Reinforcement learning vs supervised learning

A comparison on DonkeyCar autonomous driving

Master's Thesis submitted to the
Faculty of Informatics of the *Università della Svizzera Italiana*
in partial fulfillment of the requirements for the degree of
Master of Science in Informatics
Major in Artificial intelligence

presented by
Giorgio Macauda

under the supervision of
Prof. Paolo Tonella
co-supervised by
PhD Matteo Biagiola

September 2022

I certify that except where due acknowledgement has been given, the work presented in this thesis is that of the author alone; the work has not been submitted previously, in whole or in part, to qualify for any other academic award; and the content of the thesis is the result of work which has been carried out since the official commencement date of the approved research program.

Giorgio Macauda
Lugano, 12 September 2022

To my beloved

Someone said . . .

Someone

Abstract

This is a very abstract abstract. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

Nulla malesuada porttitor diam. Donec felis erat, congue non, volutpat at, tincidunt tristique, libero. Vivamus viverra fermentum felis. Donec nonummy pellentesque ante. Phasellus adipiscing semper elit. Proin fermentum massa ac quam. Sed diam turpis, molestie vitae, placerat a, molestie nec, leo. Maecenas lacinia. Nam ipsum ligula, eleifend at, accumsan nec, suscipit a, ipsum. Morbi blandit ligula feugiat magna. Nunc eleifend consequat lorem. Sed lacinia nulla vitae enim. Pellentesque tincidunt purus vel magna. Integer non enim. Praesent euismod nunc eu purus. Donec bibendum quam in tellus. Nullam cursus pulvinar lectus. Donec et mi. Nam vulputate metus eu enim. Vestibulum pellentesque felis eu massa.

Quisque ullamcorper placerat ipsum. Cras nibh. Morbi vel justo vitae lacus tincidunt ultrices. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. In hac habitasse platea dictumst. Integer tempus convallis augue. Etiam facilisis. Nunc elementum fermentum wisi. Aenean placerat. Ut imperdiet, enim sed gravida sollicitudin, felis odio placerat quam, ac pulvinar elit purus eget enim. Nunc vitae tortor. Proin tempus nibh sit amet nisl. Vivamus quis tortor vitae risus porta vehicula.

Fusce mauris. Vestibulum luctus nibh at lectus. Sed bibendum, nulla a faucibus semper, leo velit ultricies tellus, ac venenatis arcu wisi vel nisl. Vestibulum diam. Aliquam pellentesque, augue quis sagittis posuere, turpis lacus congue quam, in hendrerit risus eros eget felis. Maecenas eget erat in sapien mattis porttitor. Vestibulum porttitor. Nulla facilisi. Sed a turpis eu lacus commodo facilisis. Morbi fringilla, wisi in dignissim interdum, justo lectus sagittis dui, et vehicula libero dui cursus dui. Mauris tempor ligula sed lacus. Duis cursus enim ut augue. Cras

ac magna. Cras nulla. Nulla egestas. Curabitur a leo. Quisque egestas wisi eget nunc. Nam feugiat lacus vel est. Curabitur consectetuer.

Suspendisse vel felis. Ut lorem lorem, interdum eu, tincidunt sit amet, laoreet vitae, arcu. Aenean faucibus pede eu ante. Praesent enim elit, rutrum at, molestie non, nonummy vel, nisl. Ut lectus eros, malesuada sit amet, fermentum eu, sodales cursus, magna. Donec eu purus. Quisque vehicula, urna sed ultricies auctor, pede lorem egestas dui, et convallis elit erat sed nulla. Donec luctus. Curabitur et nunc. Aliquam dolor odio, commodo pretium, ultricies non, pharetra in, velit. Integer arcu est, nonummy in, fermentum faucibus, egestas vel, odio.

Sed commodo posuere pede. Mauris ut est. Ut quis purus. Sed ac odio. Sed vehicula hendrerit sem. Duis non odio. Morbi ut dui. Sed accumsan risus eget odio. In hac habitasse platea dictumst. Pellentesque non elit. Fusce sed justo eu urna porta tincidunt. Mauris felis odio, sollicitudin sed, volutpat a, ornare ac, erat. Morbi quis dolor. Donec pellentesque, erat ac sagittis semper, nunc dui lobortis purus, quis congue purus metus ultricies tellus. Proin et quam. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos hymenaeos. Praesent sapien turpis, fermentum vel, eleifend faucibus, vehicula eu, lacus.

Acknowledgements

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

Nulla malesuada porttitor diam. Donec felis erat, congue non, volutpat at, tincidunt tristique, libero. Vivamus viverra fermentum felis. Donec nonummy pellentesque ante. Phasellus adipiscing semper elit. Proin fermentum massa ac quam. Sed diam turpis, molestie vitae, placerat a, molestie nec, leo. Maecenas lacinia. Nam ipsum ligula, eleifend at, accumsan nec, suscipit a, ipsum. Morbi blandit ligula feugiat magna. Nunc eleifend consequat lorem. Sed lacinia nulla vitae enim. Pellentesque tincidunt purus vel magna. Integer non enim. Praesent euismod nunc eu purus. Donec bibendum quam in tellus. Nullam cursus pulvinar lectus. Donec et mi. Nam vulputate metus eu enim. Vestibulum pellentesque felis eu massa.

Quisque ullamcorper placerat ipsum. Cras nibh. Morbi vel justo vitae lacus tincidunt ultrices. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. In hac habitasse platea dictumst. Integer tempus convallis augue. Etiam facilisis. Nunc elementum fermentum wisi. Aenean placerat. Ut imperdiet, enim sed gravida sollicitudin, felis odio placerat quam, ac pulvinar elit purus eget enim. Nunc vitae tortor. Proin tempus nibh sit amet nisl. Vivamus quis tortor vitae risus porta vehicula.

Fusce mauris. Vestibulum luctus nibh at lectus. Sed bibendum, nulla a faucibus semper, leo velit ultricies tellus, ac venenatis arcu wisi vel nisl. Vestibulum diam. Aliquam pellentesque, augue quis sagittis posuere, turpis lacus congue quam, in hendrerit risus eros eget felis. Maecenas eget erat in sapien mattis porttitor. Vestibulum porttitor. Nulla facilisi. Sed a turpis eu lacus commodo facilisis. Morbi fringilla, wisi in dignissim interdum, justo lectus sagittis dui, et vehicula libero dui cursus dui. Mauris tempor ligula sed lacus. Duis cursus enim ut augue. Cras

ac magna. Cras nulla. Nulla egestas. Curabitur a leo. Quisque egestas wisi eget nunc. Nam feugiat lacus vel est. Curabitur consectetuer.

Suspendisse vel felis. Ut lorem lorem, interdum eu, tincidunt sit amet, laoreet vitae, arcu. Aenean faucibus pede eu ante. Praesent enim elit, rutrum at, molestie non, nonummy vel, nisl. Ut lectus eros, malesuada sit amet, fermentum eu, sodales cursus, magna. Donec eu purus. Quisque vehicula, urna sed ultricies auctor, pede lorem egestas dui, et convallis elit erat sed nulla. Donec luctus. Curabitur et nunc. Aliquam dolor odio, commodo pretium, ultricies non, pharetra in, velit. Integer arcu est, nonummy in, fermentum faucibus, egestas vel, odio.

Sed commodo posuere pede. Mauris ut est. Ut quis purus. Sed ac odio. Sed vehicula hendrerit sem. Duis non odio. Morbi ut dui. Sed accumsan risus eget odio. In hac habitasse platea dictumst. Pellentesque non elit. Fusce sed justo eu urna porta tincidunt. Mauris felis odio, sollicitudin sed, volutpat a, ornare ac, erat. Morbi quis dolor. Donec pellentesque, erat ac sagittis semper, nunc dui lobortis purus, quis congue purus metus ultricies tellus. Proin et quam. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos hymenaeos. Praesent sapien turpis, fermentum vel, eleifend faucibus, vehicula eu, lacus.

Contents

Figures

Tables

Chapter 1

Introduction

Reinforcement Learning has proven to be a very general framework to learn decision making task that are generally modelled as Markov Decision Processes (MDP), so, as a discrete-time stochastic control process [van Otterlo and Wiering, 2012]. Even though RL is well established and has been widely investigated in simulated environments where an agent needs to select the best set of actions to accomplish a certain task, moving it the real world is often tedius. The focus of this thesis is on DonkeyCar, a cheap remote controlled car, and in a reinforcement learning algorithm so that it can learn to drive autonomously on a toy track through a camera. Firstly, the feasability of this goal is investigated in a simulated environment, and then moved to the real world. However, as described by Viitala et al. [2020], it is well know that learning this task, in this setting, is not possible from raw images. Thus, representation learning is used to compress observations and extract relevant features. Alongside, an investigation is done to determine wheter in our context is better to use an autoencoder or a variational autoencoder. Furthermore, given that in real world we lack the supervision of the environment, several attempt are made to define a reward function that is suitable in both the simulated and the real environment. In fact, the only supervision available in this setting, is a human that can tell the algoritmh when the car is off track and the episode must terminate, in simulation, instead, a measure of the cross track error is available. Established that the model can learn, we reproduce successfully the simulated agent in real world. Finally, an unsucceful attempt is made in adapting the already trained simulated agent to the real world, without extra training, with the use of a cyclegan that is capable of transforming a simulated image into a pseudo-real image. In pratice, we want to make the agent see images similar to those used in training. In essence, we wanted to propose an extremly simple sim to real framework, which would drastically reduce training costs and would make simpler and more reliable the benchmarking. Given that the DonkeyCar's microcontroller is not powerful enough, an off-policy reinforcement algorithm (SAC) was choosen in order reduce the workload of the microcontroller and move the actual training to an external server with enough resources to accomplish the desired goals.

