Deep Reinforcement Learning in TORCS

Evghenii Beriozchin
evghenii.beriozchin@tum.de

David Bailey
david.bailey@tum.de

Bruno Miguel
bruno.miguel@tum.de

Yuqi Shen
yuqi.shen@tum.de

1. Introduction

We are creating an agent to simulate driving in The Open Racing Car Simulator (TORCS) [3]. The agent is guided by a deep reinforcement learning model which tries to maximize the reward of correctly driving by choosing the best possible actions. The inputs to the model are sets of images extracted from the simulator. The model should output values to control the car.

1.1. Related Works

- Playing Atari with Deep Reinforcement Learning [5]
- Continuous control with deep reinforcement learning [2]

2. Dataset

- We will use the TORCS simulator as a "dataset" containing race courses and a car physics engine. We will also use pre-trained models such as SegNet[1] that have already been trained to recognize street scenes.
- The data is a conjunction of possible actions which the car can take, and a tensor of the latest k frames, where k >1.
- The simulator provides the necessary information for awareness of the car's current state and a scalar reward needed for reinforcement learning.
- Inputs are raw images extracted from the simulator and the set of possible actions for the agent, while the output is used to adjust the current state of the agent.

3. Methodology

We will use a number of convolution layers to extract the core features, and then use fully connected layers at the back of our pipeline to predict output values for each possible action the vehicle could take, and based on the result change its current state.

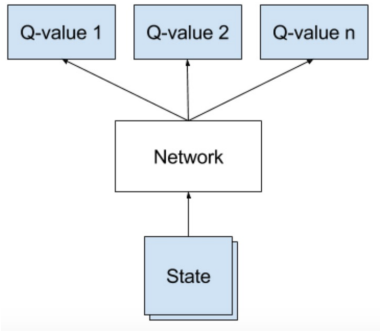

Figure 1. [4]

The network will be trained with data derived from a replay buffer, a cache that stores a queue of tuple <a, s, r, s'>where *a* belongs set of actions, *s* represents the state where action *a* was executed, *r* represents the consequence reward of executing *a* in the state *s*, which the resulting state *s'*. State is a a tensor of sequentially number of picture extracted from the driving simulator.

Since the reward from a given action may not happen immediately, we will have to predict what reward we may receive after taking an action at the current time-step. The function Q(a, s) represents the discounted future reward of taking action *a* on a state *s*. Q(a, s) is defined in regard to the next state *s'*, we use a hyper-parameter γ to specify how much we should take into consideration *s'*; if $\gamma \to 0$ then we only consider immediate rewards, while if $\gamma \to 1$, the next state is fully considered. Q(s, a) is known as the Bellman Equation and is defined as Q(s,a)=r+$\gamma max_{a'}$Q(s',a').

Given an arbitrary randomly chosen transition <s, a, r, s'>, the update rule must follow:
1. Do a feed-forward pass for the current state *s* to get predicted Q-value for all actions.
2. Do a feed-forward pass for the next state *s'* and calculate maximum over all network outputs $max_{a'}$Q(s',a').
3. Set Q-value target for action *a* to 'r+$\gamma max_{a'}$Q(s',a')'. For all other actions, set the Q-value target to the same as originally returned from step 1, making the error 0 for those

outputs.
4. Do backpropagation and update the weights using stochastic gradient descent, which has shown good results from previous experiences.

On each forward-pass we will draw samples from the experience buffer randomly in order to maximize our learning by breaking correlation between actions and states. We also believe that batch normalization may be useful to control the input's distribution of the fully connected layers and thus ameliorate the gradient flow in the backward pass. We will apply the non-linearity rectified linear unit after each layer's pass, save from layers that do not present gradients, such as max-pooling layers. The output of the network represents a scalar value for each action we should take, and according to its values, the vehicle should act properly. We believe that the training process should take one to two full days, and training will happen using rented GPU's from Google Cloud Service. We may try to use a pre-trained network as feature extractors on the gradient-freezed initial convolution layers, and then train the fully connected layers using our data.

4. Outcome

Our aspiration is to accumulate knowledge about a situation that will soon be taking over our daily routines. Autonomous vehicles are the closest robots that share the same environment to humans. Vehicles not only provide a great extent of applications, but also represent a determinant factor in the world economy. While there may exist a huge discrepancy between simulation and the real environment, with current advances in computation and hardware, we expect future generations of simulators to be used as training actors on real environments.

References

[1] V. Badrinarayanan, A. Handa, and R. Cipolla. Segnet: A deep convolutional encoder-decoder architecture for robust semantic pixel-wise labelling. *CoRR*, abs/1505.07293, 2015. 1

[2] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. Continuous control with deep reinforcement learning. *CoRR*, abs/1509.02971, 2015. 1

[3] D. Loiacono, L. Cardamone, and P. L. Lanzi. Simulated car racing championship: Competition software manual. *CoRR*, abs/1304.1672, 2013. 1

[4] T. Matiisen. Demystifying deep reinforcement learning. *Computational Neuroscience*, 2015. 1

[5] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. A. Riedmiller. Playing atari with deep reinforcement learning. *CoRR*, abs/1312.5602, 2013. 1