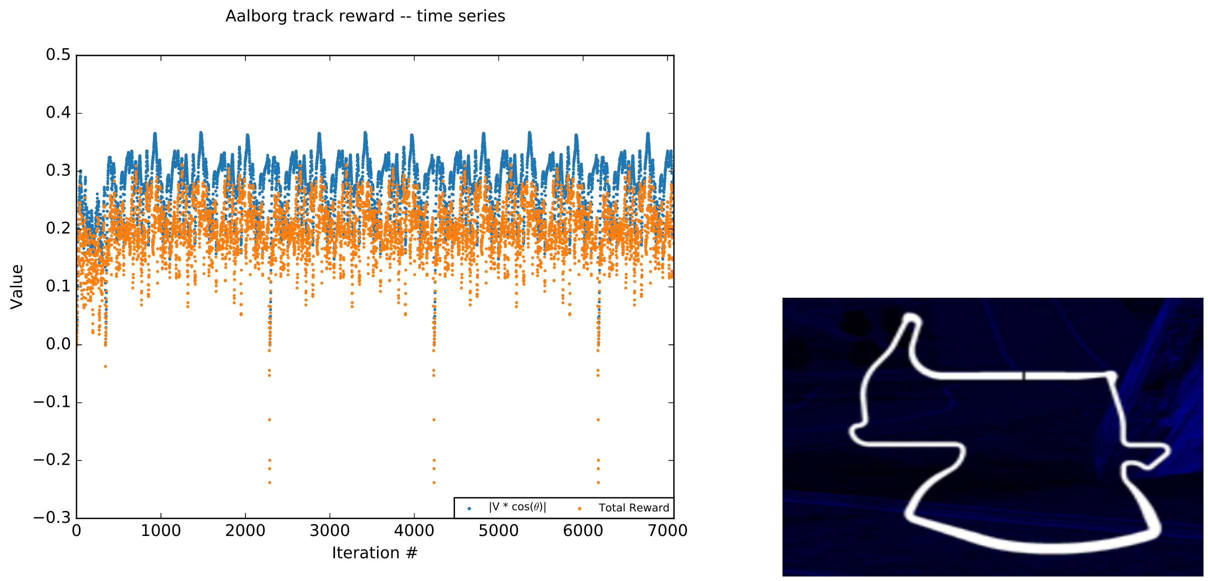
As displayed in Figure 3, over the first 80,000 iterations the observed reward values generally increased and approached a transient acceptable policy. Qualitatively, the policy after 80,000 iterations appears to drive well, persistently staying within the track and displaying stability by entering into a periodic trajectory that repeats each lap. The figure also captures the challenge of optimizing over the policy search space, which contains many local minima that may pollute the search process without careful selection of learning rates and termination conditions. While the optimizer has not converged to a constant policy, we found that choosing a policy with weights selected when the reward is at the high points of Figure 3 (roughly when $R > 80$ by our metric), produced an agent that performs relatively well in validation with respect to our reward function. Further tuning of the optimization algorithm could result in more stable convergence.

5 Conclusion and Future Work

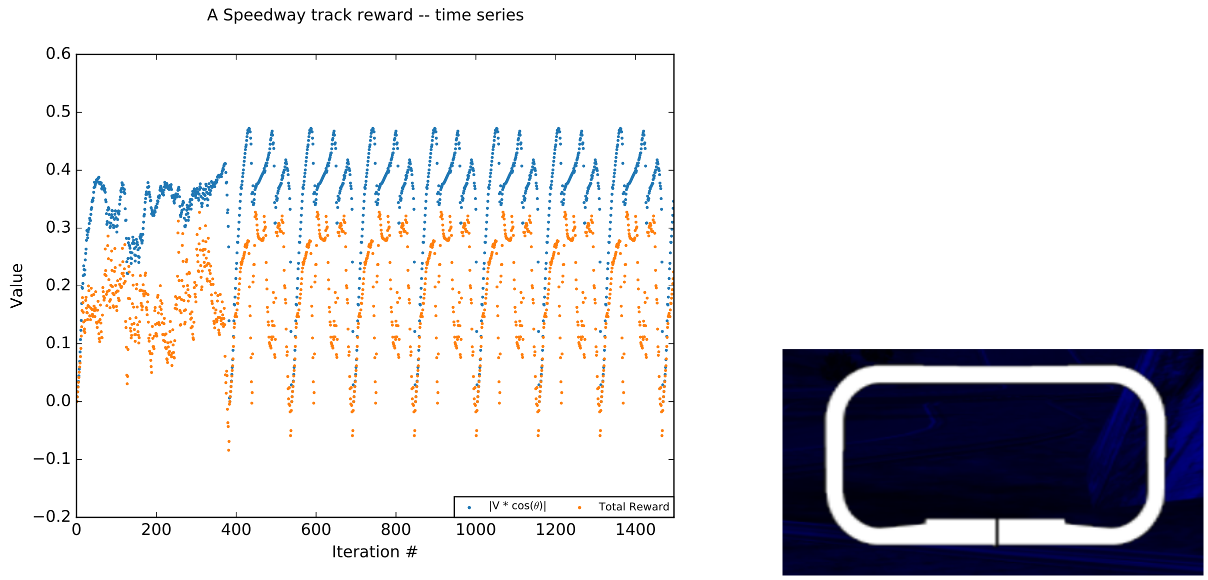
This research demonstrates a deep Q-network that can autonomously drive in TORCS, with robustness over diverse environments including the Aalborg, Alpine-1, and A-Speedway TORCS tracks.