Chapter 2

Background

2.1 Reinforcement Learning

Reinforcement Learning (RL) is a branch of machine learning, alongside supervised learning and unsupervised learning, that defines a set of algorithms meant to learn how to act in a specific environment without the need of labelled data to learn from.

The algorithm defines the agent that learns a given task, for example, walking, driving and playing a game, by trial and error, while interacting with an environment which can be real or simulated. Whenever the agent makes a set of good actions it receives a positive reward, which makes such actions more likely in the future. State, action and reward are the most important concepts in RL. The state represents the current situation of the environment. If the agent is a humanoid robot and the task is walking, one possible state representation is the positions of its actuated joints. The action set or space in case of continouos domain, describes what the agent can do in a particular state. In the humanoid robot example above, the action space is a n-dimensional vector where each dimension represents the torque command to each of the n joint motors. Finally the reward is a measure of how good are the actions carried out by the agent.

The reward function, usually human-designed, assigns a score to the action taken by the agent. Every action that leads to a *good* state increases the score and viceversa every *bad* action decreases it. As described in Figure 2.1 the agent interacts with the environment in discrete time steps. At time t it gets the current state s_t and the associated reward r_t then the action a_t is chosen from the set of available actions. After receiving the chosen action, the environment moves to a new state s_{t+1} and the reward r_{t+1} is given back to the agent. The total discounted reward to be maximized is:

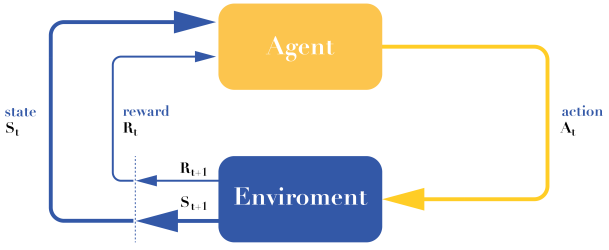


Figure 2.1. Basic reinforcement learning

$$R = \sum_{t=0}^{T} \gamma^t r_t \tag{2.1}$$

where T is the time horizon (enventually ∞), $\gamma \in [0,1)$ is the discount factor which allows R to be a finite value in case of $T = \infty$ and makes future rewards worth less than immediate reward. The total discounted reward function is fundamental to the agent in order to learn and optimize a policy function π:

$$\pi : A \times S \rightarrow [0,1] \qquad \pi(a,s) = \Pr(a_t = a \mid s_t = s) \tag{2.2}$$

The policy is a mapping that gives the probability of taking action a in state s. By following the policy the agent takes the action that maximizes the reward. However, the policy, especially during training, is not deterministic. This is due to one of the fundamental challenges in RL, i.e. the exploration-exploitation dilemma [Sutton and Barto, 2018]. Indeed, the agent needs to repeat the actions it already know to be rewarding but, at the same time, it needs to explore the environment to discover actions that can lead to an even higher reward. The final goal of the algorithm is to learn a policy that maximizes the expected cumulative reward.

$$J(\pi) = \mathbb{E}_\pi[\sum_{t=0}^{T} \gamma^t r(s_t, a_t)] \tag{2.3}$$

There are multiple ways to learn the optimal policy $\pi^*(s)$. The first one is called *Value iteration*, which exploits the state value function $V^\pi(s)$ and the action value function $Q^\pi(s,a)$. The state value function V is the expected return starting from the state s and following the policy π:

$$V_\pi(s) = E_\pi[\sum_{t=0}^{T-1} \gamma^t r_t \mid s_t = s] \tag{2.4}$$

while the action value function Q is the expected return starting from the state s, following the policy π, taking action a:

$$Q_\pi(s,a) = E_\pi[\sum_{t=0}^{T-1} \gamma^t r_t \mid s_t = s, a_t = a] \tag{2.5}$$

There is an important reletionship between Equations 2.4 and 2.5, in fact they can be written in terms of each other:

$$V_\pi(s) = \sum_{a \in A} \pi(a \mid s) * Q^\pi(s,a) \tag{2.6}$$

$$Q_\pi(s,a) = \sum_{s' \in S} P(s' \mid s,a)[r(s,a,s') + \gamma V_\pi(s')]. \tag{2.7}$$

where P is the state transition matrix that gives the probability of reaching the next state s' from state and r is the immediate reward.

In *Value Iteration*, we start from a random intialized V and the algorithm, illustrasted in Listing 2.1, repeatedly updates Q and V values until they converges, with the guarantee that they will converges to the optimal values.

```
1 Initialize V(s) to arbitrary values
2 Repeat
3     for all s in S
4         for all a in A
5             Q(s,a) = E[r|s,a] + γ ∑_{s'∈S} P(s'|s,a)V(s')
6         V(s) = max_a Q(s,a)
7 until V(s) converges
```

Listing 2.1. Value iteration pseudo code from Alpaydin [2014]

Finally the policy π can be inferred from the Q function with:

$$\pi(s) = argmax_a Q(s,a) \tag{2.8}$$

Since the agent only cares about finding the optimal policy, it could happen that the policy converges before the value function. Therefore, the so-called *Policy Iteration* algorithm seeks to learn the policy directly by updating it at each step as shown in Listing 2.2

```
1 Initializea policy π' arbitrarily
2 Repeat
3     π = π'
4     Compute the values using π
5         V_π = E[r|s,π(s)] + γ ∑_{s'∈S} P(s'|s,π(s))V_π(s'))
6     Improve the policy at each state
7         π'(s) = argmax_a(E[r|s,a] + γ ∑_{s'∈S} P(s'|s,a)V_π(s')))
8 until π = π'
```

Listing 2.2. Policy iteration pseudo code from Alpaydin [2014]

Policy iteration is also guaranteed to converge to the optimal policy and it often takes less iterations to converge than the value iteration algorithm.

A major problem arises when the environment is not entirely known to the agent or is too big that is unfeasible to store all the Q and V values in a table. As well as, in the case of a continuous action space. Deep RL algorithms introduces Deep Neural Networks in order to approximate Q and V instead of storing them in huge tables. Function approximation allows also a better generalization of states never seen before, or with partial information, by exploiting values of similar states.

Another important difference between RL algorithms that is worth a brief mention, is in the way the algorithm updates the policy. On-Policy methods evaluates and improve the same policy which is being used to select actions. Off-Policy methods evaluates and improve a policy that is different from the policy that is used for action selection bringing many advantages. Firstly, it allows a better exploration of new trajectories. Secondly, the agent can learn from demonstrations and finally it allows parallel learning speeding up the convergence.

2.2 OpenAI Gym interface

Gym is an open source library that defines a standard API to handle training and testing of RL agents, while providing a diverse collection of simulated environments.

The environment is of primary importance to a RL algorithm since it defines the world of the agent in which the agent lives and operates. The standard interface designed by Gym, makes it easier to interact with environments, both made available by Gym and externally developed. The Gym interface is simple, pythonic, and capable of representing general RL problems. Cart-pole, shown in Figure 2.2, is a classic example of a Gym environment. A pole is attached by an un-actuated joint to a base, which moves along a straight track. The pendulum is placed upright on the base and the goal is to balance the pole by moving the base to the left and right. The action set include two actions, move right and move left. The observation space includes the base position, the base velocity, the pole angle and the pole angular velocity. The documentation provides a reference template shown in Listing 2.3 that describes what are the fundamental methods a gym environment should implement to work properly. Any exisisting environment built with Gym implements the following few methods which are enough to run any basic RL algorithm or eventually override them in case of custom environments:

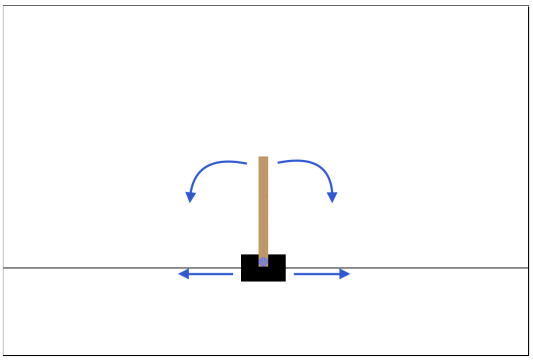


Figure 2.2. Cart-pole in Gym

```
1  class GymTemplate(gym.Env):
2    def __init__(self):
3      pass
4    def step(action):
5      pass
6    def reset():
7      pass
8    def render(mode='human'):
9      pass
10   def close(self):
11     pass
```

Listing 2.3. "Gym template"

- **init:** every environment should extend gym.Env and contain the variables *observation_space* and *action_space* specifying the type of possible observations and actions using spaces.Box or spaces.Discrete.
- **step:** this method is the primary interface between environment and agent, it takes as input the action and return informations (observation, reward, done) about the current state.
- **reset:** this method resets the environment to its initial values returning the initial state of the environment.
- **render:** this method pops up a window rendering the environment when a parameter *mode='human'* is passed.
- **close:** this method performs any necessary cleanup before closing the program.

