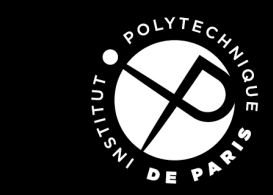
Exploration of reinforcement learning algorithms for autonomous vehicle visual perception and control

Florence Carton



INSTITUT POLYTECHNIQUE DE PARIS

NNT : 2021IPPAE007

Thèse de doctorat

Exploration of reinforcement learning algorithms for autonomous vehicle visual perception and control

Thèse de doctorat de l'Institut Polytechnique de Paris
préparée à École Nationale de Techniques Avancées

École doctorale n°626 École doctorale de l'Institut Polytechnique de Paris (EDIPP)
Spécialité de doctorat : Informatique, Données, IA

Thèse présentée et soutenue à Palaiseau, le 31 Mai 2021, par

FLORENCE CARTON

Composition du Jury :

Thierry Chateau Professor, Université Clermont Auvergne (ISPR)	Président
Fabien Moutarde Professor, Mines ParisTech (Centre for Robotics)	Rapporteur
Christian Wolf Associate Professor, HDR, INSA Lyon (LIRIS)	Rapporteur
Alain Dutech Research Fellow, HDR, INRIA (LORIA)	Examinateur
David Filliat Professor, ENSTA Paris (U2IS)	Directeur de thèse
Jaonary Rabarisoa Research Scientist, CEA (DIASI/SIALV/LVA)	Co-Encadrant
Quoc Cuong Pham Dr, Research Scientist, CEA (DIASI/SIALV/LVA)	Co-Encadrant, Invité

À Victor
À mes parents
À ma famille et mes amis

Figure 1: Reinforcement Learning: learning by trial and error, by interacting with the environment that sends reward signal - positive or negative.
CALVIN AND HOBBES ©Watterson. Reprinted with permission of ANDREWS MCMEEL SYNDICATION. All rights reserved.

Remerciements

Ce manuscrit clôture 3 ans et demi de thèse, et je tiens à remercier toutes les personnes qui m'ont permis d'achever ces travaux.

Je voudrais tout d'abord remercier Fabien Moutarde et Christian Wolf d'avoir accepté de rapporter cette thèse, ainsi que Thierry Chateau et Alain Dutech d'avoir accepté d'être mes examinateurs.

Je remercie également Jaonary Rabarisoa de m'avoir encadrée pendant ces trois années, et Quoc Cuong Pham, pour m'avoir donné la possibilité d'effectuer cette thèse au CEA, mais également pour m'avoir co-encadrée et conseillée de manière toujours utile et pertinente.

Un grand merci à mon directeur de thèse David Filliat pour ses précieux conseils, mais également pour sa bienveillance, son soutien et sa disponibilité.

Je voudrais remercier chaleureusement toute l'équipe LVA du CEA qui m'a accueillie pendant 3 ans. J'ai particulièrement apprécié la bonne ambiance et l'émulation scientifique au sein du laboratoire. Je voudrais en particulier remercier ma co-bureau Astrid Orcesi pour ces trois années de bonne humeur, Jérémy Morceaux avec qui j'ai particulièrement aimé travailler, et enfin, milles merci à Patrick Hède et Nicolas Granger pour m'avoir beaucoup aidée à effectuer les très (très) nombreuses expériences qui ponctuent ce manuscrit. Merci également à Angélique Loesch, Adballah Benzine, Camille Dupont, Bertrand Luvison, et à l'ensemble des membres du LVA, que j'étais toujours heureuse de retrouver le matin.

Je remercie également tous les membres du laboratoire U2IS de l'ENSTA, où j'étais moins présente mais toujours accueillie avec beaucoup de chaleur ! Merci à Thibault Toralba, Vyshakh Palli Thazha, Antonin Raffin, Timothée Lesort, Alexandre Chapoutot, François Pessaux, Victor Talpaert, Natalia Diaz et à tous mes collègues pour les bons moments et les passionnantes discussions.

Merci à tous mes amis de m'avoir changé les idées dans les moments difficiles. En particulier merci à Caroline, Thibault, Gabriel, Éléonore, Jean-Baptiste, Louis, Erwan, Delphine et Audrey.

Merci également à toute ma famille élargie, merci à tous pour les très nombreux encouragements qui m'ont fait chaud au cœur. Merci à mon papi d'avoir toujours cru en moi et à ma mamie pour son soutien et pour les délicieux gâteaux qui m'ont sans nul doute remonté le moral à de nombreuses reprises.

Je remercie infiniment mes parents de m'avoir toujours soutenue, et plus récemment de m'avoir accueillie pendant la rédaction de ce manuscrit. Merci de m'avoir encouragée et remonté le moral quand j'en avais bien besoin.

Et enfin, merci à Victor, pour ton soutien indéfectible pendant ces années et je l'espère pour toutes celles à venir.

Résumé

Les travaux développés pendant cette thèse s'intéressent à l'utilisation de l'apprentissage par renforcement pour la perception et le contrôle d'un véhicule autonome en environnement urbain. Nous assistons en effet ces dernières années à l'essor du véhicule autonome, largement entraîné par le développement et l'utilisation grandissante de l'apprentissage automatique - ou machine learning. Cependant, il reste encore beaucoup de progrès à faire avant une utilisation généralisée par le grand public. Pour atteindre le niveau d'automatisation actuel, les véhicules autonomes doivent être équipés de nombreux capteurs coûteux : caméras, radars, Lidar, capteur à ultrasons, etc... En plus d'être coûteux, ces capteurs sont sensibles aux changements tels que la météo, alors que les humains sont capables de conduire avec une simple vision frontale dans des conditions très diverses. Avec l'essor des réseaux de neurones et en particulier des réseaux de neurones convolutifs, de nombreux problèmes de vision par ordinateur, tels que la classification ou la détection d'objets, ont atteint des performances très élevées. Dans le domaine de la conduite, l'un des défis de la vision par ordinateur est de construire des systèmes intelligents capables d'analyser les images aussi efficacement que les humains. Ces systèmes doivent être capables de gérer les changements d'apparence (luminosité variable, nuit, brouillard, pluie...), ainsi que les occlusions partielles.

Dans cette thèse, pour être proche de la conduite humaine, nous nous limitons à une seule caméra embarquée comme capteur d'entrée, et étudions les performances qui peuvent être atteintes dans ce scénario.

Nous nous intéressons en particulier à la conduite de bout-en-bout, c'est à dire sans étapes de traitement intermédiaires entre la perception et le contrôle. L'action choisie par l'agent autonome sera directement calculée à partir des données d'entrée issues des capteurs. Cette approche bout-en-bout s'oppose à une architecture modulaire, où la navigation est découpée en sous-modules indépendants (perception, planification et contrôle). Privilégiée par l'industrie, la modularité présente l'avantage d'être facilement interprétable et permet aux différents modules d'être développés en parallèle par des équipes indépendantes. Cependant, l'indépendance de ces modules peut aussi être un facteur limitant. Les modules sont entraînés de manière indépendante, sans être directement lié à 'objectif final - la conduite. Il peut en résulter un système global sous-optimal - les modules sont optimaux pour une tâche spécifique, et dépendent les uns des autres. C'est pourquoi nous avons pris le parti de travailler de bout-en-bout car cette approche, plus complexe à mettre en oeuvre, permet potentiellement d'atteindre des performances qui ne sont pas limitées par les objectifs intermédiaires non directement reliés à la conduite.

Pour entraîner un tel modèle de conduite bout-en-bout, on peut distinguer deux paradigmes d'apprentissages. Le premier, l'apprentissage supervisé, utilise de nombreuses données labellisées, des démonstrations d'experts que l'agent apprend à reproduire. Le second, l'apprentissage par renforcement, fonctionne par essai-erreur grâce à un système de récompense. L'agent interagit avec l'environnement et reçoit des récompenses, positives ou négatives, en fonction des décisions prises, et a pour objectif de maximiser la quantité totale des récompenses reçues. En plus de ne pas nécessiter de données extérieures, dans de nombreux exemples, l'apprentissage par renforcement

a permis de dépasser les performances réalisées avec de l'apprentissage supervisé, ce qui le rend attractif.

Cependant, l'apprentissage par renforcement présente certains inconvénients. Les entraînements ont d'une part besoin d'une très grande quantité de données pour converger, et sont d'autre part longs et parfois instables, notamment car le niveau d'information contenu dans une récompense est inférieur à celui contenu dans la vérité terrain de l'apprentissage supervisé.

Dans ces travaux, nous proposons plusieurs contributions pour tenter de lever ces verrous.
Dans un premier temps, nous proposons une étude sur l'impact des différents paramètres pouvant influencer la convergence et les performances d'un agent de conduite autonome.
Dans une seconde partie, nous étudions la manière d'adapter notre agent autonome pour gérer la conduite conditionnelle, c'est-à-dire pour suivre des instructions GPS haut niveau. Nous proposons également un nouveau benchmark permettant d'évaluer la robustesse d'un agent autonome dans le cas où la commande haut niveau est incorrecte.

Enfin dans une troisième partie, nous étudions le problème de la généralisation, crucial pour les véhicules autonomes. Lors de ces travaux, nous nous sommes en particulier intéressé à l'ajout d'information sémantique pendant l'apprentissage de la navigation. En effet la segmentation sémantique contient la quasi totalité des informations nécessaires à la navigation. Nous avons donc exploré et comparé différentes méthodes permettant d'exploiter cette information tout en maintenant le caractère bout-en-bout de nos entraînements.

Les expériences sont menées en simulation, et des résultats quantitatifs et qualitatifs sont présentés tout au long de ce manuscrit. Ces expérimentations nous permettent de présenter un framework de conduite autonome de bout en bout avec de l'apprentissage par renforcement.

Contents

List of Figures

List of Tables

Glossary

AC	Actor Critic
CNN	Convolutional Neural Network
(D)DPG	(Deep) Deterministic Policy Gradient
DQN	Deep Q Network
DDQN	Double Deep Q Network
DL	Deep Learning
E-E dilemna	Exploration vs. Exploitation Dilemna
GAN	Generative Adversarial Network
HER	Hindsight Experience Replay
MCTS	Monte Carlo Tree Search
MDP	Markov Decision Process
ML	Machine Learning
NN	Neural Network
PG	Policy Gradient
POMDP	Partially Observable Markov Decision Process
RDPG	Recurrent Deterministic Policy Gradient
RL	Reinforcement Learning
TD	Temporal Difference
VAE	Variational Autoencoder

Chapter 1

Introduction

1.1 Context

Since seminal work in the 1980s in several laboratories around the world, autonomous driving has been steadily improving and is now close to general practical use. Recent breakthroughs, supported by major companies such as Waymo or Tesla, now enable people to be driven by their own vehicles for hundreds of kilometers within given conditions. Autonomous vehicles are very promising for safer traffic, as more than 90% of accidents are related to human error. An intelligent system is indeed never tired or distracted, and has a better reaction time than humans. By drastically limiting accidents, autonomous vehicles will also help reduce traffic jams. This should improve air quality by limiting pollution, and at the same time allow users to spend less time on the road.

Nevertheless, vehicles are not yet fully autonomous. The goal of autonomous driving is full automation, and before reaching this stage, a number of intermediate steps as defined by the NHTSA (National Highway Traffic Safety Administration) must be achieved (see Figure 1.1).

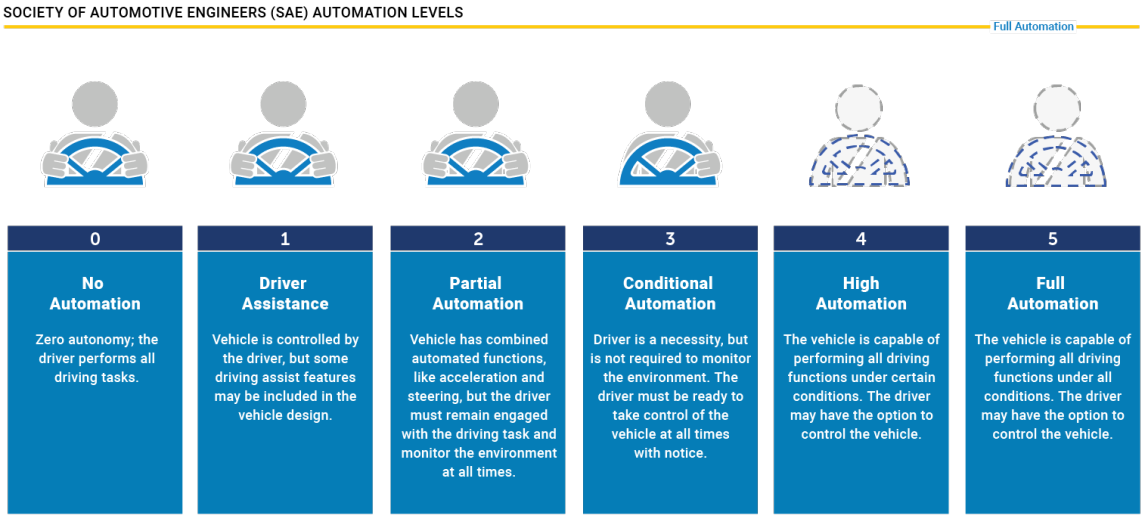


Figure 1.1: Levels of automation for autonomous vehicles defined by the NHTSA

If levels 0 to 2 are more or less advanced driving assistance, driving delegation appears at level 3, where the driver must be able to regain control at any time. Levels 4 and 5 correspond to

autonomous driving, under certain conditions for level 4, in any conditions for level 5. Not yet reached, and also facing legal and ethical problems - such as liability in case of an accident of an autonomous vehicle - the horizon of level 5 is getting closer.

1.1.1 Vision and machine learning

However, there is still a lot of progress to be made before widespread use by the general public. To achieve the current level of automation, these autonomous vehicles must be equipped with many expensive sensors: cameras, radars, Lidar, ultrasonic sensor, etc ... In addition to being extremely expensive, these sensors have limitations - like the change of weather - while humans are able to drive with a simple frontal vision. Human driving shows that it is possible to drive based on vision alone.

With the rise of neural networks and in particular convolutional neural networks, many computer vision problems, such as classification or object detection, have reached very high performances. In the driving domain, one of the challenges in computer vision is to build intelligent systems that can analyze images as efficiently as humans. These systems must be able to manage changes in appearance (varying brightness, night, fog, rain ...), as well as partial occlusions. The development of deep learning, using large databases and increasingly powerful machines, is constantly improving the performance of detection and perception tools.

1.1.2 Modular vs end-to-end approaches

The autonomous vehicles currently in circulation are based on such intelligent cameras and sensors, and composed of many independent modules. The first one is the perception module, which detects the essential elements of the environment - other road users, pedestrians, road lines, etc. This is followed by various blocks: world modeling, trajectory planning, and finally a controller that produces the output elements: acceleration/braking and steering wheel angle. This modular approach has the advantage of being easily interpretable - the modules are designed by humans - and offers the possibility that modules can be developed in parallel by independent teams.