In the future, we would like to test a variety of reward functions, exploration policies, and adaptive gradient descent strategies to optimize the likelihood of near-optimal and rapid convergence. We expect LSTM networks to be capable of learning superior policies by modeling long-term state dependencies. In particular, we expect memory on a particular track to improve performance on subsequent turns and laps. We are currently in the process of tuning the RNN hyperparameters to achieve such an optimal convergence.

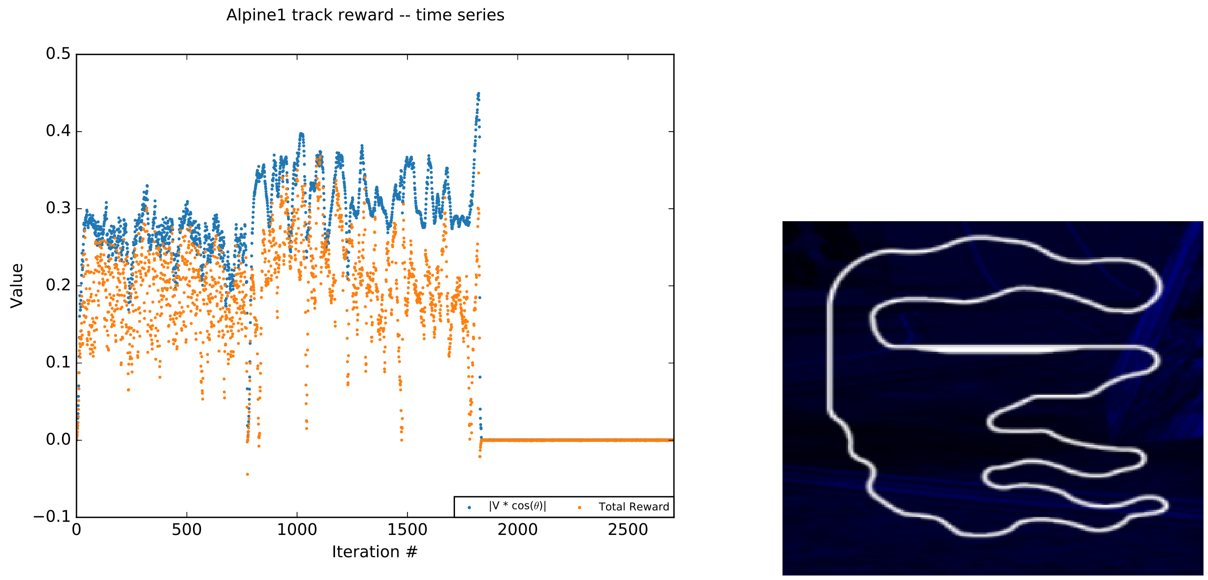
Applying reinforcement learning to train an autonomous driving system is an efficient way to simulate model architectures and failure modes without expensive labeling effort and physical hardware. In future research, we would like to explore the performance of transfer learning to physical hardware and real-world control systems. We envision configuring a sensor suite that mimics available simulated signals. One such setup would be to use LiDAR sensors along with an inertial measurement unit for obstacle distance and velocity measurements. In addition, a coordinate transformation can be applied so that real-world sensor measurements are comparable to simulated ones from TORCS. We hope this research paves the way to the realization of safe, robust autonomous driving systems.



a. Training track: Challenging Aalborg track



b. Validation track 1: Simple A-Speedway track



c. Validation track 2: Challenging Alpine-1 track

Figure 4: Total reward obtained during test racing simulations broken down into component terms. The fully connected network architecture was used for these evaluations.

References

[1] Daniele Loiacono, Pier Luca Lanzi, Julian Togelius, Enrique Onieva, David A Pelta, Martin V Butz, Thies D Lonneker, Luigi Cardamone, Diego Perez, Yago Sáez, et al. The 2009 simulated car racing championship. *IEEE Transactions on Computational Intelligence and AI in Games*, 2(2):131–147, 2010.

[2] Daniele Loiacono, Luigi Cardamone, and Pier Luca Lanzi. Simulated car racing championship: Competition software manual. *CoRR*, abs/1304.1672, 2013.

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

[4] Chenyi Chen, Ari Seff, Alain Kornhauser, and Jianxiong Xiao. Deepdriving: Learning affordance for direct perception in autonomous driving. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2722–2730, 2015.

[5] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[6] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 1998.

[7] Guy Lever. Deterministic policy gradient algorithms. 2014.

[8] John N Tsitsiklis and Benjamin Van Roy. An analysis of temporal-difference learning with function approximation. *IEEE transactions on automatic control*, 42(5):674–690, 1997.

[9] Joseph L Doob. The brownian movement and stochastic equations. *Annals of Mathematics*, pages 351–369, 1942.

[10] Martın Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*, 2016.

[11] Francois Chollet. Keras: Deep learning library for Theano and TensorFlow. `https://keras.io/`.

[12] Ben Lau. Using Keras and Deep Deterministic Policy Gradient to play TORCS. `https://yanpanlau.github.io/2016/10/11/Torcs-Keras.html`, 2016.

[13] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[14] Leslie Pack Kaelbling, Michael L Littman, and Andrew W Moore. Reinforcement learning: A survey. *Journal of artificial intelligence research*, 4:237–285, 1996.

[15] Jens Kober and Jan Peters. Reinforcement learning in robotics: A survey. In *Reinforcement Learning*, pages 579–610. Springer, 2012.

[16] Richard Bellman et al. *Introduction to the mathematical theory of control processes*, volume 2. IMA, 1971.

[17] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8(3-4):279–292, 1992.

[18] Daniel Karavolos. Q-learning with heuristic exploration in simulated car racing. 2013.