Beside the environment, gym provides a set of wrappers to modify an existing environment without having to change the underlying code directly. The three main common things a wrapper wants to do are:

- Transform actions before applying them to the base environment
- Transform observations that are returned by the base environment
- Transform rewards that are returned by the base environment

The given set of wrappers to reach any of the aforementioned goal includes: *ActionWrapper*, *ObservationWrapper*, *RewardWrapper*. Alongside with more complex wrappers which che be found in the offical documentation. Furthermore, custom wrappers can be implemented by inheriting from *Wrapper*.

2.3 Soft Actor Critic - SAC

The soft actor critic algorithm [Haarnoja et al., 2018] illustrasted in Figure 2.3 is a state-of-the-art RL algorithm designed to outperform prior on-policy and off-policy methods in a range of continuous control benchmark tasks.

It aims and succeeds to reduce the sample complexity, since even relatively simple task can require millions of steps of data collection, and brittleness in convergence. A poor sample efficiency in deep RL methods may be due to on-policy learning since it requires new samples to be collected for each gradient step. In order to improve the sample efficiency, SAC draws on the maximum entropy framework. It introduces to the objective 2.3 an entropy maximization term which aids stochastic policies by augmenting the objective with the expected entropy of the policy:

$$J(\pi) = \mathbb{E}_{\pi}[\sum_{t=0}^{T} \gamma^t r(s_t, a_t) + \alpha H(\pi(\cdot \mid s_t))] \tag{2.9}$$

where α is a temperature parameter that weighs the entropy term and thus controls the policy stochasticity.

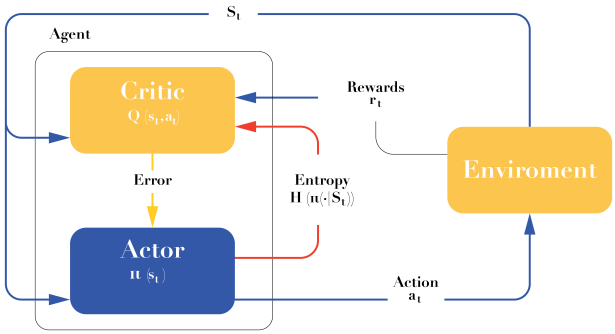


Figure 2.3. Soft actor critic

The maximum entropy framework used by SAC has several desirable properties. Firstly, it incentives a wider exploration during training. Secondly, the policy can capture multiple modes

of near-optimal behavior. Lastly, it noticeably increases the learning speed over state-of-the-art methods that optimize the standard objective.

2.4 Generative Adversarial Networks - GAN

Generative Adversarial Network is a framework introduced by Goodfellow et al. [2014] for training generative models in an unsupervised fashion. GANs can be used, for example, to generate visual paragraph [Liang et al., 2017], realistic text [Zhang et al., 2017], photographs of human faces [Karras et al., 2017], Image-to-Image translation [Isola et al., 2017].

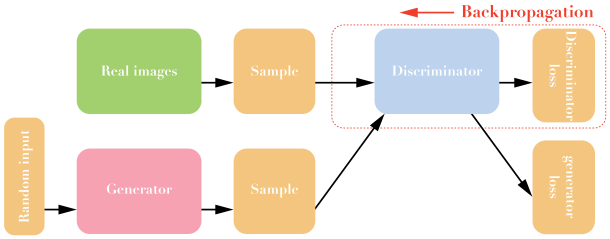


Figure 2.4. GAN diagram

The learning process involves two neural networks that are trained in an adversarial way, i.e. with an opposing objective.

Indeed, as described in Figure 2.4, the generator G generates inputs (e.g images) starting from random noise and the discriminator D needs to distinguish whether such inputs belong to the original dataset or not. GANs fall under the branch of unsupervised learning since the training process does not need labelled data as the generator is guided by the discriminator in order to generate inputs that resemble those of the original dataset.

D is trained to maximize the probability of returning the correct label to both training examples and G samples. At the same time G attempt to minimize:

$$L_G = log(1 - D(G(z))) \tag{2.10}$$

where $D(G(z))$ is the probability of $G(z)$ coming from the original dataset (p_X), then $1 - D(G(z))$ defines the probability of $G(z)$ not coming from p_X. Since the generator wants to fool the discrimitar, it needs to minimize 2.10.

In order to learn the generator's distribution p_g over the training dataset X such that $p_g \approx p_X$, a prior is defined on input noised variables $p_z(z)$, then it is mapped into the data space with the generator $G(z, \theta_g)$. Beside that, the discriminator $D(x; \theta_d)$, with $x \sim p_g$, outputs a single value which estimates the probability that x came from the dataset X rather than p_g. D and G are both differentiable function represented by a neural network with θ_d and θ_g respectively being their parameters.

In other words, the discriminator and the generator play a minimax game with the value function $V(G, D)$:

$$min_G max_D V(G, D) = \mathbb{E}_{x \sim \rho_{data}(x)}[log D(x)] + \mathbb{E}_{z \sim \rho_z(z)}[log(1 - D(G(z)))] \tag{2.11}$$

However, G is initally weak since it has not learned a good p_g yet, thus it cannot deceive the discriminator and D can say with high confidence (probability close to 1) that $G(z)$ does not come from p_X and therefore 2.10 would be at the minimum. To solve the aforementioned problem, 2.10 can be substituded with the maximization of the formula below in order to always have enough gradient:

$$log(D(G(z))) \tag{2.12}$$

2.5 CycleGAN

Image-To-Image translation is a complex task where the goal is to transform an image from one domain to another and viceversa, as shown in Figure 2.5.

Prior papers have been presented to translate images, however they often require paired training examples between the domains [Sangkloy et al. [2016], Karacan et al. [2016]]. Such paired datasets can be very expensive or even impossible to gather, as in the case of object transfiguration ($horse \leftrightharpoons zebra$).

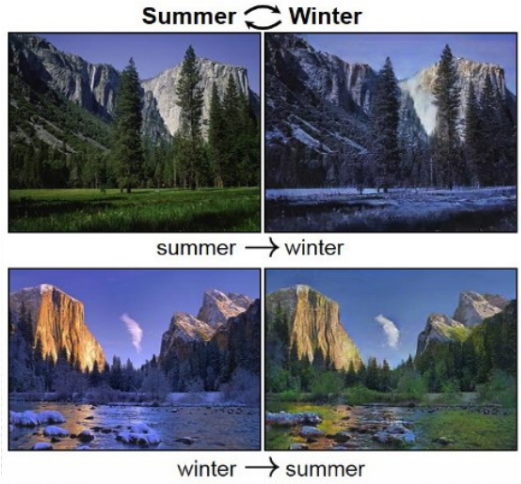


Figure 2.5. Image-to-image translation example [Zhu et al., 2017].

Cycle-Consistent Adversarial Networks from Zhu et al. [2017] (CycleGAN), aim to solve this problem in an unsupervised fashion. The main goal is to learn, using an adversarial loss, a mapping $G : X \rightarrow Y$ such that the image $G(x)$ with $x \in X$ is indistinguishable from a real image $y \in Y$. Since the mapping is highly under-costrained, an inverse mapping $F : Y \rightarrow X$ is introduced, together with a cycle-consistency loss to enforce $F(G(x)) \approx x$ and viceversa. To accomplish the goal two discriminator D_X and D_Y are provided. D_X tries to distinguish between training samples $\rho_{data}(X)$ and their translations $F(Y)$ and viceversa for D_Y. The full objective 2.13 includes the adversarial losses and the cycle-consistency loss to encourage a consistent translation from one domain to the other:

$$L(G, F, D_X, D_Y) = L_{GAN}(G, D_Y, X, Y) + L_{GAN}(F, D_X, Y, X) + \lambda L_{cyc}(G, F) \tag{2.13}$$

where the loss $L_{GAN}(G, D_Y, X, Y)$ and $L_{GAN}(F, D_X, Y, X)$ are described in 2.11 from standard GANs and the following is the cycle-consistency loss:

$$L_{cyc}(G, F) = \mathbb{E}_{x \sim \rho_{data}(x)}[\|F(G(X)) - x\|_1] + \mathbb{E}_{y \sim \rho_{data}(y)}[\|G(F(y)) - y\|_1] \tag{2.14}$$

where λ is a temperature parameter to define the importance of such loss in Equation 2.13 and $\|\cdot\|_1$ is the L1 norm or rather the sum of the magnitudes of the vectors in a space, a measure of the distance between vectors.

2.6 AutoEncoder and Variational AutoEncoder

2.6.1 AutoEncoder

AutoEncoders (AEs) are artificial neural networks that fall under the branch of unsupervised learning since they learn efficient encoding into a latent space without the supervision of labelled data. They are generally used for vary purposes, for example, dimensionality reduction,

image compression, image denoising, image generation, feature extraction and sentence generation [Hinton and Salakhutdinov [2006], Cheng et al. [2018], Gondara [2016], Hou et al. [2017], Liu and Liu [2019]].