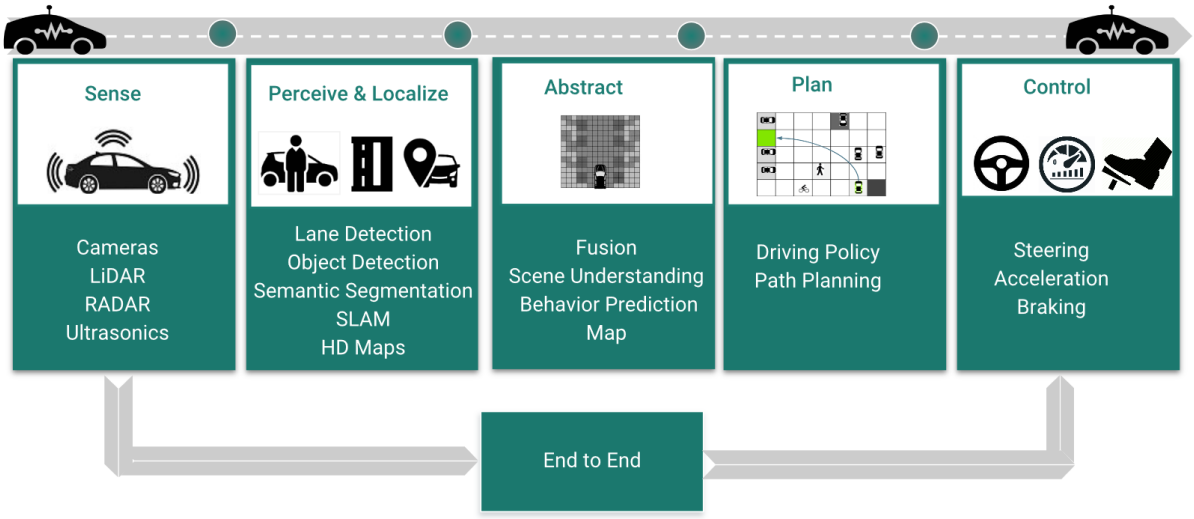


Figure 1.2: Modular vs end-to-end autonomous driving [Talpaert 2019]

However, the independence of these modules can also be a limiting factor. The modules are trained separately, with no vision of the final objective - driving. This can result in a sub-optimal

overall system - the modules are optimal for a specific task, and depend on one another. This modular approach is opposed to the end-to-end approach (see Figure 1.2). It consists of a direct optimization from the input sensors to the control commands. Less interpretable, it reduces the number of intermediaries - and thus potential errors. Although considered superior in theory by the scientific community, the industrial world remains wary of end-to-end, particularly because of the difficulty of interpreting the behavior and validating the proposed models.

1.1.3 Supervised vs reinforcement learning

Such an end-to-end driving model is learned either through expert demonstrations (supervised learning) or by exploring and improving itself through a reward system (reinforcement learning). In the case of supervised learning, the agent learns to reproduce the expert behavior from many labeled data. In reinforcement learning, the agent interacts with the environment and receives rewards, positive or negative, depending on the decisions it makes. It has no examples and its goal is to maximize the total amount of rewards it gets.

Supervised Learning	Reinforcement Learning
• Training with annotated data	• Training with reward funtion
• Requires a large number of annotated data	• No need for expert demos
• Offline training	• Interaction with the environment
• Simple and stable	• Not always convergent
• Only as good as the demo	• Can become arbitrarly good

Table 1.1: Supervised vs Reinforcement Learning

Brought to light in 2016 with Alpha Go's victory over Lee Sedol in the game of Go [Silver 2016], reinforcement learning is booming in the scientific community. Developed by Deep Mind, Alpha Go is composed of neural networks trained with both supervised and reinforcement learning. While the best chess players have long been beaten by computers, the game of Go was previously considered one of the biggest challenges for artificial intelligence. Indeed, in the game of Go, the number of possible configurations exceeds the number of atoms in the universe - 10^{80}, which prevents any exhaustive search algorithm. There are about 10^{170} possible configurations in Go, for a total of about 10^{600} possible games, against 10^{40} legal configurations and 10^{120} possible games for chess.

The power of reinforcement learning was established for good a few months later with Alpha Go Zero [Silver 2017]. While Alpha Go uses a combination of supervised learning from expert moves followed by reinforcement learning with self-play, Alpha Go Zero is trained from scratch with reinforcement learning in self-play only. After the first 72 hours of a 40 day training, Alpha Go Zero beats the Alpha Go version used for the game against Lee Sedol, which had been trained for several months.

In the case of a sequential decision problem such as a game, reinforcement learning can in

principle exceed the performance of supervised learning. Supervised learning is limited by the performance of the experts who provide the training data, whereas reinforcement learning can become arbitrarily good, as shown in the Alpha Go Zero example.

However, reinforcement learning has shortcomings, such as the long training time required - the level of information contained in a reward is lower than that contained in the ground truth of supervised learning. Reinforcement learning also needs a very large amount of data to converge, is sample-inefficient and sometimes unstable (see Table 1.1 for comparison with supervised learning).

1.2 Problematic

We propose in this thesis to study the application of end-to-end reinforcement learning to urban navigation for autonomous vehicle. The objective is to scale up reinforcement learning for autonomous driving, based only on visual input. To be close to human driving, we limit ourselves to one onboard camera as input sensor to study the performances that can be reached in such scenario. To drive in urban environment, we will also study the management of intersection with high level commands. The problems addressed are therefore the following.

Learning end-to-end driving without intermediate representations. Our objective is to perform end-to-end driving, by directly mapping visual input to driving control. Such an approach has the advantage of limiting the integration of human bias in the system and potentially leading to higher performances.

Training a complex task by reinforcement learning. Although very powerful in theory, reinforcement learning suffers from several drawbacks. Because of the low information given at each iteration, the training times are usually very long, and trainings can be unstable.

High level command handling for urban driving. In addition to driving safely, an autonomous agent must be able to follow a required path. It must therefore be able to handle an intersection, and be robust in the sense that it must give priority to the visual information it receives in the event that the high-level command is erroneous.

Generalization to an unseen environment. Finally, an autonomous vehicle must be able to drive in a place it has never seen, in weather conditions different from those seen during training.

1.3 Contributions

This thesis is the result of a collaboration between CEA List and ENSTA Paris. The general contribution of this work is to propose a reinforcement learning approach for end-to-end autonomous driving, especially in urban environment. In particular, we focused on the effect of adding semantic information on the performance and generalization capabilities of our autonomous agents. This work is the subject of the following publication:
[Carton 2021b]: Florence Carton, David Filliat, Jaonary Rabarisoa, and Quoc Cuong Pham "Using Semantic Information to Improve Generalization of Reinforcement Learning Policies for Autonomous Driving". In: *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV) Workshops*, 2021.

We also focused on the robustness of the different autonomous agents in case of unexpected event, i.e. an impossible command, and propose a method to evaluate this robustness. This work

is the subject of a second publication :
[Carton 2021a]: Florence Carton, David Filliat, Jaonary Rabarisoa, and Quoc Cuong Pham "Evaluating Robustness over High Level Driving Instruction for Autonomous Driving". In: accepted to Intelligent Vehicles Symposium (IV21), 2021.

1.4 Outline

The manuscript is structured as follows.

- Chapter 2 introduces the principles of supervised and reinforcement learning that are used in this thesis.
- Chapter 3 proposes a global overview of state of the art approaches related to autonomous driving and image-based navigation.
- Chapter 4 presents a first step to build a driving agent based on reinforcement learning only. The implemented methods, as well as all the necessary setup are introduced.
- Chapter 5 focuses on urban driving. In this chapter, the driving agent is adapted to handle conditional driving, i.e. to follow a high level gps instruction. We also propose a study on the robustness of autonomous agents, in particular in the case where the high level instruction is not correctly given.
- Chapter 6 studies the generalization problem, crucial for autonomous vehicles. The influence of the addition of semantic information is explored in order to improve training and generalization.
- Chapter 7 concludes these researches and discusses possible perspectives.

Chapter 2

Supervised Learning and Reinforcement Learning Background

Contents

2.1 Introduction

Although they have existed since the 1950s, neural networks have gained in popularity since 2012 with the emergence of deep learning. With the development of powerful computing machines, GPU (Graphics Processing Unit), which enable a lot of calculations to be parallelized, coupled with the creation of numerous annotated datasets (like ImageNet [Deng 2009]), deep learning has made it possible to rapidly surpass previous methods of computer vision.

Deep learning has a wide range of applications. From natural language processing to computer vision, deep learning is now omnipresent. The scope of this thesis is limited to decision making from mostly image input. Our work here focuses on algorithms that learn how to drive from road image and driving instructions.

After a quick introduction about the basis of neural network and deep learning, the aim of this chapter is to introduce the general principles of supervised and reinforcement learning. For more extensive explanations, we encourage the reader to refer to the following books: *Deep Learning* [Goodfellow 2016], *Neural Networks and Deep Learning* [Nielsen 2015], *Reinforcement Learning: An Introduction* [Sutton 2017].

2.2 Neural Network and Deep Learning

2.2.1 Artificial Neuron

The first definition of artificial neuron dates back to 1943 in *A logical calculus of the ideas immanent in nervous activity* [McCulloch 1943]. Inspired from biological neurons, a neuron is a mathematical object that follows specific rules. Considering an input $x = [x_0, ..., x_n]$, the neuron outputs $y = \sigma(\sum w_i x_i + b_i)$, where w_i and b_i are internal parameters of the neuron, respectively the weight and bias. σ is an activation function that may add non-linearity. Classical activation function are sigmoid $\sigma(x) = \frac{1}{1+e^{-x}}$, hyperbolic tangent $(tanh)$ $\sigma(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$ or rectified linear unit $(relu)$ $\sigma(x) = max(0, x)$.

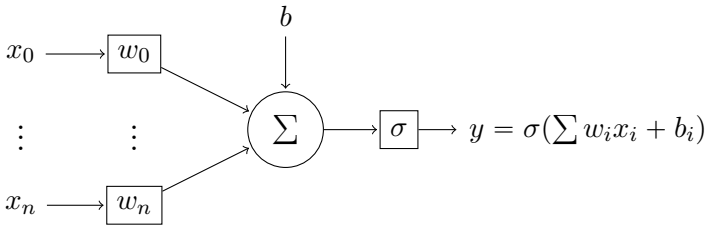


Figure 2.1: Artificial neuron

Originally, the first artificial neuron was a perceptron. It is a binary classifier, which activation function is $sign(\dot{})$, and no bias (see equation 2.1).

$$output = \begin{cases} 1 \text{ if } \sum w_i x_i > 0 \\ 0 \text{ otherwise} \end{cases} \tag{2.1}$$

2.2.2 Artificial Neural Network

Due to its construction, the perceptron is only able to classify data that can be separated by a single line. It is therefore not able to model more complex functions such as the XOR function. To model the latter, several perceptrons must be used. From this was born the Multi Layer Perceptron (MLP). Although often confused, a MLP is composed of several perceptrons - and therefore with only $sign(\dot{})$ binary activation functions, whereas an artificial neural network (ANN) contains neurons that can have all kinds of activation functions.

Figure 2.2 shows an example of a multi layer artificial neural network. Every circle represents a neuron. They are usually organized in layers, where the outputs of one layer L_i are the inputs of the neurons of the next one L_{i+1}. Internal layers are called hidden layers. We speak about deep learning when there are more than 2 hidden layers.

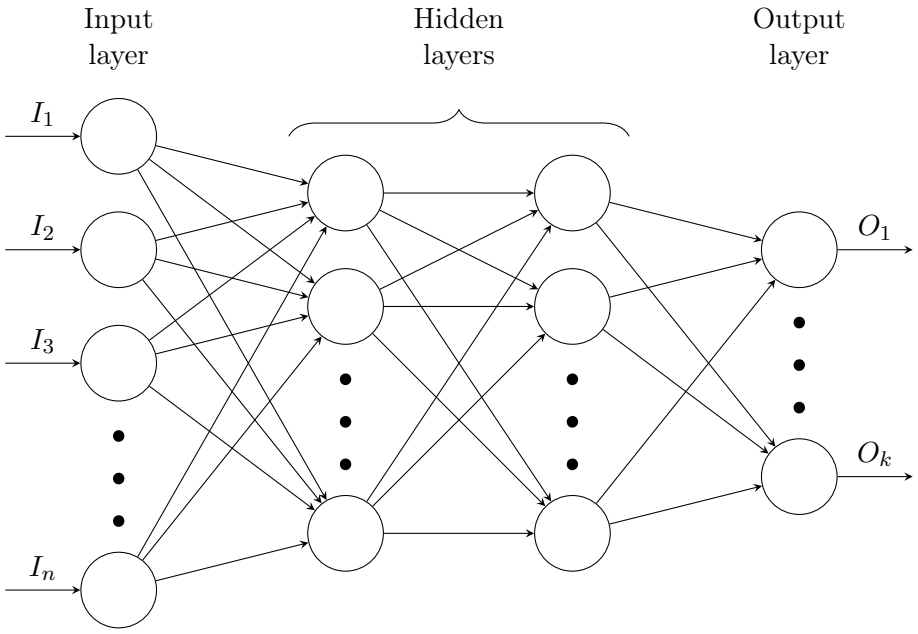


Figure 2.2: Multi layer artificial neural network

2.2.3 Convolutional Neural Network

When network layers are organized as before, they are called fully connected layers, because each neuron is connected to all neurons of the previous layer. However, this architecture does not apply well to image processing. Indeed, the size of the images would require a large number of parameters. For RGB images of size 224×224 - size of ImageNet dataset images, it would already require 150 528 ($= 224 \times 224 \times 3$) weights for each neuron of the first layer.

Convolutional Neural Networks (CNN) have emerged to allow images to be processed with fewer parameters than fully connected layers. They were introduced in 1998 by LeCun et al. in *Gradient-based learning applied to document recognition* [Lecun 1998] to allow handwriting recognition (see Figure 2.3). CNNs are composed of convolution layers usually followed by pooling layers. The first ones process the image by area (and not by pixel) to detect patterns. The pooling layers are used to sub-sample the image. In addition to reducing the number of parameters, another advantage of CNN is the translation invariance.