Taking as example the case of image dimensionality reduction, an AE is composed of two main parts, an encoder E and a decoder D.

$$E_\phi : X \to Z \qquad\qquad D_\theta : Z \to X^{'} \tag{2.15}$$

where $X = \mathbb{R}^{mxn}$ and $Z = \mathbb{R}^k$ for some m, n, k and $k \ll mxn$ to reach the goal of dimensionality reduction. The optimal case is reached when $X = X^{'}$. They are both parametrized function with ϕ and θ being respectively their parameters, to put it another way the parameters of the neural networks, generally multilayer perceptrons, of which they are composed of. As shown in Figure 2.6, the main goal of the encoder is to learn a mapping of each observation of the dataset $x \in X$ into a latent space of smaller dimensionality. Since a label is not available, in order to measure the quality of the embedded image into the latent space, the decoder is used to reconstruct to image and then compute the loss.

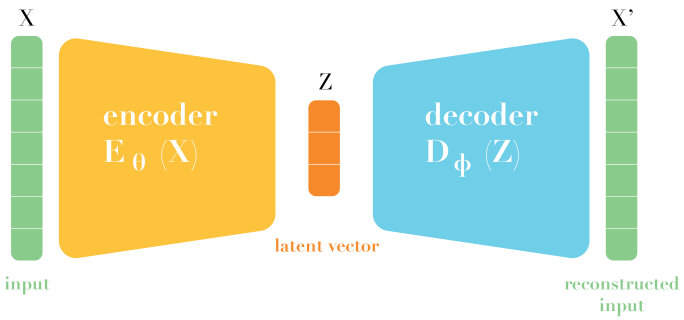


Figure 2.6. AE diagram

In other words, the encoder maps an image $x \in X$ into the latent space producing $z = E_\phi(x)$ with $z \in Z$, then z is reconstructed by the decoder to bring it back to the orginal space $x' = D_\theta(z)$ with $x^{'} \in X^{'}$. Finally, $x^{'}$ can be used as a label with any distance measure $d(x, x^{'})$. Thus the loss to be minimized is so computed:

$$L(\theta, \phi) = d(x_i, D_\theta(E_\phi(x_i))) \tag{2.16}$$

In Figure 2.7 is shown a typical example of how the MNIST dataset looks like once mapped into the latent space. After the AE is trained, a random generated observation could be given to it to generate a new sample from the dataset distribution. However, there are parts of the latent space that does not correspond to any data point. Thus using sample from the *white space* will not generate any meaningful image. That is why a basic so-designed AE cannot be used as a general generative model, eventhough a random sample that falls into any cluster in the latent space can certainly produce a meaningful image, also an image never seen in training even if only with small variations.

AEs, results to be much more capable in data compression since the latent space of a linear autoencoder strongly resembles the eigenspace achieved during the principal component

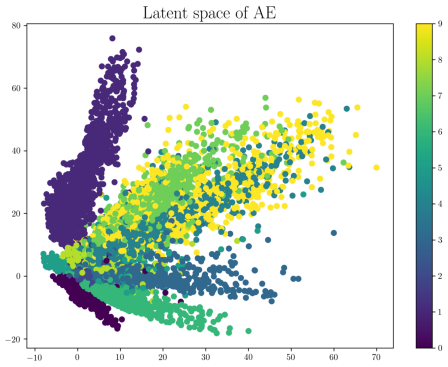


Figure 2.7. Example AE latent space Z on MNIST datasetk

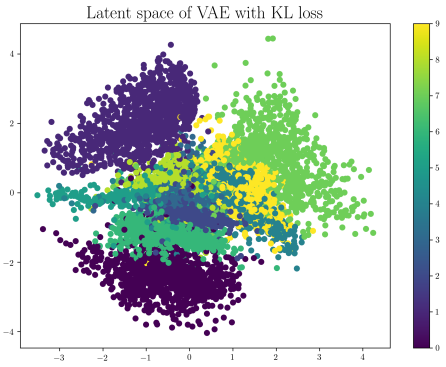


Figure 2.8. Example VAE latent space Z on MNIST dataset

analysis of the data, with the added value of the non-linearity. Thus making AEs capable of learning rather powerful representations of the input data in lower dimensions with much less information loss.

2.6.2 Variational AutoEncoder

Variational AutoEncoders (VAEs) addresses the problem of *strong localization* of data point into the latent space thus providing a more powerful generative capability then AEs.

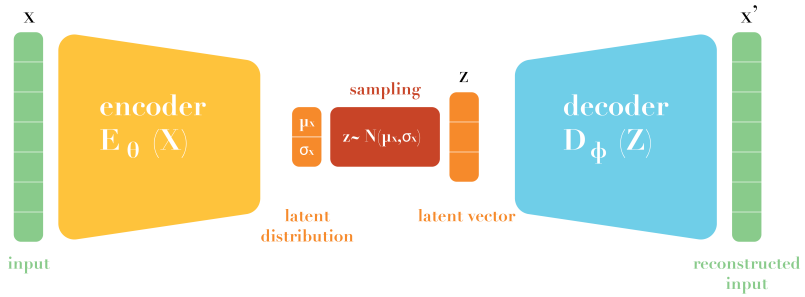


Figure 2.9. VAE diagram

As shown in Figure 2.9 only a small change with respect to AEs is introduced, the encoder instead of mapping directly samples into the latent space outputs parameters of a pre-defined distribution (usually Normal) in the latent space for every input. Then z is produced by sampling from a normal distribution with the outputted parameters.

In other words, the encoder, starting from an image $x \in X$, produces the gaussian parameters $[\mu_x, \sigma_x] = E_\phi(x)$, then z is sampled from a normal distribution $z \sim \mathcal{N}(\mu_x, \sigma_x)$. Consequently, the decoder brings it back to the orginal space $x' = D_\theta(z)$ with $x^{'} \in X^{'}$. Finally, $x^{'}$ can be used as a label with any distance measure $d(x, x^{'})$. Thus the loss to be minimized is so

computed:

$$L(\theta, \phi) = d(x_i, D_\theta(E_\phi(x_i))) + KL[\mathcal{N}(\mu_x, \sigma_x), \mathcal{N}(0,1)] \tag{2.17}$$

where the firs term is equivalent to 2.16 and KL is the Kulback-Leibler divergence, a measure of how one probability distribution is different from another. The KL divergence acts as a regularization term by enforcing predicted distributions to be close to the identity, giving to the latent space two main properties, continuity (close points in the latent space should be close also when decoded) and completeness (any point sampled from the latent space should always be meaningful once decoded) as shown in Figure 2.8.

2.7 DonkeyCar

DonkeyCar, shown in Figure 2.10, is an open source DIY platform providing software and hardware tools for the development of self-driving car algorithms. The basic car is a simple remote controlled car that can be 3D printed or bought as a kit for a reasonable and affordable price. The main models recommended by the official documentation are:

- Exceed Magnet Blue, Red
- Exceed Desert Monster Green
- Exceed Short Course Truck Green, Red
- Exceed Blaze Blue, Yellow, Wild Blue, Max Red

These cars are electrically identical but have different tires and mounting. They are also equipped with brushed motors which make ML training easier since they handle rough driving surfaces well and are inexpensive. The car can be customized with additional sensors as LIDARs and IMUs to provide more information about the surroundings of the car.

In particular, the car used for the purposes of this thesis, is a basic donkey car equipped with an 8-megapixel IMX219 sensor that features an 160 degree field of view. It is capable of taking photos with a resolution of 3280x2464 and video recording up to a resolution of 1080p at 30 frames per seconds. In order to process all the information coming from the camera, control the motors and run the self-driving car softwarethe self-driving car software it is equipped with an NVIDIA Jetson Nano microcontroller. To power comes from a LiPO battery 11.1V and 2200mAh that powers the electric motor and the microcontroller. Additionally, to expand the operational life of the car, a power-bank can be added to exclusively power the microcontroller, while the LiPO battery is dedicated at powering the engine. A DonkeyCar can be remotely controlled either with a joypad or directly by the software.

Figure 2.10. Assembled donkeycar

Chapter 3

Related works

When training a Reinforcement Learning model there are several problem that can arise and need to be tackled, especially when the learning process moves from simulated environments to the real world. In this section a few methods, particularly useful for the purposes of this thesis, are presented.

3.1 State Representation Learning

Reinforcement Learning is a very general method for learning sequential decision making tasks. On the other hand, Deep Learning has become in recent years the best set of algorithms capable of Representation Learning (ReL). Representation learning algorithms are designed to extract abstract features from data. A mix of the two provides a particularly powerful framework for learning state representation, especially when dealing with real world environments that tend to be much more complex and unpredictable than simulated environments. In particular, State Representation Learning (SRL) is a specific type of ReL where extracted features are in low dimension, evolves in time and are affected by an agent's action. The low dimensionality allows easier interpretation by humans, helps in handling the curse of dimensionality and speeds up the policy learning process. Thus, SRL is well suited for Deep Reinforcement Learning applications. Lesort et al. [2018] presented a complete survey that covers the state-of-the-art on SRL. Feature extraction and learning is a wide topic that tries to discover features that characterize data by decomposing them. Taking as example a dataset of portraits, a set of features that can compose each picture can be the hair color, skin color, face shape and so on. Training a neural network to learn those features may be accomplished by compressing the image into a smaller vector, discarding all the useless informations for the model, where each dimension would represents a feature like the ones just described. However, a feature not necessarily describes a humam readable aspect of the data but can be even without a semantic meaning. In particular, SRL exploits the time steps, actions, and eventually rewards, to transform observations into states, a low dimensionality vector that contains the most relevant features to learn a particular policy that acts as a supervisor. The better the policy or the speed with which it is learned, the more the features extracted are significant to the model.

3.2 Improving sample efficiency

In order to define the state of the environment in our experiment we use a camera as described in Section 4.2. However, training a model from high-dimensional images with reinforcement learning is difficult, in previous Section 3.1 we described an approach to mitigate those difficulties. In this section we present a specific method that is used for the purposes of this thesis.

Deep convolutional encoders can learn a good representation even though they generally require large amounts of training data. Using off-policy methods and adding an auxiliary task with an unsupervised objective can naturally improve sample efficiency and add stability in optimization but they often lead to suboptimal policies as described in Yarats et al. [2019]. They revisit the concept of adding an encoder to off-policy RL methods and provide a simple and effective autoencoder-based off-policy method that can be trained end-to-end. The main focus is in finding the optimal way of training a RL agent using SRL.

In practice, in their experiment, the AE is composed of a convolutional encoder that maps an image observation to a low dimension vector into the latent space and a deconvolutional decoder the reconstruct the latent vector back to the original image. While several auxiliary objectives could be used to improve the learned representation, they target on the most general and widely applicable, an image reconstruction loss avoiding task dependent losses. After that, a SAC algorithm is used to learn some task from the latent state of the environment.

There are two options, the first one seeks to train the agent alternately with the encoder with both kept indipentent from each other. So the AE is pretrained and then a few iteration are used to improve the AE with its own loss, later on, the agent is trained with the encoder kept constant. The algorithm keeps iterating between this two phases until convergence. The second option, seeks to learn a latent representation that is well aligned with the underlying RL objective, thus the AE network is updated with the gradient coming from the actor, critic and the AE itself. However, this attempt of joint representation learning was proven unsuccesful. For this reason, our focus is on the first alternating representation learning. The last thing to define is how often the encoder should be updated. From the tested tasks is evident that it should be updated at the end of every episode, however, even if it is never updated after the first pre-training, the result are still very good. Beside that, an on going update would require more computational power to complete all the algorithm steps in the same amount of time. Since this work aims to solve a real-time problem, it is necessary that a certain number of frames are processed per second that is why the single pretrain is preferable in the context of microcontroller, PC without a GPU and over-the-air communication. Even though this could lead to a slightly longer training, it would speed up the single iteration.

3.3 Smooth exploration

When moving a RL algorithm from a simulated environment to the real world, the unstructured step-based exploration often very successful in simulation, leads to unstable motion patterns. This may results in poor exploration, longer training and even damages to the robot's motors that can be expensive. Raffin et al. [2022] handle the issue by including a state-dependent exploration (SDE) to current Deep Reinforcement Learning algorithms. In most RL algorithm the standard for exploration is to sample a noise vector from a Gaussian distribution indipendent from the environment and the agent, and then it is added to the controller output. SDE replaces the sampled noise with a state-dependet exploration function. This results in smoother explo-

ration and less variance for each episode. In practice the solutions is as simple as sampling a noise vector as a function of the actual state s_t and adding it to the choosen action.

3.4 Learning to Drive - L2D

Learning to Drive (L2D) [Viitala et al., 2020] is a low-cost benchmark for real world autonomous driving learned through reinforcement learning. Since training this types of RL algorithms can be very expensive due to the nature of trial-and-error learning and the cost of a real car, the benchmark are carried out using a DonkeyCar as described in Section 4.2. The authors also provide the source code in order to let every one implent his own RL algorithm to solve DonkeyCar autonomous driving task which we, for the sake of simplicity, use as a baseline of our experiments. They demonstrate that existing RL algorithms, like Imitation learning, SAC+VAE and Dreamer, can learn to drive the car from scratch. SAC+VAE is also our choise since it performs the best in terms of High-Speed Control. Beside that, they also show as SAC trained directly from the images is not able to learn, which is why we do not consider this option in our test, insted we focus on the aforementioned State Representation Learnign as they did.

Chapter 4

Experimental setup

4.1 The track and the environment

The track we used is called USI track, shown in Figure 4.1. It strongly resembles one used by Viitala et al. [2020] in Learning to Drive. We do have a simulated version built in Unity and an actual printed track. The choice fell on this track since it is complete, it includes a straightaway, right turn, left turn, wide turn and very steep turn. Beside that, Viitala et al. [2020] already proved that the agent can learn on this type of track and the focus of this thesis is more on replicating a real agent learning to self-drive in real world and not creating a new model with a particular feature.

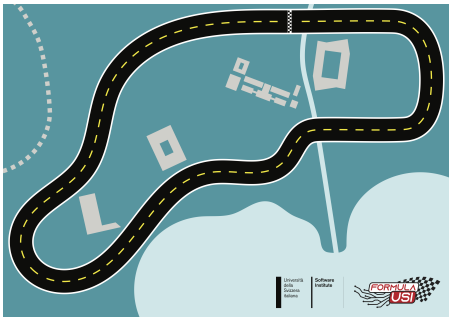


Figure 4.1. Real USI track TODO

Figure 4.2. Simulated USI track

The learning of the agent, as shown later on, is straightforward, except the very steep turn which is considerably harder than the others. This difficulty is due to the limited steering angle of the robot and in the real world the aforementioned adversity is even more marked as well see.

The default starting line in both tracks is where the Donkey is placed in Figure 4.2. Certainly, in the real track it is an imaginary line that we use as a starting point and of course the laying of the car at each episode beginning cannot be exact but approximate. Beside that, there are a few checkpoints, approximately highlighted with a cross in Figure 4.2, troughout the track that can be used as starting points depending on the learning strategy chosen. The simulator provides the following possibilities:

- **Start:** The episode start always at the starting line.
- **Checkpoint:** The episode start at the latest checkpoint reached during the previous episode.
- **All:** All the checkpoints are used cyclically starting from the starting line and proceeding one by one forward for each episode.
- **Random:** The starting point is chosen randomly, between the available checkpoints, at the beginning of each episode.

Furthermore, the simulator let us choose where a lap ends. It can end where the DonkeyCar started or at the starting line.

4.2 DonkeyCar

The real Remote Controlled DonkeyCar is essentially a standard DonkeyCar as described in Section . To recap it is a remote controlled car equipped with a microcontroller NVIDIA Netson Nano and a camera sensor. The RGB pictures are taken at a resolution of $320x240$ and at $20hz$ (20 frames per second), which means the algorithm must finish all iteration steps within $0,05$ seconds otherwise it would skip some frames and the learning or the driving may be compromised by the agent's non-responsiveness. This limitation is present only in real world since the simulator time can be slowed down to meet the needs. In our setting we have a standard 3 cell LiPO battery of 11.1V and 2200 mAh to power just the motors and the controller. During the training in the real world we often noticed a slow regression in term of speeds of the car, iteration after iteration. However, this problem can be solved by disconnecting the LiPO battery for a moment from time to time to restore full speed, which is why we suspect this problem may be caused by the battery. Furthermore, an external power bank with 10000mAh/37Wh of capacity and an output of 5V and 2.4A powers the Jetson Nano which with this capacity, is more than enough to overcome the longevity of the LiPO battery.

4.3 Training modality

4.3.1 Simulation

In simulation, the training of the agent is straightforward since all the operation are done on the host machine, moreover it is not required a GPU machine to accomplish all the steps in time, at least until the cyclegan is introduced. Even though an on-policy RL algorithm could be implemented, an off-policy algorithm is chosen since in real world, in our setting, the on-policy method is not replicable given the limited computational power of the microcontroller. Beside that, we want to compare the same type of algorithm in the two type of environments. In pratice, a pretrained AE/VAE provides a representation of the state in the form of a latent vector, the agent drives with a policy kept constant during the episode and all the frames and actions are collected into a buffer. When the simulator reports that the car has crashed or went out of track more than a predefined distance, the episode terminates. The episode ends also when the car has reached the starting point or, in our setup, it reaches 1000 steps. Finally, at the end of the episode, a policy is trained using the collected buffer and the new parameters are used to update the driving policy.

4.3.2 Real world

Since the microcontroller equipped by the DonkeyCar is a low capability calculator, a few precautions need to be taken in order to to train the agent. Firstly, as mentioned above, an off-policy method like SAC allows to relocate the actual training, and consequently the very expensive gradient back-propagation, to another machine with more resources. Secondly, the use of representation learning (AE/VAE) allows to reduce significantly the size of the RL neural networks and moreover the pre-training of the encoder can be done in advance on the host machine speeding up the process. In practice the functioning is similar to the one seen in previous Section 4.3.1. The microcontroller operates the driving policy, collect the image, forward it through the AE/VAE, then the agent chose an action based on that representation. This process is repated until a human supervisor ends the episode for the car gone off the track. The episode also terminates when the DonkeyCar reaches, in our setup, 1000 steps. Hereafter, all the steps information, like latents vectors, actions and rewards are collected into a buffer up to a predefined size and are sent through the network to the host calculator that actually train the policy at the end of the episode. After the training, the new parameters are sent back to the DonkeyCar and the process is repeated until convergence.