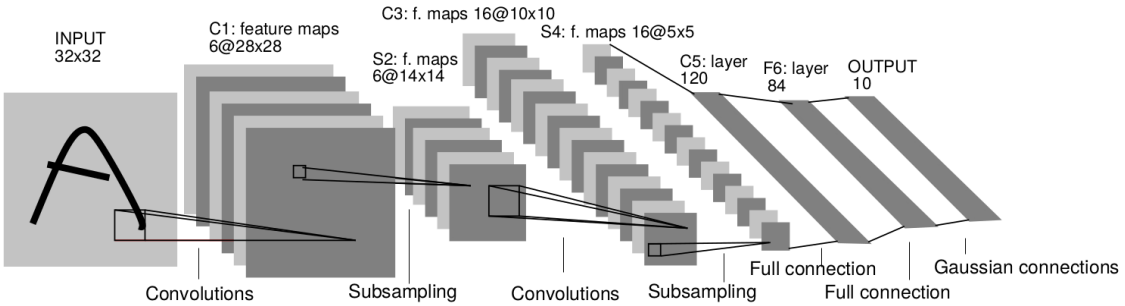


Figure 2.3: Convolutional network for handwriting recognition [Lecun 1998]

Convolutional and pooling layers are usually followed by some fully connected layers at the end of the CNN. Among the classical architectures used in computer vision, can be cited VGG

[Simonyan 2015], Resnet [He 2016] or Inception [Szegedy 2015]. The research for new architectures, lighter and/or more powerful, is also very active.

2.2.4 Training and Backpropagation

Training an artificial neural network consists in finding the optimal parameters θ (weights and bias of the neural network) for a given task. For this purpose a loss function $\mathcal{L}$ is defined - the form of which depends on the problem and the type of learning. Examples of loss in the case of supervised or reinforcement learning are given in the following sections. The parameters of the network will be modified according to the derivative of this loss function. For every $\theta_i \in \theta$,

$$\Delta\theta_i = \frac{\partial \mathcal{L}}{\partial \theta_i} \tag{2.2}$$

Parameter θ_i is updated with the gradient descent rule: (we also speak about gradient ascent when the loss is made to be maximized)

$$\theta_i \leftarrow \theta_i - \alpha\Delta\theta_i \tag{2.3}$$

with α the learning rate. The most commonly used optimizers are SGD (Stochastic Gradient Descent with momentum [Qian 1999]) and Adam optimizer [Kingma 2015].

In machine learning, we can distinguish three types of learning frameworks: supervised learning, reinforcement learning, and unsupervised learning. Supervised learning learns from annotated data, reinforcement learning uses a reward function, and unsupervised learning deals with unlabeled data. In this thesis we are only interested in the two first ones, that are presented in the next sections.

2.3 Supervised learning

The most commonly used learning type is supervised learning, where the model learns from annotated data. Classical applications in computer vision are object detection, image classification, depth prediction or semantic segmentation. The dataset is an ensemble (x_n, y_n) with $n \in \mathbb{N}$, where x_i are the inputs of our model (picture for instance), and y_i the groundtruth outputs we want the model to predict (e.g. type of object in the input image). The model can be defined by a function f with θ parameters (weights and bias of the neural network). The goal is to learn the parameters θ such as $f(x_i, \theta) = y_i$.

In supervised learning, classical loss functions is the L2 loss, or mean square error (equation 2.4)

$$\mathcal{L} = MSE(f(x, \theta), y) = \frac{1}{N}\sum^{N}(f(x_i, \theta) - y_i)^2 \tag{2.4}$$

For classification problem, cross entropy, also call log loss is often used (equation 2.5). In this case, output $f(x, \theta) = p$, p being the probability to be in the considered class, and groundtruth $y \in 0, 1$. For multi-class, the loss is the sum over the all classes i (equation 2.6).

$$\mathcal{L} = CE(p, y) = -ylog(p) + (1 - y)log(1 - p) \tag{2.5}$$

$$\mathcal{L} = -\sum_i y_i log(p_i) \tag{2.6}$$

Supervised learning presents many advantages. It is rather fast and stable, and usually offers good convergence. However, it requires annotated data, which are costly. Dataset are always

limited, and are biased by human. When it comes to performing a succession of actions (as in the case of autonomous driving), supervised learning can also suffer from distributional shift, which means that the distribution of the dataset is not identical to the one encountered during test. In this case, when the system will commit one mistake, it will pile up errors and will never know how to recover from its mistake.

2.4 RL Algorithms

Reinforcement learning offers an alternative to supervised learning. Also called learning by trial and error, it does not use annotated data, but only a reward function that indicates good or bad behavior. External bias is then reduced, but learning can take longer, since no demonstration of good behavior is given.

2.4.1 RL Definitions and Notions

2.4.1.1 Definitions

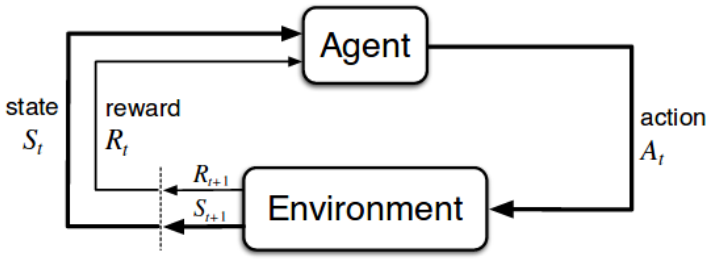


Figure 2.4: Reinforcement learning diagram [Sutton 2017]

In reinforcement learning, an agent interacts with an environment. The environment produces a state s_t at each timestep t, and when receiving the current state s_t, the agent reacts with an action a_t. The agent acts according to a policy $\pi(a_t|s_t)$, which represents the probability to take an action a_t when being in state s_t (in a deterministic environment, $\pi(s_t) = a_t$). After the agent has taken the action a_t, the environment provides a new state s_{t+1} alongside a reward r_t, which indicates how good the new state is.
The goal of the agent is to maximize the cumulative rewards: $max \sum r_t$ (the rewards are often discounted to avoid exploding sums). The sum of cumulative (often discounted) reward is called the return R_t:

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + ... = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \tag{2.7}$$

We also define V-function and Q-function as follow:

$$V_\pi(s) = \mathbb{E}_\pi[R_t|S_t = s] \tag{2.8}$$

$$Q_\pi(s, a) = \mathbb{E}_\pi[R_t|S_t = s, A_t = a] \tag{2.9}$$

The V-function, or value function is the expected return from a given state, and the Q-function, or quality function, is the expected return from a given state-action pair. These functions evaluate

then how a good a state (respectively a state-action pair) is when following a given policy π. We define as well the advantage function:

$$A^{\pi}(s,a) = Q^{\pi}(s,a) - V^{\pi}(s) \tag{2.10}$$

The advantage defines how much better action a is compared to the action chosen by the policy π.

2.4.1.2 MDP and POMDP

In reinforcement learning, the agent makes a decision according to a state it receives. The state signal is assumed to have the Markov property, i.e. to be only dependent on the previous state. Therefore the transition probability is a function only of the previous state and action: $p(s_{t+1}|s_t, a_t)$. The reinforcement learning problem is then a Markov Decision Process (MDP) defined by a 5-tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$:

- $\mathcal{S}$: set of states
- $\mathcal{A}$: set of actions
- $\mathcal{P}$: transition function $\mathcal{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0, 1]$
- $\mathcal{R}$: reward function $\mathcal{R} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \times \mathbb{R} \rightarrow [0, 1]$ or in deterministic cases $\mathcal{R} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \mathbb{R}$
- γ is the discount factor (trade-off between present and future rewards that indicates that present rewards matter more)

However the complete state with full information from the environment at timestep t is not always easy to get and in some environments, only partial information can be obtained. One can think of image occlusions in computer vision problems. The observation is then defined as incomplete information from the state. In that case the reinforcement learning problem can be defined as a Partially Observable Markov Decision Process (POMDP), which is a 7-tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma, \Omega, \mathcal{O})$ with $\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}$ and γ defined like in a MDP, Ω a set of observations and $\mathcal{O}$ a conditional observation probability function: $\mathcal{O} : \mathcal{S} \times \Omega \rightarrow [0, 1]$ (defines the probability of observing o when being in state s). To treat POMDP, several approaches are possible. Either the state can be inferred/guessed from observation, using probability distribution named belief state, or the hidden state can simply be ignored, but there can be problems as similar observations can be derived from different states.

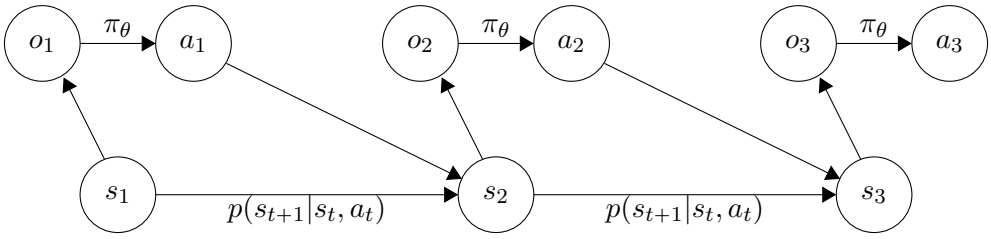


Figure 2.5: POMDP in RL [Levine 2017]

In this thesis, we are in the case of a POMDP, since our observation contains road images. Road images do not contain the whole state, since there can be occlusions for instance, and we don't have access to the complete state of the environment (e.g. position of other actors). In the following, we confuse state and observation.

2.4.1.3 Model-free vs Model-based

Reinforcement learning algorithms can be classified as model-free or model-based. In model-based algorithms, the agent tries to model the problem, i.e. to learn the MDP (as once we know the reward and transition functions, the problem can be solved by optimization). In model-free algorithms, the agent learns directly how to optimize the reward (Q-learning or Policy Gradient for instance - see section 2.4.2 for more details). In autonomous driving, the model of the environment is not known (one cannot know the future actions of surrounding actors), and it is difficult to learn a model of the environment in a large dimensional space - such as images. In this thesis we will then focus on model-free approaches.

2.4.1.4 Exploration vs Exploitation

Exploration versus Exploitation is a very famous dilemma in reinforcement learning. Exploitation on one side, aims at making the best so far, i.e. with current knowledge, and exploration consists in testing other solutions and then gather information, as an unexplored decision could be better that the other already tested. Both exploration and exploitation are crucial in RL, and the challenge lies in the trade-off between these two notions. It seems reasonable to explore a lot at the beginning of the algorithms, when all paths have not been explored yet, and explore less and less when the algorithms improves.

Two common approaches to face the E-E dilemma are ϵ-greedy and softmax. ϵ-greedy approach consists in picking the best action (= greedy action) with probability $1 - \epsilon$ and a random action with probability ϵ (ϵ from 0.01 to 0.1 are common choices). With ϵ-greedy, all actions can be chosen with the same probability. Softmax approach allows to differentiate according to their quality $Q(a, s)$. Next action is selected according to the Boltzmann distribution:

$$\pi(a, s) \sim \frac{e^{\frac{Q(a,s)}{T}}}{\sum_{a_i \in \mathcal{A}} e^{\frac{Q(a_i,s)}{T}}} \tag{2.11}$$

where a in the chosen action at time t and T is a constant called the *Temperature*.

Both ϵ-greedy and softmax work in the case of discrete actions. In the case of continuous actions, exploration is usually done by sampling random action around model output, often using Gaussian probability:

$$\pi(a, s) \sim \mathcal{N}(f(s), \sigma) \tag{2.12}$$

In equation 2.12, $f(s)$ is the model output, and σ the standard deviation of the Gaussian (that can be fixed of a learned parameter of the model).

2.4.1.5 On-Policy vs Off-Policy

RL algorithms operate with two phases: interaction with the environment to collect data, and update of the policy using collected data. On-policy algorithms always use current policy to collect data, whereas off-policy algorithms use an other policy to collect data (an older one, or a greedy-policy). On-policy algorithms are theoretically more stable - since the policy is updated with data collected with itself, but are usually slower than off-policy algorithm. Indeed off-policy algorithms are sample-efficient - i.e. one data can be reused several time - and data collection and policy update can be performed at the same time, which speeds up training.

2.4.2 Classification of Main RL Algorithms

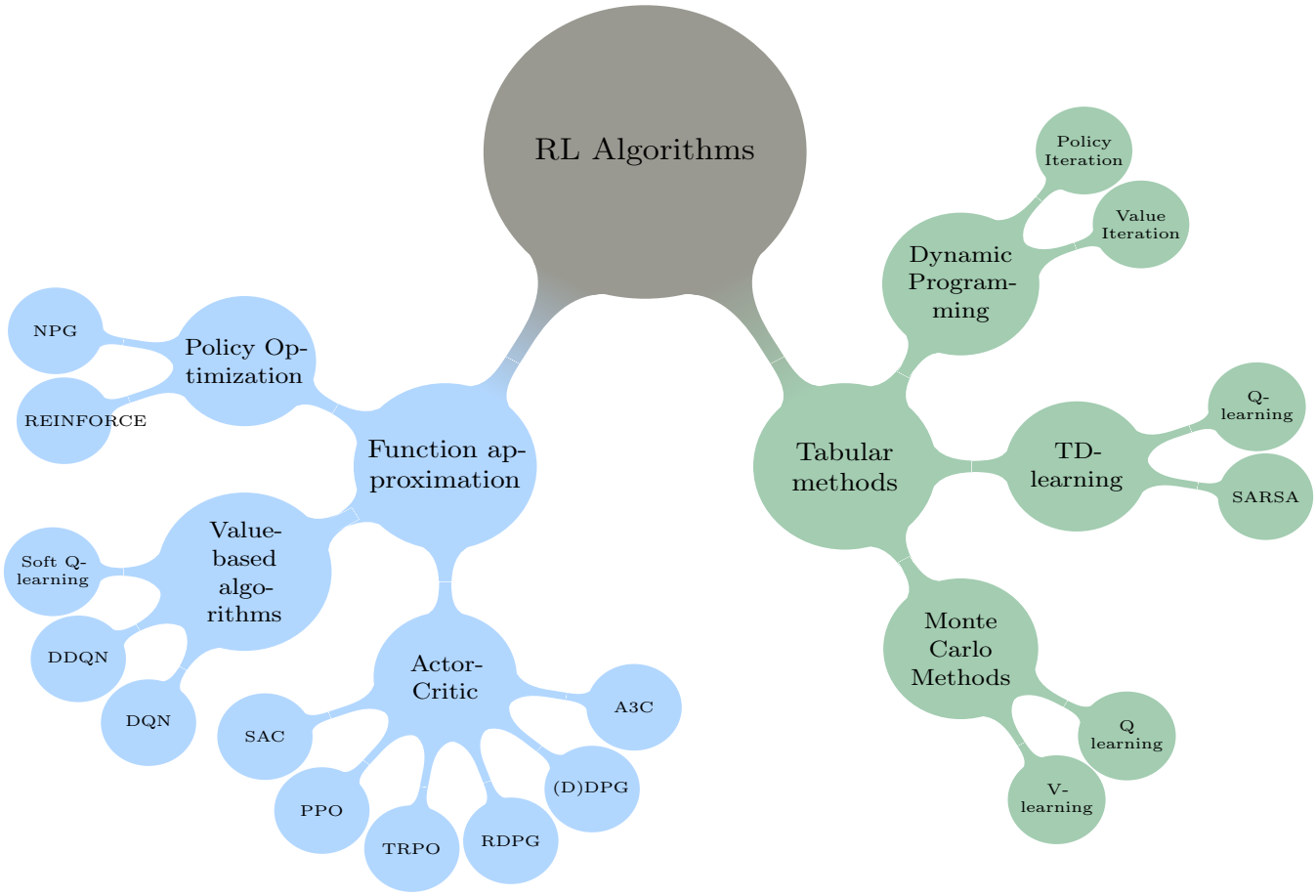


Figure 2.6: Classification of the main RL algorithms

We present here a diagram to classify the reinforcement learning algorithms. We can first separate reinforcement learning algorithm in two categories: tabular methods and function approximation. Tabular methods assume a finite MDP, i.e. with a finite number of states and actions. We will quickly present these methods in the next paragraph, and we will then go deeper into function approximation, where the MDP can be non finite.