4.4 Communication - MQTT

As described in Section 4.3.2, when training in real world, the host machine and the DonkeyCar must communicate wirelessly multiple times during the training. MQ Telemetry Transport (MQTT) is the most used messaging protocol for the Internet of Things (IoT). It includes all the rules that define how devices can write (publish) and read (subscribe) data over the internet. The sender (Publisher) and the receiver (Subscriber) communicate via topic and are decoupled from each other. The connection between them is handled by an MQTT broker that filters all incoming messages and distributes them correctly to the Subscribers of the topic. In pratice any device can publish a message on a topic, then the broker take care of dispatching it to subscribers of that topic. In particular we used the HiveMQ broker that allows the connection of up to 100 clients with no cost. The topics defined to manage the communication between the host machine and the DonkeyCar are:

- **Stop car:** The host machine writes a signal on this topic, that is constantly monitorated by the DonkeyCar, to inform that the episode must terminate.
- **Replay buffer:** Once the episode terminate, all the collected information by the DonkeyCar are sent to the host machine through this topic.
- **Replay buffer received:** The host machine uses this topic to acknowledge DonkeyCar that it has received the buffer.
- **Parameters:** Once the training is complete, the host machine sent the updated neural network parameters through this topic.
- **Start episode:** The host machine uses this topic to acknowledge DonkeyCar that a new episode can start.
- **Speed modifier:** This topic can be used by the host machine to inform the DonkeyCar that it must change its throttle by the sent value that can be either positive or negative.

Notice that this protocol is not complety reliable so some precautions and check must be done when implenting it, especially in real-time system where some actions cannot be delayed.

4.5 Dataset

With regard to the dataset we need to define two types of dataset. A dataset composed of images collected on the simulator to train the relative autoencoder and after that the simulated RL agent, and a similar one composed of real images collected on the printed track. A few example of each one are respectively shown in Figures 4.3 and 4.4.

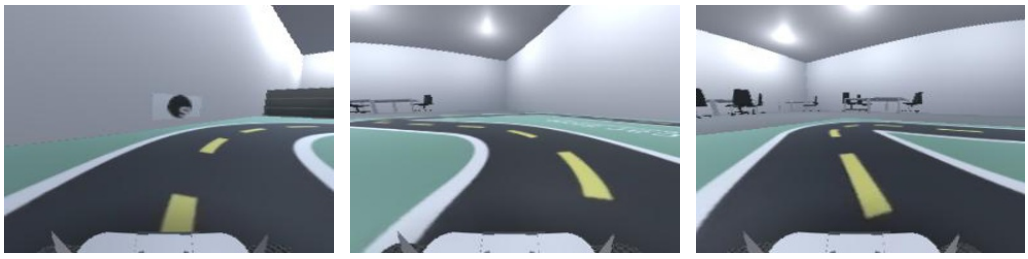


Figure 4.3. Images extracted from the simulated dataset

Figure 4.4. Images extracted from the real dataset

During the experiments resulted that a dateset of $\sim$ 10000 pictures was enough to reach our goals, furthermore notice that smaller datasets may not be sufficient for the encoder to learn a good representation. To collect each of the datasets are enough $\sim$ 10 minutes if we run the algorithm at $20hz$ (20 frames per second) as we did. Beside that, all the pictures from the real world must be collected with a certain environmental condition that should remain consistent in time, also during the training of the agent to avoid problems. In our case, it was collected with all windows closed and the with maximum light to make it easy to be replicated.

Since we want our RL agent to focus exclusively on the track we found convinient to crop the top 100 rows of each pictures to remove the background, and to reduce the complexity of our algorithm, we downscale each images from $320x140$ to $160x80$ before feeding them to the encoder. The resulting pictures are shown in Figures 4.5 and 4.6. Note that during training, the training set is split in validation and training set with a ratio 20/80 and the test set is collected apart and consist of $\sim$ 1000 images for each dataset.

Finally, for the cyclegan, the dataset can be even smaller $\sim$ 5000 pictures for both real and simulation. No crop is applied and a resize to $256x256$ pixels is done to match the network size

Figure 4.5. Examples of cropped simulated images

Figure 4.6. Examples of cropped real images

provided by Zhu et al. [2017]. And the test sets match the ones used for the encoders. After applying the cyclegan in our experiments, the pictures are reshaped and cropped to match the need of the autoencoder.

Chapter 5

Experiments

5.1 AE vs VAE

Since our main goal is to create an end-to-end RL algorithm composed of an encoder followed by SAC we first need to decide wheter to use an autoencoder or a variational autoencoder. In other words, we want to explore if the stochasticity of a VAE can help in learning a good representation of the actual state. In order to do so, we follow a simple approach, train multiples both AEs and VAEs to compare how much information they are able to recover from the latent vector with an MSE loss on average. As described above, we run a single pre-train on the dataset and then the chosen encoder remains unchanged for the entire duration of the RL agent training. There are varius choices that must be made before proceeding with the RL training. Firts we have to choose between AE and VAE, then the size of the latent vector z and finally wheter to use data augmentation or not to improve the generalization of our model. In particular we consider an AE and a VAE network composed as respectively described in Listings 5.1 and 5.2. In Tables 5.1,5.2, 5.3,5.4 are shown the result of trainings, each encoder has been trained three times to increase the reliability of the results and the average is reported in the tables. As we see in all cases the encoder performs better when augmentation is disabled, furthermore increasing z_size to 64 dimensions results in a better reconstruction loss. Finally, the VAE performs slightly better then AE. In figure 5.1 is shown what the reconstructed images looks like for the chosen VAE trained without augmentation and a latent vector size of 64 dimension. All the other encoder reconstructions are shown in APPENDIX TODO.

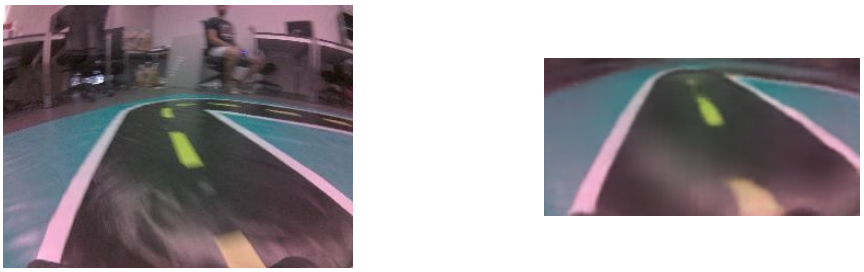


Figure 5.1. Real world image processed after cropping with a VAE, z_size=64 and no augmentation. Reconstruction_loss=112

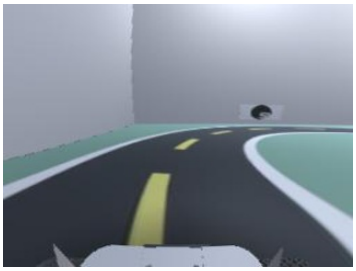


Figure 5.2. Simulator image processed after cropping with a VAE, z_size=64 and no augmentation. Reconstruction_loss=17

Z_SIZE	AUGMENTATION	MEAN	STD	MAX	MIN
32	False	121.54	102.42	795.44	45.61
	True	164.57	95.51	783.03	65.13
64	False	103.54	79.14	588.14	40.84
	True	137.24	74,02	611,81	63,05

Table 5.1. AE trained in simulation - reconstruction loss

Z_SIZE	AUGMENTATION	MEAN	STD	MAX	MIN
32	False	377.07	87.53	756.7	239.46
	True	493.84	99.40	807.67	289.99
64	False	311.1	78.5	695.65	177.77
	True	411.37	77.30	647.68	241.87

Table 5.2. AE trained in real world - reconstruction loss

Z_SIZE	AUGMENTATION	MEAN	STD	MAX	MIN
32	False	59.1	60.41	620.93	18.88
	True	116.31	71.11	771.88	51.10
64	False	45.15	43.49	480.22	14.34
	True	112.17	59.79	573.19	54.28

Table 5.3. VAE trained in simulation - reconstruction loss

Z_SIZE	AUGMENTATION	MEAN	STD	MAX	MIN
32	False	227.4	44.74	418.7	140.12
	True	263.87	52.29	478.26	172.70
64	False	184.56	36.86	347.59	96.7
	True	230.66	42.24	402.67	156.61

Table 5.4. VAE trained in real world - reconstruction loss

Listing 5.1. AE network

```
(encoder): Sequential(
  (0): Conv2d(3, 16, kernel_size=(4, 4), stride=(2, 2))
  (1): ReLU()
  (2): Conv2d(16, 32, kernel_size=(4, 4), stride=(2, 2))
  (3): ReLU()
  (4): Conv2d(32, 64, kernel_size=(4, 4), stride=(2, 2))
  (5): ReLU()
  (6): Conv2d(64, 128, kernel_size=(4, 4), stride=(2, 2))
  (7): ReLU()
)
(encode_linear): Linear(in_features=3072, out_features=z_size, bias=True)
(decode_linear): Linear(in_features=z_size, out_features=3072, bias=True)
(decoder): Sequential(
  (0): ConvTranspose2d(128, 64, kernel_size=(4, 4), stride=(2, 2))
  (1): ReLU()
  (2): ConvTranspose2d(64, 32, kernel_size=(4, 4), stride=(2, 2))
  (3): ReLU()
  (4): ConvTranspose2d(32, 16, kernel_size=(5, 5), stride=(2, 2))
  (5): ReLU()
  (6): ConvTranspose2d(16, 3, kernel_size=(4, 4), stride=(2, 2))
  (7): Sigmoid()
)
```