2.4.3 Tabular Methods

This section is inspired from the book *Reinforcement Learning: An Introduction* by Richard Sutton and Andrew Barto [Sutton 2017]. The first methods presented are the tabular methods where the MDP is finite (finite MDP means that S, A and R are finite) and the dimensions of the state and action spaces are small, i.e. the value function and the Q function can be represented as tables (cf Table 2.1).

State	Value
State s_0	15
State s_1	20
...	...
State s_n	...

(a) V-table

State \ Actions	up	down	left	right
State s_0	-1	12	0	15
State s_1	3	20	4	14
...	...	...	...	...
State s_n	...	...	...	...

(b) Q-table

Table 2.1: Examples of V- and Q-tables in a gridworld with four possible actions: go up, down, left and right

The objective of tabular methods are to fill up these tables, since once they are known, the optimal policy can be inferred:

- From value function in case of known transition function:

$$\pi^*(s) = \underset{a}{argmax} \sum_{s',r} p(s', r|s, a)[r + \gamma V(s')] \tag{2.13}$$

- From Q-function:

$$\pi^*(s) = \underset{a}{argmax} Q(a, s) \tag{2.14}$$

We can divide tabular methods in 3 categories: dynamic programming, Monte-Carlo methods, and TD-learning.

2.4.3.1 Dynamic Programming

Dynamic programming is model-based RL, it assumes a knowledge of the MDP, and is used for planning and control. Dynamic programming is used to compute the optimal policy of a finite MDP with a perfect knowledge of the environment (i.e. states, actions, reward function and transition function are known). The Policy Iteration algorithm is a dynamic programming algorithm that iteratively improves the policy, until finding the optimal policy $\pi*$. As we are in a finite MDP, there exists only a finite number of policies. The Policy Iteration algorithms is in two parts. First, the current policy π is evaluated, meaning the value function V is computed for policy π. Then the policy π will be improved by using the computed value function V. When the policy can no longer be improved, it means it has reached a fixed point, i.e. the optimal policy, else the succession of policy evaluation and improvement is done again. Policy Iteration algorithm has the drawback of being computationally expensive, since both policy and value are computed at the same time. To reduce the computational expense, V-values can be directly computed. And as the transition function is known, the optimal policy can be derived from the optimal V-values: $\pi \approx \pi*$ such that $\pi(s) = argmax_a \sum_{s',r} p(s', r|s, a)[r + \gamma V(s')]$

2.4.3.2 Monte-Carlo Methods

Dynamic programming assumes some knowledge of the environment. On the contrary, Monte-Carlo Methods are model-free algorithms (= that don't know the transition function), that learn from experience. Monte Carlo methods learn from complete episodes, they don't bootstrap like TD-learning (see section 2.4.3.3). The basis idea of these algorithms is to use the mean values i.e. Q(s,a) = mean return starting from state s and performing action a (see Algorithm 1).

Algorithm 1 First-visit Monte-Carlo policy evaluation [Sutton 2017]

Initialize, for all $s \in \mathcal{S}, a \in \mathcal{A}$:
  $Q(s,a) \leftarrow$ arbitrary
  $Returns(s,a) \leftarrow$ empty list for all $s \in \mathcal{S}$
  $\pi(a|s) \leftarrow \epsilon$-greedy policy
repeat forever
  Generate an episode using π
  for each state (s,a) appearing in the episode : **do**
    $R \leftarrow$ return following the first occurrence of s
    Append R to $Returns(s,a)$
    $Q(s,a) \leftarrow average(Returns(s,a))$
  end for

2.4.3.3 Temporal Difference Learning (TD-learning)

Contrary to Monte-Carlo methods, TD-learning algorithms don't wait until the end of the episode to estimate the value, they can update values during the episode, which presents the advantage of being faster. We present two major TD algorithms: SARSA (Algorithm 2), which is on-policy, and Q-learning (Algorithm 3), off-policy, both for computing the Q-table.

Algorithm 2 SARSA [Sutton 2017]

Initialize Q(s,a) arbitrarily, $\forall s \in \mathcal{S}, a \in \mathcal{A}$, and $Q(terminal\_state, \cdot)$
repeat (for each episode)
  Initialize s
  choose action a from s using policy derived from Q (e.g. ϵ-greedy)
  repeat(for each step of the episode)
    take action a, observe reward r and next state s'
    choose action a' from s' using policy derived from Q (e.g. ϵ-greedy)
    $Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma Q(s',a') - Q(s,a)]$
    $s \leftarrow s'$; $a \leftarrow a'$
  until s is terminal

The name SARSA comes from the quintuple $(S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1})$ needed for the algorithm. At every step, the agent chose an action a according to its current policy (with sometimes a random action to explore), and gets a reward r and a new state s'. The agent will then chose another action according to its current policy, and update the Q-table with the reward r, and the Q value of new state s' and new action a'. SARSA is on-policy since the new action a' is chosen according to the current policy. Q-learning is an off-policy version of SARSA: the agent does not have to chose a second action a', the update will be performed with the best possible action $(max_a Q(s,a))$.

Algorithm 3 Q-learning [Sutton 2017]

Initialize Q(s,a) arbitrarily, $\forall s \in \mathcal{S}, a \in \mathcal{A}$, and $Q(terminal\_state, \cdot)$
repeat (for each episode)
  Initialize s
  repeat(for each step of the episode)
    choose action a from s using policy derived from Q (e.g. ϵ-greedy)
    take action a, observe reward r and next state s'
    $Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma max_a Q(s',a) - Q(s,a)]$
    $s \leftarrow s'$
  until s is terminal

As SARSA is on-policy, its convergence properties depend on the policy used, whereas it is not the case for Q-learning, which is independent of the policy being followed.

2.4.4 Function Approximation

All three classes of algorithms shown previously assumed a finite MDP, which can be limiting for continuous problems, or even for problems with high dimensions. In this section we will go deeper in the second class of reinforcement learning algorithms, called function approximation. This time, the algorithms will not necessarily converge to the optimal solution as is the case in tabular methods, since we are now in infinite dimension. The term *'curse of dimensionality'* was invented by Richard Bellman in 1961 to describe the difficulties that arise when increasing the size of data. Function approximation algorithms can be divided in three categories. The first one is value-based algorithms, that rely in inferring Q for every state-action pair, and then derive the optimal policy from it. In function approximation the state space can now be very big, but to use value-based algorithms, the action space still needs to be finite. The second category, policy-based algorithms, allows to have big or infinite action space by directly computing the policy, with no intermediary. But instead of policy-based algorithms, a third category is rather used because it is more stable. These are the actor-critics, which calculate both policy and state value at the same time.

2.4.4.1 Value-based Algorithms

Value-based algorithm are based on the calculation of the value of every state, and more precisely on the Q-value for every pair (state, action). Like Q-learning and SARSA in previous paragraph, they are based on the fact that knowing the Q function is enough to get the optimal policy $\pi^*(s) = \underset{a}{argmax}\ Q(a, s)$. Deep Q learning is the most famous value-based algorithm. It computes Q in the case of a non finite state space. We present Deep Q learning in this section, as well as some of its variants.

2.4.4.1.1 Deep Q-learning

Deep Q-learning was introduced by Mnih et at. in *Playing Atari with Deep Reinforcement Learning* [Mnih 2013]. They implemented Q-learning with image inputs. In this case, the dimensions (especially the state space dimensions) are too big for tabular methods.

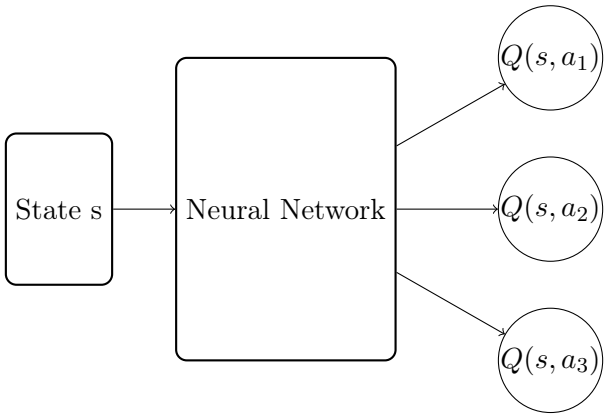


Figure 2.7: Deep Q learning

The Q function was represented by a convolutional neural network (a deep neural network, hence the name Deep Q Learning), of which inputs are preprocessed images (current state + previous states), and outputs are the predicted Q-values for every possible action. The parameter

of Q is θ (the weights of the neural network) The loss function to optimize in DQN is :

$$Loss = \mathbb{E}\left[(y_i^{DQN} - Q(s, a, \theta))^2 \right] \tag{2.15}$$

with

$$y_i^{DQN} = r_i + \gamma \, max_{a'} \, Q(s_{i+1}, a' | \theta) \tag{2.16}$$

However, many learning algorithms assume the data to be iid (independent and identically distributed), whereas it is not the case in reinforcement learning. As we deal with sequential states, successive states are strongly correlated. To overcome the problem of correlated data, Mnih et al. use the experience replay introduced by Lin in his thesis in 1993, and later reused, formalized and tested by many like for instance Adam et al. in *Experience replay for real-time reinforcement learning control* [Adam 2012]. During training, samples are stored in a buffer, and the updates of the model (i.e. a neural network) are performed with minibatches randomly sampled from the buffer. Detailed in algorithm 4, Deep Q-learning achieved state of the art results on seven Atari games.

Algorithm 4 Deep Q-learning with Experience Replay [Mnih 2013]

Initialize replay memory $\mathcal{D}$ to capacity N
Initialize action-value function Q with random weights
for episode $= 1, M$ **do**
  Initialize sequence $s_1 = x_1$ and preprocessed sequences $\phi_1 = \phi(s_1)$
  for $t = 1, T$ **do**
    With probability ϵ select a random action a_t
    otherwise select $a_t = max_a Q^*(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
    $y_i = \begin{cases} r_j & \text{for terminal } \theta_{j+1} \\ r_j + \gamma max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non terminal } \theta_{j+1} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))$ according to equation 2.15
  end for
end for

2.4.4.1.2 Improvements and Variants

Later in 2015, the authors Mnih et al. got further in Deep Q-learning and implemented successfully Deep Q-learning on the 49 available Atari games in *Human-level control through deep reinforcement learning* [Mnih 2015]. The improvement compared to the previous version of Deep Q learning is the addition of a target network used for updates, which improves the stability of the algorithms. Introduced by Van Hasselt et al. in *Double Q learning* [Hasselt 2010] for Q learning, and later in *Deep Reinforcement Learning with Double Q-Learning* [Hasselt 2015] for Deep Q learning, double Q learning is the addition of the target network. The problem of classic DQN lies in equation 2.15 as y_i (in Equation 2.16) is also a function of Q, which can lead to unstable behavior, and overestimation of the values due to the maximum in Equation 2.16. Therefore a target network Q with parameter θ' is used for target y_i computation, so that the network Q used to compute y_i don't change at every iteration. Equation 2.16 becomes:

$$y_i^{DDQN} = r_i + \gamma \, max_{a'} \, Q(s_{i+1}, a' | \theta') \tag{2.17}$$

Every N step, the parameters of the target network are updated with $\theta' \leftarrow \theta$. In [Hasselt 2015], Van Hasselt et al. compare the value estimations and scores on six Atari games with DQN and DDQN ((Double) Deep Q Network), and DQN shows worse score and stability than DDQN, as

well as a much larger overshooting of the Q value estimation, Double Deep Q learning is thus a more efficient and stable version of DQN.

Others have also improved the experience replay with *Prioritized Experience Replay* [Schaul 2016]. In this paper, the transitions are given probabilities of being selected, so the more interesting ones are sampled from replay buffer more frequently. The importance of a sample depends on its TD error $\delta = r_{t+1} + \gamma max_{a'} Q_{target}(s_{t+1}, a') - Q(s_t, a_t)$, so that they can improve faster the Q network. Andrychowicz et al. proposed in *Hindsight Experience Replay* [Andrychowicz 2017] another way - called HER - to deal with sparse rewards without tuning the reward function. The idea of HER is to replay unsuccessful episodes while changing the initial goal by the state the agent actually reached. It is specially meant to deal with sparse rewards, in environments where exploration can not always be enough to reach the goal. The goal is then stored and becomes a variable of the transition. Tests are made with the Mujoco simulator [Todorov 2012] with three tasks: pushing a box, sliding a puck, or pick and place a box, with sparse rewards (the agent only gets a reward when the task is successful). The authors show that the combination of DDPG + HER achieve to do the task in a high percentage of the cases, whereas DDPG alone was not able to even learn in some of the tasks.

One of the main problem in reinforcement learning is the exploration. Classical exploration is ϵ - greedy, but if the goal can only be achieved after a specific succession of actions (like in the mountain car environment, where the car has to gain momentum in order to go up a hill), performing one random action once in a while may not be enough. In *Deep Exploration via Bootstrapped DQN* [Osband 2016], Osband et al. propose a version of DQN where Q functions are sampled over a distribution at the beginning of an episode. Therefore the agent can explore and be consistent over one episode, which is called deep exploration.

Others test new architecture: in *Dueling Network Architectures for Deep Reinforcement Learning* [Wang, Schaul, 2016], Wang et al. propose a new architecture with two streams: on for state value (V function) estimation, and the other for Advantage estimation. Both streams are combined to produce the Q values (see figure 2.8), since $Q(s, a) = V(s) + A(s, a)$

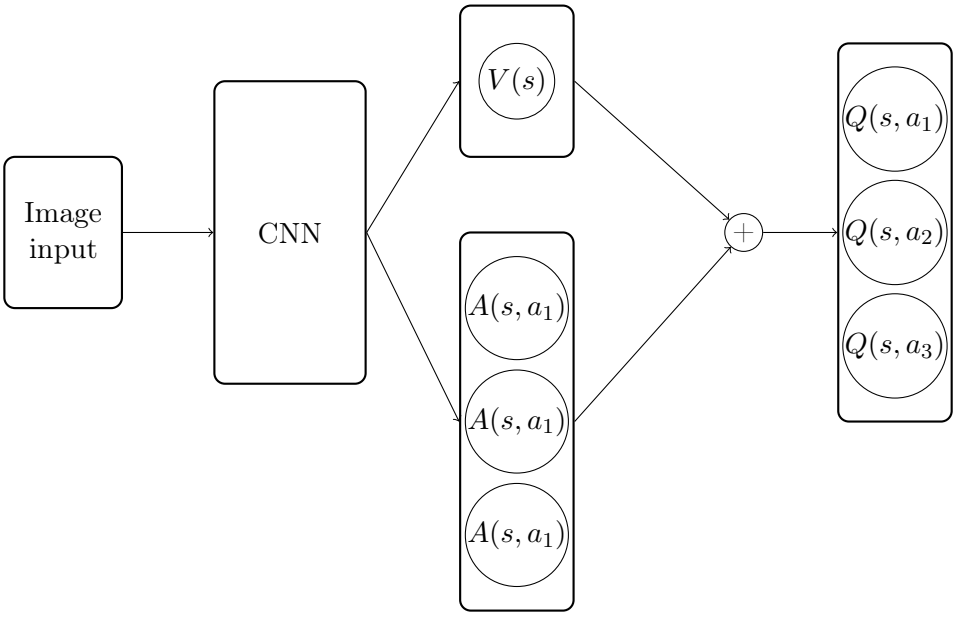


Figure 2.8: Dueling architecture [Wang, Schaul, 2016]

The key idea behind this architecture is that in many environments, actions only matters sometimes (for instance, in the Breakout game, whatever the paddle goes right and left only matters when the ball is near the paddle). By explicitly computing both the value and the advantage instead of directly computing Q, the network will learn in which state the action actually matters. In particular, the advantage estimates whether an action is better than average, and thus detects whether the action has little importance in the considered state. Wang et al. compare their dueling architecture with single stream architecture, and the result is that the dueling Q network converge faster and leads to better performance than the single stream one. Particularly on the 57 Atari games, the mean score over human performance of the dueling architecture is 591.9% whereas it is 341.2% for the DQN from [Mnih 2015].

More recently, a team of Deep Mind have combined several improvements of DQN to measure the complementarity of these extensions like double Q learning [Hasselt 2010] or experience replay [Schaul 2016] in *Rainbow: Combining Improvements in Deep Reinforcement Learning* [Hessel 2018]

2.4.4.2 Policy Gradient

Deep Q learning is an effective algorithm, but can only be applied in discrete action spaces. In the case of continuous actions, it is obviously possible to discretize them, this however quickly faces the curse of dimensionality: the number of possible actions grows exponentially with the number of dimension of the action. For instance, if the action has 3 dimensions, and every dimension is discretized in 10 intervals, the Q function output dimension is $10^3 = 1000$, and the more precision needed, the more intervals are necessary. To bypass this dimensionality curse, Policy Gradient is a branch of RL algorithms that directly optimize the policy. We don't cover here neither genetic algorithms nor evolutionary algorithms, only policy gradient that learns the policy $\pi(action|state)$. In contrary to Q-learning or Value iteration algorithms that first estimate how good a state is, and afterwards construct a policy that leads to good states, the policy is directly computed. Policy gradient presents different advantages: first of all, it is effective in high dimensional or continuous action space, it has theoretical good convergence properties, and last but not least, it can deal with stochastic policies, like in poker game, or Rock-Paper-Scissor.
In Policy Gradient methods, the policy is directly parametrized, with a parameter vector θ (classically the weights and bias of a neural network) and takes a state s as an input. In the case of discrete action space, the output of the policy network will a vector of probabilities over all the possible actions, and in the case of continuous action space, classical output of the policy network are mean (and potentially variance) of a Gaussian.

The REINFORCE algorithm is the first policy gradient algorithm detailed in *Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning* [Williams 2012]. However, since its development in 2012, there have been many improvement to make this algorithm more efficient. This section is inspired by the lectures of John Schulman [Schulman 2017] and David Silver [Silver 2015] that we recommend to the reader. The basic idea of REINFORCE (see Algorithm 5) is to maximize the expected sum of rewards over a trajectory τ: $\mathbb{E}_\pi[R(\tau)]$ by using gradient ascent over the policy parameter θ (gradient ascent as we want to maximize the reward, and not minimize a loss):

$$\theta = \theta + \nabla_\theta \mathbb{E}_\pi[R(\tau)] \tag{2.18}$$

Later we will use the notation $J(\theta) = \mathbb{E}_\pi[R(\tau)]$, so the previous equation becomes:

$$\theta = \theta + \nabla_\theta J(\theta) \tag{2.19}$$

The gradient $\nabla_\theta J(\theta)$ can be derived as follows.

$$\nabla_\theta J(\theta) = \nabla_\theta \mathbb{E}_\pi[R(\tau)] \tag{2.20}$$
$$= \nabla_\theta \int p(\tau|\theta) R(\tau) d\tau \tag{2.21}$$
$$= \int \nabla_\theta p(\tau|\theta) R(\tau) d\tau \tag{2.22}$$
$$= \int p(\tau|\theta) \frac{\nabla_\theta p(\tau|\theta)}{p(\tau|\theta)} R(\tau) d\tau \tag{2.23}$$
$$= \int p(\tau|\theta) \nabla_\theta log p(\tau|\theta) R(\tau) d\tau \tag{2.24}$$
$$= \mathbb{E}_\pi[\nabla_\theta log p(\tau|\theta) R(\tau)] \tag{2.25}$$

Using the fact that the gradient of the log probability of $p(\tau|\theta)$ is:

$$p(\tau|\theta) = p(s_0) \prod_{t=0}^{T-1} [\pi(a_t|s_t, \theta) P(s_{t+1}, r_t|s_t, a_t)] \tag{2.26}$$

where $P(s_{t+1}, r_t|s_t, a_t)$ is the transition probability and $p(s_0)$ is the initial state s_0 probability. The gradient of the log probability of p() is:

$$logp(\tau|\theta) = logp(s_0) + \sum_{t=0}^{T-1} [log\pi(a_t|s_t, \theta) + logP(s_{t+1}, r_t|s_t, a_t)] \tag{2.27}$$

When we differentiate by θ, as $logp(s_0)$ and $logP(s_{t+1}, r_t|s_t, a_t)$ do not depend on θ, we obtain:

$$\nabla_\theta logp(\tau|\theta) = \nabla_\theta \sum_{t=0}^{T-1} log\pi(a_t|s_t, \theta) \tag{2.28}$$

So:

$$\nabla_\theta J(\theta) = \nabla_\theta \mathbb{E}_\pi[R(\tau)] \tag{2.29}$$
$$= \mathbb{E}_\pi \left[R(\tau) \nabla_\theta log p(\tau|\theta) \right] \tag{2.30}$$
$$= \mathbb{E}_\pi \left[R(\tau) \nabla_\theta \sum_{t=0}^{T} log\pi(a_t|s_t, \theta) \right] \tag{2.31}$$
$$= \mathbb{E}_\pi \left[\left(\sum_{t=0}^{T} r(s_t, a_t) \right) \left(\nabla_\theta \sum_{t=0}^{T} log\pi(a_t|s_t, \theta) \right) \right] \tag{2.32}$$

To compute an estimate of the expected value, N trajectories are sampled, and the expected value is the mean over these trajectories.

Algorithm 5 REINFORCE

1: Initialize θ
2: **while** not done **do**
3: sample $\{\tau^i\}$ from $\pi_\theta(a_t|s_t)$ (run the policy to sample trajectories)
4: $\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_i^N \left[\left(\sum_{t=0}^{T} r(s_t^i, a_t^i) \right) \left(\nabla_\theta \sum_{t=0}^{T} log\pi(a_t^i|s_t^i, \theta) \right) \right]$
5: Update $\theta \leftarrow \theta - \alpha \nabla_\theta J(\theta)$
6: **end while**

The key principle of policy gradient is to set high probability to actions when rewards are high. The problem of the equation 2.32 is that all actions in one trajectory are treated equally, even if a part of them is bad and a part of them good. Moreover, Policy gradient has a high variance and a slow convergence.
The first trick to make it work better is causality, meaning that the policy at time t can only affect rewards at time t' if $t' > t$

$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_i^N \left(\nabla_\theta \sum_{t=0}^T log\pi(a_t^i|s_t^i,\theta)\right)\left[\left(\sum_{t=0}^T r(s_t^i,a_t^i)\right)\right] \tag{2.33}$$

$$\approx \frac{1}{N}\sum_i^N \sum_{t=0}^T \left(\nabla_\theta log\pi(a_t^i|s_t^i,\theta)\right)\left[\sum_{t'=0}^T r(s_{t'}^i,a_{t'}^i)\right] \tag{2.34}$$

$$\approx \frac{1}{N}\sum_i^N \sum_{t=0}^T \left(\nabla_\theta log\pi(a_t^i|s_t^i,\theta)\right)\left[\sum_{t'=t}^T r(s_{t'}^i,a_{t'}^i)\right] \tag{2.35}$$

With this new formula, actions are boosted proportionally to future rewards. The second trick to reduce the variance is to add a baseline as we only boost actions that are better than average. A very common baseline is the expected sum of rewards.

$$b = \frac{1}{N}\sum_{i=0}^N r(\tau^i) \tag{2.36}$$

We can add a baseline since:

$$\mathbb{E}_\pi[\nabla_\theta log\pi_\theta(\tau)b] = \int \pi_\theta(\tau)\nabla_\theta log\pi_\theta(\tau)b d\tau \tag{2.37}$$

$$= \int \nabla_\theta \pi_\theta(\tau) b d\tau \tag{2.38}$$

$$= b\nabla_\theta \int \pi_\theta(\tau) d\tau \tag{2.39}$$

$$= b\nabla_\theta 1 \tag{2.40}$$

$$= 0 \tag{2.41}$$

The gradient then becomes:

$$\nabla_\theta J(\theta) = \frac{1}{N}\sum_i^N \sum_{t=0}^T \left(\nabla_\theta log\pi(a_t^i|s_t^i,\theta)\right)\left[\sum_{t'=t}^T r(s_{t'}^i,a_{t'}^i - b)\right] \tag{2.42}$$

The policy needs now to be defined for the algorithm to be complete. If the action space is finite, then the policy can be represented as a neural network that outputs a vector of probability over the different possible actions. In the case of a continuous action space, the policy can be represented as a neural network that outputs the mean $\mu(s)$ (and possibly variance $\sigma^2(s)$, that can also be fixed) of a Gaussian distribution. The action a will be sampled from that distribution: $a \sim \mathcal{N}(\mu(s), \sigma^2(s))$. See Figure 2.9 for an illustration of trajectory sampling in case of continuous action space.

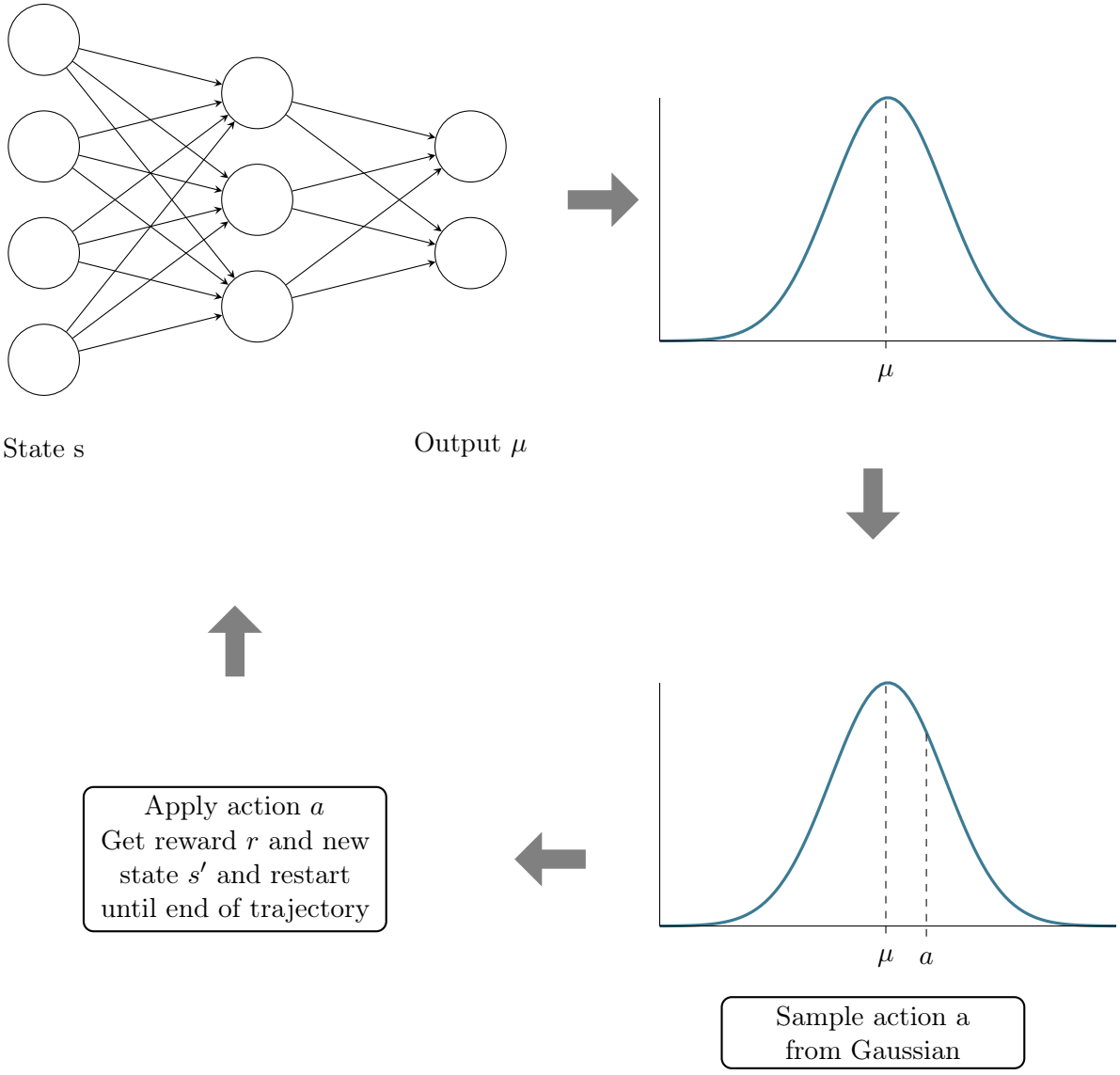


Figure 2.9: Trajectory sampling

2.4.4.3 Actor Critic

This section is inspired from the lecture of Sergey Levine [Levine 2017]. Actor Critic combines policy Gradient and Q learning. At the end of policy gradient algorithm, we had (not taking the baseline in account):

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_i^N \sum_{t=0}^T \left(\nabla_\theta log \pi(a_t^i | s_t^i, \theta)\right) \left[\sum_{t'=t}^T r(s_{t'}^i, a_{t'}^i)\right] \tag{2.43}$$

The idea of the equation above is to go on the direction where the good rewards are. However, $\sum_{t'=t}^T r(s_{t'}^i, a_{t'}^i)$ is an single-sample estimate of the future rewards starting from state s_t and taking action a_t. As this is a single sample estimate, there is a high variance in policy gradient, and reducing this variance will lead to better and/or faster convergence. It would be more effective to better estimate the future rewards starting from state s_t and taking action a_t, and a good estimator is the Q value as $Q_\pi(s, a) = \mathbb{E}_\pi[R_t | S_t = s, A_t = a]$. The equation 2.43 becomes:

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_i^N \sum_{t=0}^T \left(\nabla_\theta log\pi(a_t^i|s_t^i,\theta)\right) Q(s_t^i, a_t^i) \tag{2.44}$$

Moreover, a baseline is added to reduce bias. The sum of rewards $b = \frac{1}{N}\sum_{i=0}^N r(\tau^i)$ is a commomly used baseline. But in this case, a more precise average can be computed: $b = \frac{1}{N}\sum_i^N Q(s_t^i, a_t^i) \approx \mathbb{E}_{a_t \sim \pi_\theta(a_t|s_t)}\left[Q(s_t, a_t)\right] = V(s_t)$. The gradient then becomes:

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_i^N \sum_{t=0}^T \left(\nabla_\theta log\pi(a_t^i|s_t^i,\theta)\right) \left[Q(s_t^i, a_t^i) - V(s_t^i)\right] \tag{2.45}$$

We defined earlier the advantage $A(s,a) = Q(s,a) - V(s)$ which indicates how much action a is better than average action.