Listing 5.2. VAE network

```
(encoder): Sequential(
  (0): PreProcessImage()
  (1): Conv2d(3, 32, kernel_size=(4, 4), stride=(2, 2))
  (2): ReLU()
  (3): Conv2d(32, 64, kernel_size=(4, 4), stride=(2, 2))
  (4): ReLU()
  (5): Conv2d(64, 128, kernel_size=(4, 4), stride=(2, 2))
  (6): ReLU()
  (7): Conv2d(128, 256, kernel_size=(4, 4), stride=(2, 2))
  (8): ReLU()
  (9): PostProcessImage()
)
(fc_mu): Linear(in_features=6144, out_features=z_size, bias=True)
(fc_var): Linear(in_features=6144, out_features=z_size, bias=True)
(decoder_input): Linear(in_features=z_size, out_features=6144, bias=True)
(decoder): Sequential(
  (0): ConvTranspose2d(256, 128, kernel_size=(4, 4), stride=(2, 2))
  (1): ReLU()
  (2): ConvTranspose2d(128, 64, kernel_size=(4, 4), stride=(2, 2))
  (3): ReLU()
  (4): ConvTranspose2d(64, 32, kernel_size=(5, 5), stride=(2, 2))
  (5): ReLU()
)
(final_layer): Sequential(
  (0): ConvTranspose2d(32, 3, kernel_size=(4, 4), stride=(2, 2))
  (1): PostProcessImage()
  (2): Sigmoid()
)
```

5.2 RL algorithm

5.2.1 Reward function

Designing a reward function that can work on both simulated and real environments is not trivial. In simulation, the environment can provide a better supervision than a human can do. In our real setup we have only the camera frames as states and so no other sensor is available, even though they can be used. In simulation, instead, the environment can tell us how far the donkeycar is from the center line of the track, with this value, we can decrease the reward function by a value proportional to the distance of the car from the center of the roadway as done in Equation 5.1. So the reward function is:

$$r_t = 1 + throttle\_reward + cte\_penalty + \begin{cases} if\,done & crash\_error \\ else & 0 \end{cases} \tag{5.1}$$

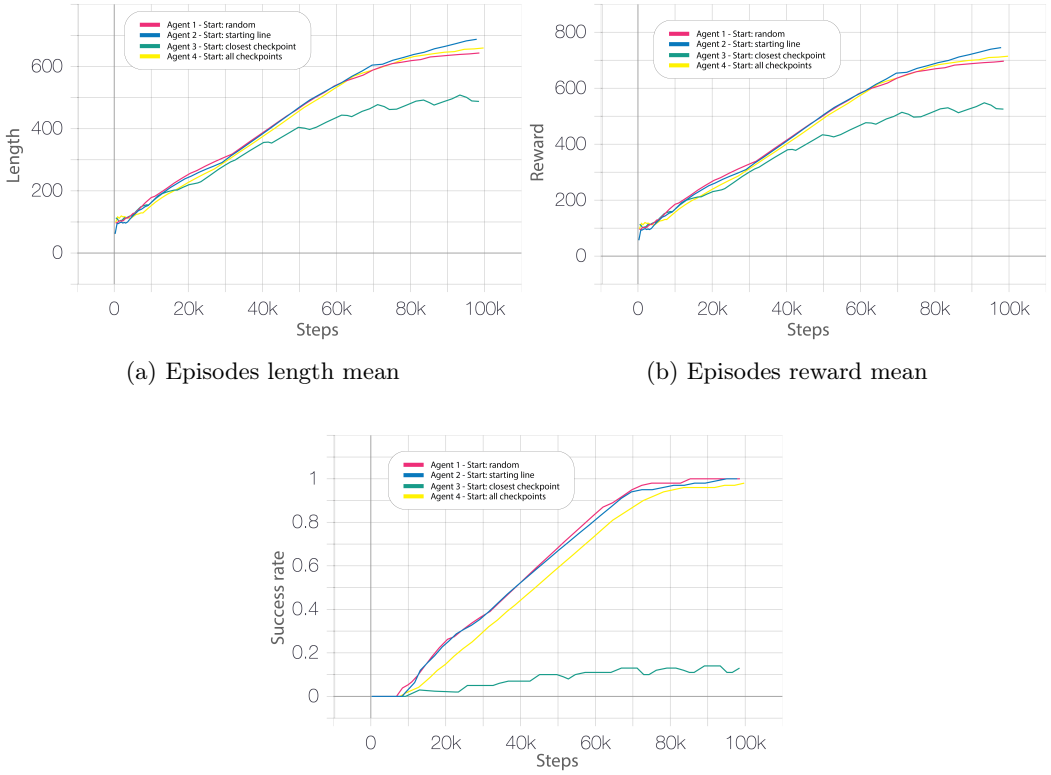
where 1 is given for each step taken by the agent in order to incentivize the completion of the largest possible number of steps. The throttle reward incentives the car to go as fast as possible, however in our setup, the throttle is kept constant for the purposes of this thesis. The Cross Track Error (CTE) penalty, is a negative value to incentive the car to stay as close as possibile to the center of the roadway. In case the car exceeds a predefined maximum CTE or crashes, the penalty becomes very high and it is added through the *crash_error* term. This first reward function is used to test our algorithm and after proving it can work, we need to adapt it such that it can work also in real world. To tackle the issue we simply remove the CTE penalty with a negative value to be activated only when an episode ends due to human intervention. The final reward function that has proven to work in both environment and we use in our trainings is:

$$r_t = 1 + throttle\_reward + \begin{cases} if\,done & crash\_error \\ else & 0 \end{cases} \tag{5.2}$$

Since we want the real and the simulated version of our agent as similar as possibile Equation 5.2 is used in both cases.

5.2.2 Training the simulated RL agent

As a baseline for RL algorithm we used the source code provided by Viitala et al. [2020]. Their algorithm allows both simulated and real training, however training on simulation with communication being over-the-internet is more computationally expensive and more prone to errors. Thus, for the simulation, we refactor the algorithm such that the communication happens locally. Beside that, his algorithm uses an AE which need to be changed with the VAE chosen above. Given that the simulator provides several training modality we test them all to find out which one is the best, to eventually save time in real world. To chose where the car should starts a new lap we trained 4 different model, one for each option provided by the simulator, to identify which one eventually converges and if it does. The quality measures, illustrated below in Figure 5.3, are the *Episode success rate* that shows how many laps has been complete on average, the *Episode Reward mean* and the *Episode Length mean*. All the model have been trained for 100k iterations, with a different starting point. *Agent 1* started each lap at a random checkpoint, *Agent 2* started always at the starting line, *Agent 3* at the latest checkpoint reached during the last episode and, finally, *Agent 4* cyclically uses all the checkpoints. During our experiments we noted that, in the best case, a lap may takes $\sim$ 350 iterations to be completed. Our agents



(a) Episodes length mean

(b) Episodes reward mean

(c) Success rate mean

Figure 5.3. Agents trained in simulation. Each agent has been trained with a different starting modality and has been trained for 100k steps.

are all able to successfully learn to drive with a strict maximum CTE except for the agent that started new laps from the latest checkpoint. There are two interesting evidences that come out of those trainings. The first one is that even though the success rate mean approaches 100%, that means the agent is able to consistently finish laps, the reward mean keeps growing. This shows a limitation in the reward function used, given that the agent gets a reward for every steps, it learns to finish the lap following the longest path it has discovered. Beside that, the best way to lengthen the path is a zig-zag behavior that allows also a doubling of the reward per lap. Secondly, the agent that start at the latest checkpoint keeps improving the reward up to more than an equivalent completed lap, however it never finishes a lap as described in Figure 5.3c. The reason behind this strange behavior is that the agent found a bug in the simulator used, as shown in Figure 5.4. Essentially, there is a a little spot, off track, close to the steepest turn where the CTE is not is correctly detected and consequently the episode is not terminated. The fact this behavior happens only on this agent stands in the training modality. When the agent reach this checkpoint, he cannot easily reach the next checkpoint given the toughness of the turn, instead it finds much easier to explore the bugged spot which is almost right in front of it when it approaches the turn.

To furter test the trained agents, for 10 laps it is measured how many times the lap has been completed, how many times the agents crashes, and finally after disabling the CTE threshold,

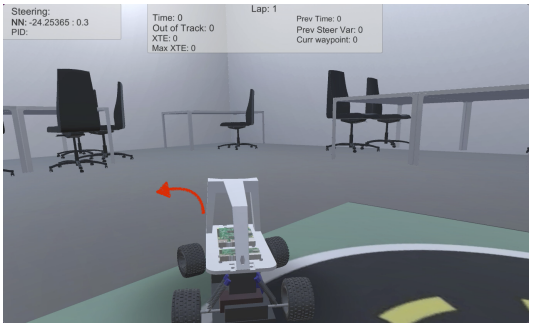


Figure 5.4. Spotted bug in the simulator

AGENT	OOT	OBE	LAPS	AVG LENGTH	AVG REW
1	0	0	10	595	644
2	0	4	10	599	647
3	0	0	10	624	676
4	10	29	0	460	495

Table 5.5. Agents results averaged over 10 laps. Out Of Track (OOT) measures crashes, Out of Bound Error measure how many times it exceed the max CTE, and finally LAPS counts the completed laps.