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_i^N \sum_{t=0}^T \left(\nabla_\theta log\pi(a_t^i|s_t^i,\theta)\right) \left[A(s_t^i, a_t^i)\right] \tag{2.46}$$

To compute the advantage, the following approximation is used:

$$Q(s_t, a_t) \approx r(s_t, a_t) + \gamma V(s_{t+1}) \tag{2.47}$$

Which leads to:

$$A(s_t, a_t) = Q(s_t, a_t) - V(s_t) = r(s_t, a_t) + \gamma V(s_{t+1}) - V(s_t) \tag{2.48}$$

One example of Actor-Critic is described on algorithm 6.

Algorithm 6 Actor Critic

1: Initialize θ and ϕ
2: **while** not done **do**
3: Sample $\{s_i, a_i\}$ from $\pi_\theta(a|s)$ (run the policy to sample trajectories)
4: Fit $V_\phi^\pi(s)$ to sampled reward sums
5: $A^\pi(s_i, a_i) = r(s_i, a_i) + \gamma V_\phi^\pi(s_i') - V_\phi^\pi(s_i)$
6: $\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_i^N \nabla_\theta log\pi_\theta(a_i|s_i) A^\pi(s_i, a_i)$
7: Update $\theta \leftarrow \theta - \alpha\nabla_\theta J(\theta)$
8: **end while**

To summarize, the actor critic algorithms will have two components, one actor, the policy π, with parameter θ, and one critic (Q, V or A depending on the algorithm) with parameter ϕ that estimates the quality of the actor's output. In the following we will present different actor-critic algorithms.

2.4.4.3.1 Deterministic Policy Gradient: (D)DPG, RDPG, D4PG

Classical policy gradient algorithms deal with stochastic policies, however, in *Deterministic Policy Gradient Algorithms* [Silver 2014], Silver et al. build a modified actor-critic algorithm to consider deterministic policies. The deterministic policy gradient is computed more efficiently as Q is only derivated over actions. The authors present an off-policy algorithm that learns a deterministic policy using a stochastic policy for exploration.

As an extension of DPG [Silver 2014], Lillicrap et al. present in *Continuous Control with Deep reinforcement Learning* [Lillicrap 2016] the algorithm DDPG (Deep Deterministic Policy Gradient) which is a combination of DPG and DQN [Mnih 2015]. The algorithm is based on

actor-critic for deterministic policy like DPG, but uses Deep Q learning techniques to estimate the critic (Q values): experience replay and a target network. Moreover the target network is slowly updated: $\theta' \leftarrow \tau\theta + (1-\tau)\theta'$ with $\tau \ll 1$. The result of DDPG is a noteworthy speedup on most Atari games that are solved in 20 times fewer steps than DQN. Implementation of DDPG is shown on Algorithm 7. Studies [Henderson 2018; Duan 2016] has shown that DDPG was however very sensitive to hyperparameters, and less stable than batch algorithms (DDPG updates the policy at every step, whereas many actor-critic algorithms sample a batch of trajectories before updating).

Algorithm 7 DDPG from [Lillicrap 2016]

Initialize critic network $Q(s, a|\phi)$ and actor $\mu(s|\theta)$ with weights ϕ and θ
Initialize target network Q' and μ' with weights $\phi' \leftarrow \phi$, $\theta' \leftarrow \theta$
Initialize replay buffer R
for episode = 1, M **do**
    Initialize a random process $\mathcal{N}$ for action exploration
    Receive initial observation state s_1
    for t = 1, T **do**
        Select action $a_t = \mu(s_t|\theta) + \mathcal{N}_t$ according to the current policy and exploration noise
        Execute action a_t and observe reward r_t and observe new state s_{t+1}
        Store transition (s_t, a_t, r_t, s_{t+1}) in R
        Sample a random minibatch of N transitions (s_i, a_i, r_i, s_{i+1})
        Set $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta')|\phi')$
        Update critic by minimizing the loss $L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\phi))$
        Update the actor policy using the sampled policy gradient:

$$\nabla_\theta J \approx \frac{1}{N}\sum_i \nabla_a Q(s, a|\phi)|_{s=s_i, a=\mu(s_i)} \nabla_\theta \mu(s|\theta)|_{s_i}$$

        Update the target networks:

$$\phi' \leftarrow \tau\phi + (1-\tau)\phi'$$

$$\theta' \leftarrow \tau\theta + (1-\tau)\theta'$$

    end for
end for

There exists a recurrent version RDPG (Recurrent Deterministic Policy Gradient) in *Memory-based control with recurrent neural networks* [Heess 2015] that also uses experience replay and target networks. RDPG is particularly efficient in partially observed environments, where the agent needs to remember what happened in the previous states. Another extension of DDPG is D4PG: *Distributed Distributional Deterministic Policy Gradients* [Barth-Maron 2018]. The modifications are the use of multiple actors to sample trajectories, and the distributional critic updates [Bellemare 2017]. This means that the return is treated as a random variable, and the objective is to learn its distribution instead of the expected return. D4PG also includes Prioritized Experience Replay and N-step return, the latter producing the best gain in performance.

2.4.4.3.2 RL with Multiple Workers: A3C, A2C, ACER

We have previously seen the D4PG algorithm which proposes to use multiple actors to collect data. A3C algorithm proposed by Google Deep Mind in *Asynchronous Methods for Deep Reinforcement Learning* [Mnih 2016] proposes to parallelize the gradient computation to avoid the use of a replay buffer. Instead of doing one propagation with a batch of episodes, smaller back-propagation are made in parallel with smaller batches. A3C (for Asynchronous Advantage Actor Critic) is composed of the following:

- A Global agent
- Several Workers that contains local networks

The global agent contains up-to-date networks. Every worker interacts with its own instance of the environment. Each worker copies the global networks locally, runs episodes and computes the corresponding gradients. Then the global network is updated with the local gradients of the worker, and the worker starts over again, with copying the global networks, run episode, etc.... As every worker does it in parallel, the global network's update is asynchronous. The algorithm is an actor critic algorithm, so two gradients are computed, one for the policy and one for the Value function, and the global agent and the workers have each two network (or a shared network, depending on the architecture), with parameters θ and ϕ for respectively the policy and the Value. The following drawing (Figure 2.10) represents the idea of A3C algorithm, and Algorithm 8 the pseudo-code for one worker. For clarity reasons, the complete tour is only shown for the first worker.

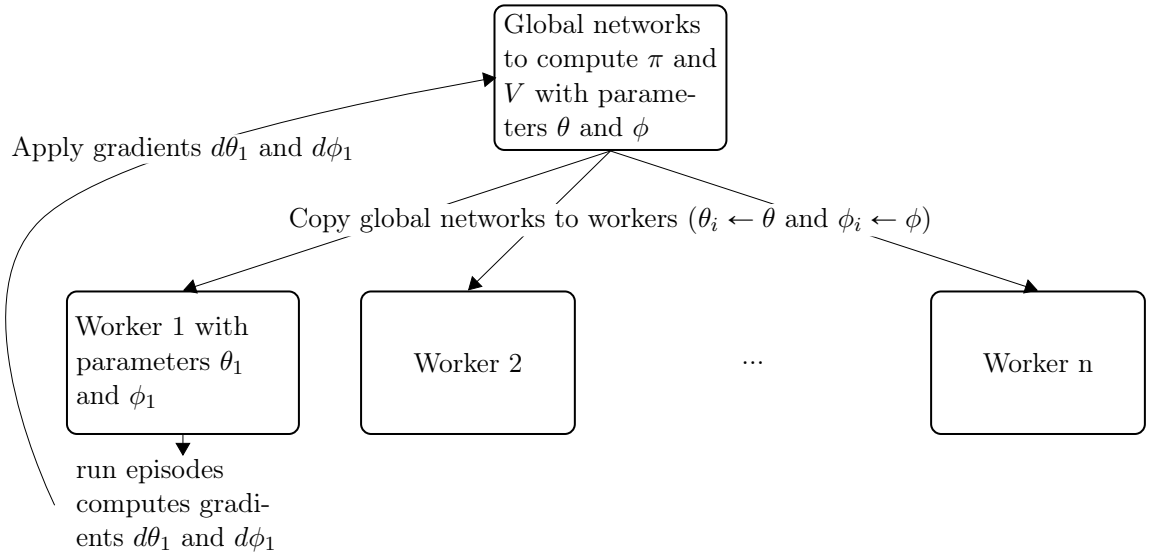


Figure 2.10: A3C algorithm

Algorithm 8 A3C - pseudo-code for one worker i (one actor-learner thread) from [Mnih 2016]

// Assume global shared parameter vector θ and ϕ and global shared counter $T = 0$
// Assume thread-specific parameter vectors θ_i and ϕ_i
Initialize thread step counter $t \leftarrow 1$
repeat
  Reset gradient: $d\theta_i \leftarrow 0$ and $d\phi_i \leftarrow 0$.
  Synchronize thread-specific parameters $\theta_i = \theta$ and $\phi_i = \phi$
  $t_{start} = t$
  Get state s_t
  repeat
    Perform a_t according to policy $\pi_{\theta_i}(a_t|s_t)$
    Receive reward r_t and new state s_{t+1}
    $t \leftarrow t + 1$
    $T \leftarrow T + 1$
  until episode done
  $R = \begin{cases} 0 & \text{for terminal state } s_t \\ V_{\phi_i}(s_t) & \text{otherwise} \end{cases}$
  for k from $t-1$ to t_{start} **do**
    $R \leftarrow r_k + \gamma R$
    Accumulate gradients wrt θ_i: $d\theta_i \leftarrow d\theta_i + \nabla_{\theta_i} log \pi_{\theta_i}(a_k|s_k)(R - V_{\phi_i}(s_k))$
    Accumulate gradients wrt ϕ_i: $d\phi_i \leftarrow d\phi_i + \partial(R - V_{\phi_i}(s_k))^2/\partial\phi_i$
  end for
  Perform asynchronous update of θ using $d\theta_i$ and of ϕ using $d\phi_i$
until $T > T_{max}$

The efficiency of A3C has been shown on Atari games as well as on labyrinth navigation task. There exist different extensions of A3C:

- A2C. The defect of asynchronism is that workers are potentially working with old versions of the policy. The synchronous version of A3C, A2C, waits until all workers are done and then update the global agent and copy up-to-date weights to the workers. A2C performs equivalent or even better than A3C.
- ACER [Wang, Bapst, 2016] (Actor-Critic with Experience Replay). It is an off-policy extension of A3C, with a replay buffer for sample efficiency.
- Impala [Espeholt 2018]. Contrary to A3C, the actors do not perform gradient computation, but send the full transition to the global agent that perform the update. Acting is completely separated from training - that can also be split into several learners.

2.4.4.3.3 Trust Region Algorithms: TRPO, ACKTR, PPO

Policy gradient algorithms allow the agent to perform actions in a continuous space, however, the step size α must be chosen carefully: if it is too small, the learning will be very slow, and if it is too big, it will be overwhelmed by the noise, as the next batch will be collected under a bad policy. In *Trust Region Policy Optimization* [Schulman, Levine, 2015], Schulman et al. propose a new algorithm to iteratively improve the policy. At every update, the policy is modified to improve performance while being close to previous policy. The true objective is to maximize $\eta(\theta) = \mathbb{E}_\pi[\ \gamma^t r(s_t)]$, but this objective is hard to compute, so the authors introduce a surrogate function

$$L(\theta) = \mathbb{E}\left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}(a_t|st_t)}} A_t\right] \tag{2.49}$$

that is a local approximation of the true objective and which first order derivatives are equals.

$$\nabla_\theta L(\pi_\theta)|_{\theta_{old}} = \mathbb{E}_{s,a\sim\pi_{old}}\left[\frac{\nabla_\theta \pi_\theta(a|s)}{\pi_{old}(a|s)} A^{\pi_{old}}(s,a)\right]|_{\theta_{old}} \tag{2.50}$$

$$= \mathbb{E}_{s,a\sim\pi_{old}}\left[\nabla_\theta log \pi_\theta(a|s) A^{\pi_{old}}(s,a)\right]|_{\theta_{old}} \tag{2.51}$$

$$\tag{2.52}$$

We get the formula of the gradient derivative as in the equation 2.46, so at first order, when θ and θ_{old} are close:

$$\nabla_\theta L(\pi_\theta)|_{\theta_{old}} = \nabla_\theta \eta(\pi_\theta)|_{\theta=\theta_{old}} \tag{2.53}$$

This surrogate function is only precise when π and π_{old} are close, so the two distribution are constrained with the Kullback-Leibler divergence. Therefore the objective of TRPO algorithm is:

$$\begin{cases} \underset{\theta}{maximize} & \sum_n^N \frac{\pi_\theta(a_n|s_n)}{\pi_{\theta_{old}(a_n|s_n)}} A_n \\ \text{subject to} & \overline{KL}[\pi_{old}, \pi] < \delta \end{cases} \tag{2.54}$$

ACKTR [Wu 2017] is a variation of TRPO, using Kronecker-factored approximate curvature instead of Kullback-Leibler divergence for for trust region optimization.