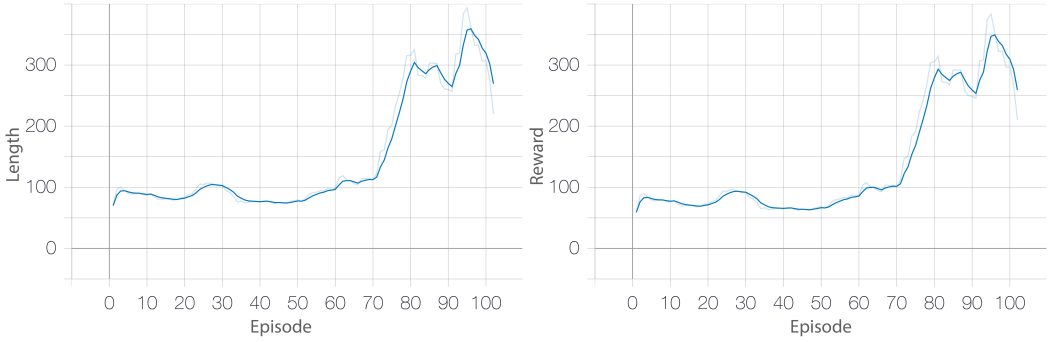
if the agent is still able to recover and finish the lap without crashing. The result are presented in Table 5.5.

Thus, excluding *Agent 4* because of the simulator's bug, three agents learned to drive successfully and in most cases they always stay entirely on track without the need of any additional sensor and with the only problem of the shaky driving which is still acceptable for the purposes of this thesis.

5.2.3 Training the real RL agent

In real world instead the source code provided by Viitala et al. [2020] is kept untouched beside the encoder, with the main goal being to replicate their results but with a more performing VAE as resulted in our tests. Given that in real world the simulator supervision is not available, all the agents tested in previous section are good candidates to be used, also *Agent 4* that cannot explore anymore the simulator's bug. However from the tests, it resulted that all the agents struggle to learn driving an entire lap in reasonable time, except *Agent 4* that start his laps at the latest checkpoint reached in the last episode. Human supervision is about stopping the episode anytime the car exceeds the track boundaries with all 4 wheels, while the host machine automatically stops the car when it reaches 1000 steps ($\sim$ 2.5 laps). In figure 5.5 are shown the performances in training. The laying of the car on the latest checkpoint has been intentionally approximate on the area close to the checkpoint. This brought a main advantage, the agent learns quicker since it is able to see the area in front of it from many points of view. It results useful when the car will start to cross many checkpoints per episode, since the direction from which the car arrives to a checkpoint can vary a lot, it has been trained to drive on many possibile

trajectories and will join the various sections well. The overall training procedure results to be simple and fast (∼ 30 minutes) to get an entire lap completed. In five to twenty episodes, the first two turn are learned decently. Most of the time is spent on the steepest turn. As shown in Figure 5.5a, the graph is characterized by ups and downs, as soon as the car starts at the starting line, it learns quickly, then, when the steepest curve is reached it struggle to overcome it and the when it enventually does the length start to increase again. The process is repeated until it is almost consistently able to finish a lap. As soon as the agent has learned the steepest turn, it generalizes well on following turn, in fact little time is spent on them.



(a) Episodes length

(b) Episodes reward

Figure 5.5. Agent trained in real world starting each lap on the latest checkpoint

Unfortunately, in real world more metrics to measure the quality of the driving and to make comparisons with the simulated agents are not available.

5.3 Sim to Real

SimToReal (S2R) or viceversa aims in deploying model trained in one environment to the other. In our case, since the real world environment, in our setup, does not provide enough metrics to benchmark our real world agents, we aim to make it works also in simulation. Another advantage brought by this approach is that an agent trained in simulation, can be moved into the real world and this would result in less expensive training procedures. The idea is to pre-train a CycleGan [Zhu et al., 2017] for image trasfiguration. In fact, the CycleGan is able to move an image into another domain keeping the original structure unaltered, but applying the style of the other domain. Thus we leverage this property to transform images seen by the simulated camera of the DonkeyCar into what it would see in real world or viceversa. Then, a real agent will eventually be able to drive on the simulator since it does see pseudo-real images. On the other hand, in order to drive a real car with a simulated agent, our DonkeyCar has not enough computational power, hence it could not run in time a CycleGan, that has millions of parameters, in order to make the real car see pseudo-simulated images and drive with the simulated agent. However, the problem can be circumvented by training an agent entirely on simulation but with pseudo-real images.

Chapter 6

Future work and conclusion

Another reward function was initally tested trying to improve the total time an agent takes to complete a lap:

$$r_t = -0.1 + throttle\_reward + cte\_penalty + \begin{cases} if\,done & crash\_error \\ else & 0 \end{cases} \tag{6.1}$$

The idea behind this function is that any step gives a negative reward and thus the agent must finish a lap in the smallest number of step to maximize the total episode reward. Minimizing the number steps means also finding the shortest way and consequently reducing the total time spent for a lap. Unfortunately, this approach did not let the agent learn to drive decently, at least in reasonable time.

Appendix A

Some retarded material

Lara

A.1 It's over...

Ciaooo mbareeeeee

Glossary

Bibliography

Ethem Alpaydin. *Introduction to Machine Learning*. The MIT Press, 2014. ISBN 0262028182, 9780262028189.

Zhengxue Cheng, Heming Sun, Masaru Takeuchi, and Jiro Katto. Deep convolutional autoencoder-based lossy image compression. In *2018 Picture Coding Symposium (PCS)*, pages 253–257, 2018. doi: 10.1109/PCS.2018.8456308.

Lovedeep Gondara. Medical image denoising using convolutional denoising autoencoders. In *2016 IEEE 16th International Conference on Data Mining Workshops (ICDMW)*, pages 241–246, 2016. doi: 10.1109/ICDMW.2016.0041.

Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks, 2014. URL `https://arxiv.org/abs/1406.2661`.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor, 2018. URL `https://arxiv.org/abs/1801.01290`.

G. E. Hinton and R. R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, 2006. doi: 10.1126/science.1127647. URL `https://www.science.org/doi/abs/10.1126/science.1127647`.

Xianxu Hou, Linlin Shen, Ke Sun, and Guoping Qiu. Deep feature consistent variational autoencoder. In *2017 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 1133–1141, 2017. doi: 10.1109/WACV.2017.131.

Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A. Efros. Image-to-image translation with conditional adversarial networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

Levent Karacan, Zeynep Akata, Aykut Erdem, and Erkut Erdem. Learning to generate images of outdoor scenes from attributes and semantic layouts. 12 2016.

Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. Progressive growing of gans for improved quality, stability, and variation, 2017. URL `https://arxiv.org/abs/1710.10196`.

Timothée Lesort, Natalia Díaz Rodríguez, Jean-François Goudou, and David Filliat. State representation learning for control: An overview. *CoRR*, abs/1802.04181, 2018. URL `http://arxiv.org/abs/1802.04181`.

Xiaodan Liang, Zhiting Hu, Hao Zhang, Chuang Gan, and Eric P. Xing. Recurrent topic-transition gan for visual paragraph generation. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, Oct 2017.

Danyang Liu and Gongshen Liu. A transformer-based variational autoencoder for sentence generation. In *2019 International Joint Conference on Neural Networks (IJCNN)*, pages 1–7, 2019. doi: 10.1109/IJCNN.2019.8852155.

Antonin Raffin, Jens Kober, and Freek Stulp. Smooth exploration for robotic reinforcement learning. In Aleksandra Faust, David Hsu, and Gerhard Neumann, editors, *Proceedings of the 5th Conference on Robot Learning*, volume 164 of *Proceedings of Machine Learning Research*, pages 1634–1644. PMLR, 08–11 Nov 2022. URL https://proceedings.mlr.press/v164/raffin22a.html.

Patsorn Sangkloy, Jingwan Lu, Chen Fang, Fisher Yu, and James Hays. Scribbler: Controlling deep image synthesis with sketch and color, 2016. URL https://arxiv.org/abs/1612.00835.

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, second edition, 2018. URL http://incompleteideas.net/book/the-book-2nd.html.

Martijn van Otterlo and Marco Wiering. *Reinforcement Learning and Markov Decision Processes*, pages 3–42. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012. ISBN 978-3-642-27645-3. doi: 10.1007/978-3-642-27645-3_1. URL https://doi.org/10.1007/978-3-642-27645-3_1.

Ari Viitala, Rinu Boney, and Juho Kannala. Learning to drive small scale cars from scratch. *CoRR*, abs/2008.00715, 2020. URL https://arxiv.org/abs/2008.00715.

Denis Yarats, Amy Zhang, Ilya Kostrikov, Brandon Amos, Joelle Pineau, and Rob Fergus. Improving sample efficiency in model-free reinforcement learning from images. *CoRR*, abs/1910.01741, 2019. URL http://arxiv.org/abs/1910.01741.

Yizhe Zhang, Zhe Gan, Kai Fan, Zhi Chen, Ricardo Henao, Dinghan Shen, and Lawrence Carin. Adversarial feature matching for text generation. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 4006–4015. PMLR, 06–11 Aug 2017. URL https://proceedings.mlr.press/v70/zhang17b.html.

Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *Computer Vision (ICCV), 2017 IEEE International Conference on*, 2017.