An OpenAI team proposed a *Proximal Policy Gradient Algorithm* [Schulman 2017] that modifies the TRPO algorithm for a simpler version. The authors suggest 2 different surrogate functions that constraint the new policy π to be closed to the old one π_{old}. The first one is naturally defined from Equation 2.54: instead of the separated constraint, the KL penalty is directly added in the surrogate function L^{KLPEN}.

$$L^{KLPEN}(\theta) = \mathbb{E}_t \left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)} A_t - \beta KL[\pi_{\theta_{old}}, \pi_\theta] \right] \tag{2.55}$$

The parameter β is updated as follow. With $d = \mathbb{E}_t[KL[\pi_{\theta_{old}}, \pi_\theta]]$

$$\begin{array}{ll} \text{if} & d < d_{targ}/1.5 \quad \beta \leftarrow \beta/2 \\ \text{if} & d > d_{targ} \times 1.5 \quad \beta \leftarrow \beta \times 2 \end{array} \tag{2.56}$$

However this surrogate function can be replaced by a stabler one, a clipped surrogate function L^{CLIP}. Let's define $r_t(\theta)$ as the probability ratio $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$ (and A_t is still the advantage at timestep t). The clipped surrogate function is:

$$L^{CLIP}(\theta) = \mathbb{E}_t \left[min(r_t(\theta)A_t, clip(s_t(\theta), 1-\epsilon, 1+\epsilon)A_t) \right] \tag{2.57}$$

This clipped function will force the $r_t(\theta)$ to stay in $[1-\epsilon, 1+\epsilon]$ range.

The algorithm is an Actor-Critic, so the value function V is also computed. In the event that both policy and value function share neural network parameters, the objective function is:

$$L(\theta) = \mathbb{E}_t \left[L_t^{PG}(\theta) + c_1 L_t^{VF}(\theta) - c_2 S[\pi_\theta(s_t)] \right] \tag{2.58}$$

where L^{PG} is either L^{CLIP} or L^{KLPEN}, L^{VF} is a squared error value loss, and S is an entropy bonus to encourage exploration. These algorithms are compared with each other using the Atari game benchmark, and on robotics tasks using the Mujoco simulator [Todorov 2012], where PPO algorithm performance far exceed A2C and TRPO. On Atari games, PPO performance competes with ACER, and exceeds it on the Enduro game. The PPO algorithm presents also the advantage of being simpler to TRPO and ACER, and performs at least as well. It also allows the value and policy network to share parameters, which is not the case in TRPO because of the constraint on the policy.

2.4.4.3.4 Entropy with Reinforcement Learning: SQL, SAC

A classical problem in reinforcement learning is that agents become very specialized, as they have been trained to solve a very specific task the best way possible. More specifically, the agents learn only one way to solve the task. We can see and example in figure 2.11: in classical RL, the robot will only focus on the upper passage, which is the optimal one, and will have to relearn from scratch if this passage is blocked (and might not be able to relearn at all). One way of solving this issue and to force the robot to explore and learn all possible ways to solve the task is to add entropy, and maximizing both the sum of rewards and the entropy, as presented in *Reinforcement Learning with Deep Energy-Based Policies* [Haarnoja 2017]. The optimal policy becomes:

$$\pi* = \underset{\pi}{argmax} \mathbb{E}_\pi \left[\sum_t r_t + \mathcal{H}(\pi(\cdot, s_t)) \right] \tag{2.59}$$

With this method, named Soft Q-Learning (SQL), the robot will learn both passages, even if he will prefer the most optimal one, and will be able to quickly move on to the second passage when the first one will be blocked.

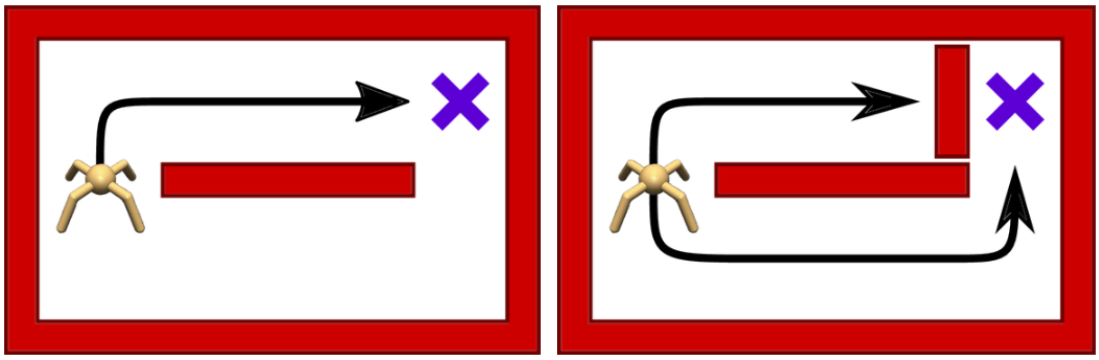


Figure 2.11: A robot navigating a maze [Tang 2017]. With classical RL (left), the robot will learn only the best way to reach the goal. With entropy regularization (right), the robot will learn both ways, so that it can adapt if the environment changes.

Soft Q-Learning is the predecessor of Soft Actor-Critic (SAC) from *Soft Actor-Critic: Off-Policy Maximum entropy deep reinforcement learning with a stochastic actor* [Haarnoja, Zhou, Abbeel, 2018]. SAC is the actor-critic version of SQL, combining off-policy updates and maximum entropy.

Later work [Haarnoja, Zhou, Hartikainen, 2018] extends SAC to add automatic tuning of hyperparameter, which lead to stabler training. A lot of algorithms (A3C, PPO...) now includes entropy in their surrogate function, for a better exploration and improved robustness [Ahmed 2019].

2.5 Conclusion

In this chapter we presented an overview of deep neural networks, supervised learning and reinforcement learning algorithms, with a focus on model-free deep reinforcement learning. On some sequential decision problems (like the game of Go), reinforcement learning has outperformed supervised learning, and RL does also not require annotated data. This thesis therefore focuses on reinforcement learning to solve vision-based navigation. To be close to human behavior, we chose to work with an algorithm that allows continuous actions. We have opted for PPO algorithm. At the time this thesis began, it was one of the state of the art algorithms for image based RL, and has notably surpassed A2C and ACER on the Atari game Enduro. PPO is on-policy, which is usually more stable than off-policy, and does not require the use of a replay buffer. Indeed, storing many images (of sometimes large size) is a disadvantage of off-policy algorithms. On-policy are however sample-inefficient, but as we will work with a simulator (see Chapter 3 for environment choice and details) - and not in real world where data collection is always more costly - data collection was not a key element in our choice. PPO is an actor-critic that also offers to share the parameters between policy and value (which TRPO does not for instance). In the next chapter, we present an overview of the application of reinforcement learning to vision-based navigation tasks.

Chapter 3

State of the art

Contents

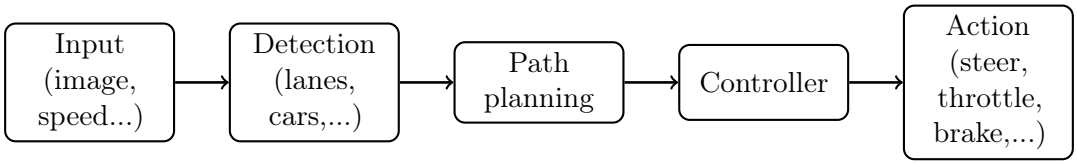
3.1 Introduction

The purpose of this chapter is to present existing work related to our problematic, namely reinforcement learning for visual-based autonomous driving. We are interested in two aspects, the perception of the environment, and the control of the driving. We have divided this state of the art in four sections. The first one covers autonomous driving, with both supervised learning and reinforcement learning. The second one is interested in related issue, that are not directly autonomous driving, but whose ideas can be applied to it. The third part focuses on the tools to work with, and in particular the driving simulators. Finally, we will synthesize the different approaches and indicate the research directions we have chosen.

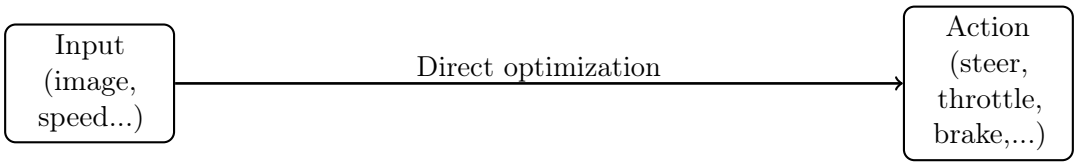
3.2 Autonomous Driving

Traditionally in autonomous driving, two approaches can be distinguished (see Figure 3.1). The first one consists of a modular pipeline, which breaks down driving into intermediate tasks, such as

perception, path planning, and control. This allows monitoring all stages of the decision-making process but needs individual optimization for every subsystem, making it difficult to find an overall optimal solution. The second approach that has recently become popular is end-to-end driving, where the system directly maps raw inputs, such as road images, to driving controls. The advantage of end-to-end learning is the reduction of intermediate operations, especially as these are mostly designed for human drivers. It can reduce time and complexity not only during training, but also in later application, as no extra features are computed. Besides, no human bias is added in the choice of intermediate representations.



(a) Traditional autonomous driving architecture



(b) End-to-end autonomous driving architecture

Figure 3.1: Traditional vs end-to-end learning

We focus in this work on the use of a single camera as input sensor, but many other elements can be used as input information: lidar, odometry, radar, etc... In this section will be presented approaches that work end-to-end (perception to control), as well as approaches that use intermediate representations. We mostly focus on approaches that uses one front-facing camera - and so RGB images as input, but we will also present some works that use birdview images.

3.2.1 Autonomous Driving with Imitation Learning

In the case of a sequential decision problem (such as autonomous driving), we often speak of *imitation learning* or *behavior cloning* to speak of supervised learning. In what follows, we will not differentiate between the three terms. Imitation learning is learning by demonstration. In the case of autonomous driving, performing imitation learning would mean to feed a learning agent with examples from human driving, and let the agent reproduce what humans do. Supervised learning therefore requires a large amount of annotated data, which can be difficult to collect in the real world. This is especially true since autonomous driving requires seeing a large number of different situations (different weather, times of day and night, different roads, etc.). The use of simulated data is a way to get around these challenges, but it poses the problem of the transition from simulation to real world.

Driving Using Neural Networks

ALVINN (Autonomous Land Vehicle In a Neural Network) [Pomerleau 1989] is the first autonomous vehicle using a neural network trained with supervised learning by Navlab in 1989. The neural network used contained only three layers. The input layer had 1217 units: 960 from a

30×32 road image, 256 from a 8×32 laser rangefinder, and one feedback unit from output layer. The hidden layer contained 29 units, and output layer 46. ALVINN is capable of performing lane following task on simple real-world road. Since ALVINN and its 3 fully-connected-layers neural network, the sizes and possibilities of neural networks have evolved a lot. In *Off-road obstacle avoidance through end-to-end learning* [LeCun 2005], a mobile robot is trained to avoid obstacles while driving off-road. The input size is much larger than ALVINN's, with two stereo cameras that produce a input of size $149 \times 58 \times 6$, and a CNN is used with 72000 trainable parameters. This approach effectively avoids obstacles in real-world navigation. More recently in *End to End Learning for Self-Driving Cars*, Bojarski et al. trained a CNN with a large dataset (around 72 hours of driving) to perform lane following and deployed it on a real car [Bojarski 2016]. The same idea is successfully applied in [Rausch 2017] in *Learning a deep neural net policy for end-to-end control of autonomous vehicles* with a driving simulator. However both [Bojarski 2016] and [Rausch 2017] only control steering angle, and not the speed of the vehicles, on lane-following task.

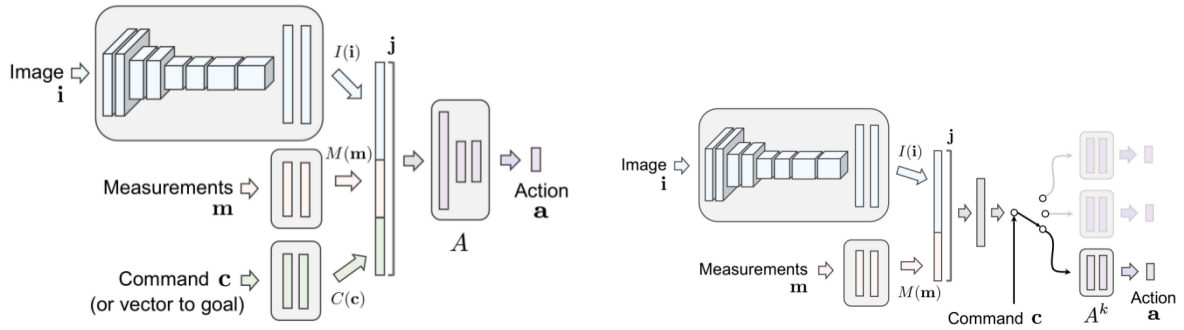
Taking Temporality into Account

Autonomous driving involves a sequential decision-making process. Adding temporality improves the robustness and consistency of a model. In both *Going deeper: Autonomous steering with neural memory networks* [Fernando 2018] and *End-to-end learning of driving models from large-scale video datasets* [Xu 2017], LSTM layers are added to handle memory. In [Xu 2017] only one layer of LSTM is used. The output of the CNN is merged with the features of the trajectory (speed and angle), and the whole is passed through a LSTM layer. In [Fernando 2018] on the other hand, there are two separate memory modules, one for the CNN output (the spatial memory module), and one for the trajectory features (the trajectory memory module). Output are merged after the memory modules. This architecture allows to have differentiated memories and to better capture long-term dependencies.

Handling More Complex Scenarios

All these approaches present end-to-end autonomous driving, but they mostly focus on lane following task, no driving command ('turn left', 'go straight') is given. When introducing the CARLA simulator in [Dosovitskiy, Ros, 2017], Dosovitskiy et al. also propose a baseline of conditional autonomous driving with three different approaches: modular pipeline, end-to-end imitation learning and end-to-end reinforcement learning. Along with the simulator, a benchmark is also proposed. The purpose of this benchmark (which we will call CARLA CoRL benchmark) is to evaluate the driving agent on goal-directed navigation. Four tasks are proposed: straight (when destination is straight on), one turn (destination is one turn away), navigation (full navigation in empty town), and dynamic navigation (full navigation with dynamic obstacles - cars and pedestrians). To evaluate the generalization capabilities of the agents, 6 weathers are used: 4 are seen during training, and 2 only at test time, as well as two different towns: one for training, and one for test. The benchmark outputs the percentage of successful episodes, i.e. when the driving agent has managed to reach destination. On this benchmark, both modular pipeline and imitation learning show satisfactory results, especially on training conditions, whereas reinforcement learning approach is way beyond. However in unseen town, performance drops, for both modular pipeline and imitation learning. More detail on imitation learning training are given in *End-to-End Driving via Conditional Imitation Learning* [Codevilla 2017], along with transfer to real world. The inputs of the supervised learning algorithm are: the current image i from the road, a vector of measurements m (containing the current speed for instance), and a

high level command c (to indicate where to go). Two global architectures are studied here (see Figure 3.2).



(a) command input architecture

(b) branched architecture

Figure 3.2: Two network architectures for conditional driving [Codevilla 2017]. Left: the driving command is used as an input and its features are concatenated with features fom image and measurements. Right: the driving command (that needs to be discrete), is used to split the network into command-specific branches.

In the first approach, the three inputs pass through independent networks: a convolutional network $I(i)$ for the image, and two fully connected networks for command and measurement ($C(c)$ and $M(m)$), the outputs of which are concatenated and passed through a control module represented as a fully connected network $A(< I(i), M(m), C(c) >)$. This architecture is named *command input* as the command is given as input. The problem of this approach is that the algorithm is not forced to take the command c in account. To ensure that, a second approach is proposed: both image and measurement networks are kept, and command is supposed to be discrete $c \in C = \{c^1, c^2, ..., c^k\}$. After $I(i)$ and $M(m)$ are concatenated, according to the command c, a branch A^k is chosen, which is a sub-policy corresponding to command c^k. This architecture is called *branched*. This branched architecture has been used a lot since [Sauer 2018; Müller 2018; Toromanoff 2019], as it has shown better results than classical concatenation.

In *Explaining autonomous driving by learning end-to-end visual attention* [Cultrera 2020], the authors add an attention module to the imitation learning architecture. This attention module allows to explain the prediction of the model, and improves the performance of the driving agent compared to simple imitation learning.

Finally in *Learning Situational Driving* [Ohn-bar 2020], training is performed in three steps. First a mixture of models is trained with imitation learning. A context embedding system is then learned: it is a VAE of which encoder is used to extract the information from the input image while reducing the dimension (see Figure 3.3). The policy is then composed of the mixture of models as well as the context embedding. The last step is policy refinement using CMA-ES, only in the last layers of the policy.

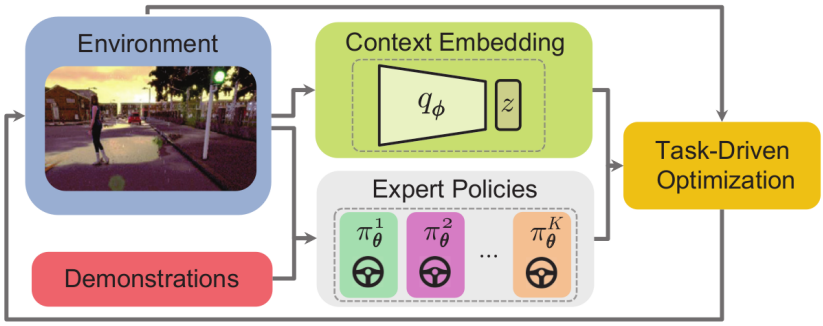


Figure 3.3: Learning Situational Driving [Ohn-bar 2020]. The policy is a combination of a mixture of experts with context embedding, refined with task-driven rewards using evolutionary algorithm.

From Simulation to Real World

First developed and trained for urban driving in simulation, the model from [Codevilla 2017] is also trained and tested in the real world, in a residential area with a small robotic truck. Data is collected directly with the truck, and the network is trained in the same way as in simulation. In the same way, the branch architecture allows the robot not to miss any crossover. However, real-world data is expensive and difficult to collect, so many approaches focus on the transition from simulation to the real world rather than training with real world images.

To facilitate the transition to real-world driving, in *Driving Policy Transfer via Modularity and Abstraction* [Müller 2018], the driving policy is not learned from end-to-end, but from segmented images and produces waypoints rather than driving command (steering angle and acceleration/braking) (see Figure 3.4).

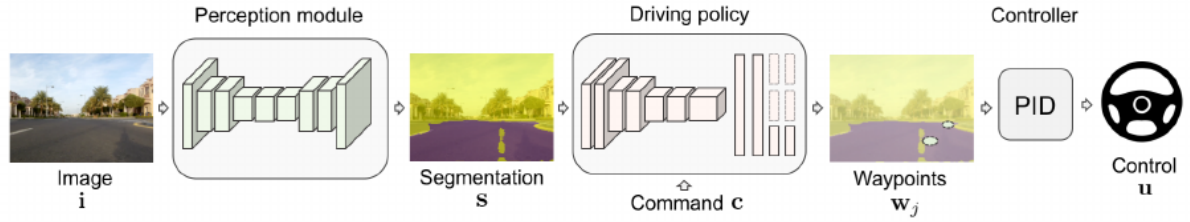


Figure 3.4: Driving system network architecture [Müller 2018]. Three modules are used: the perception module that segments the input image, the driving policy that computes waypoints from segmented image, and finally the PID controller that computes driving commands.

The perception module is trained on Cityscapes dataset [Cordts 2016]. It is composed of and encoder-decoder network with ERFNet architecture [Romera 2018], and then fixed to train the driving policy. The driving policy is trained on CARLA simulator [Codevilla 2017]. It takes one segmented map as input, and output the two next waypoints at every frame, according to the driving instruction (turn left, go straight, etc...). The architecture is the same as in [Dosovitskiy, Ros, 2017], with branched network. The waypoints are then converted to driving control through PID. Extensive ablation study is performed, the model is compared with several variations: no perception module (image is used to compute driving policy instead of segmented image), direct mapping with driving command (instead of waypoints), with and without data augmentation or domain randomization, etc... They show that the proposed approach outperformed all the others in terms of generalization in both simulation and real world.

In *Learning to Drive from Simulation without Real World Labels* [Bewley 2019], a translation

network [Liu 2017] is used to bridge the gap between simulation and real world. The driving agent is trained with imitation learning only in simulation, and the translation network transforms real world images to the latent space that is common with simulated images (see Figure 3.5). The agent is then capable of driving in real town without having seen any real driving examples.

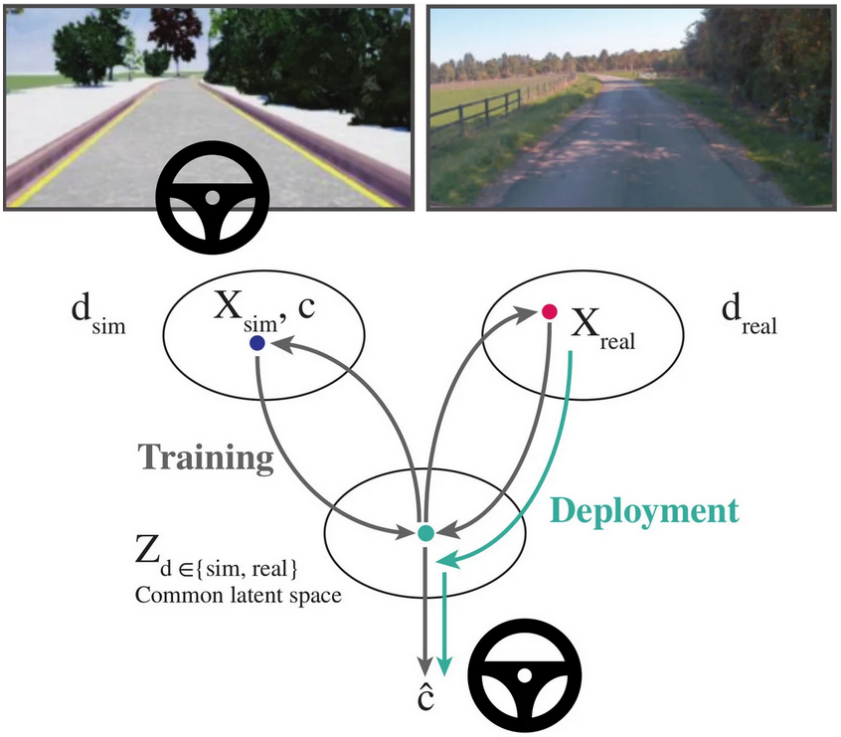


Figure 3.5: Sim2Real transition network for autonomous driving [Bewley 2019]. Z is the common latent space between real and simulation images, and is used to compute the driving command. Two Variational AutoEncoders are used to compute this latent space, and to generate images in any of the two domains.

Using Intermediate Representations

As introduced earlier (see Figure 3.1), two approaches traditionally prevail in autonomous driving: modular pipeline, and end-to-end driving. While end-to-end training can be seen as a black box, difficult to interpret, and difficult to train, the modular pipeline requires independent optimization of each module. The idea of computing intermediate representations for autonomous driving can be seen as a compromise between end-to-end learning and modular pipeline.

In *DeepDriving: Learning Affordance for Direct Perception in Autonomous Driving* [C. Chen 2015], the authors Chen et al. have decided to use this intermediate approach, which they call *Direct Perception*. A CNN processes the input image, and outputs meaningful indicator - affordances, like distances to other cars and to lane marks (see Figure 3.6). A simple controller is then used to make driving decisions. To do so, a convolutional network is trained with 12 hours of human driving from the racing video game TORCS [Wymann 2014]. The controller uses the affordance computed by the CNN to compute desired steering command and acceleration/brake based on the distance to the vehicle in front, its placement in its lane, and the command requested (stay in its lane or change lines). Tests are performed on unseen TORCS tracks. Real world tests are also performed, with both smartphone videos and KITTI dataset [Geiger 2013] to evaluate

the lane detection and car distance estimation.

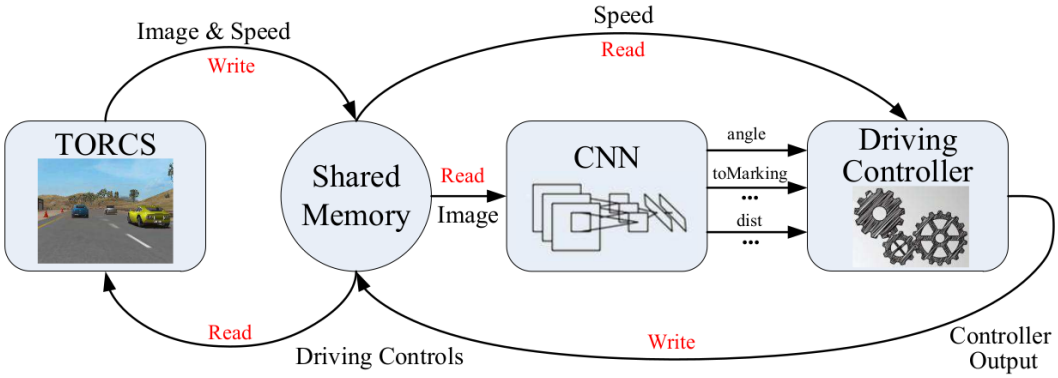


Figure 3.6: DeepDriving architecture [C. Chen 2015]. Input images are used to compute affordance - e.g., distance to lane marks.

This work has been improved by Al-Qizwini et al. in *Deep learning algorithm for autonomous driving using GoogLeNet* [Al-Qizwini 2017]. Only 5 affordances are computed (against 13 in [C. Chen 2015]) - however distance to preceding car is assumed to be given by a distance-measuring device and not computed from image input. A comparison between state-of-the-art CNNs (VGG [Simonyan 2015], GoogleNet [Szegedy 2015] and Clarifai [Zeiler 2014]) is carried out. This approach outperforms the previous one by using a more accurate feature extractor (GoogleNet [Szegedy 2015] is showed to be the best), and to remove redundant affordances.

Using intermediate representation has been applied to conditional driving in the simulator CARLA in *Conditional Affordance Learning for Driving in Urban Environments* [Sauer 2018]. The main difference with previous approaches is the taking into account of conditional driving, and use of video input instead of frame-by-frame (see Figure 3.7). The effectiveness of this approach is demonstrated on the CARLA CoRL benchmark. It also takes into account the management of traffic lights and the detection of traffic signs.

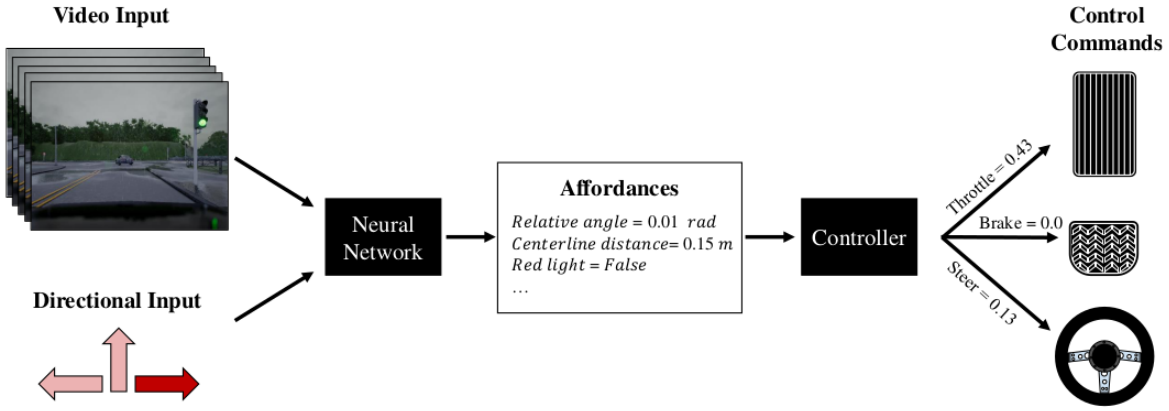


Figure 3.7: Conditional affordance learning (CAL) architecture [Sauer 2018]

These affordances can also be trained as auxiliary task, like in [Mehta 2018; Hawke 2019]. In *Learning End-to-end Autonomous Driving using Guided Auxiliary Supervision* [Mehta 2018] two types of auxiliary predictions are computed: visual affordance (e.g. distance to vehicle ahead, lateral distance to pedestrian approaching, etc...) and action primitive (slow down, turn right,

speed up, ...). From high level direction given by the planner input, and considering visual and speed inputs, the action primitive serves as a planner to predict the action to be carried out at any given moment. In other words, if the instruction (planner input) is to turn right, the action primitive predicts if the agent needs to turn right immediately, or wait until it reaches the crossing, or slow down first to stop at a traffic light, etc... As shown in Figure 3.8, the auxiliary predictions are trained at the same time as the driving task, and they are also used as input for the last block of the architecture.

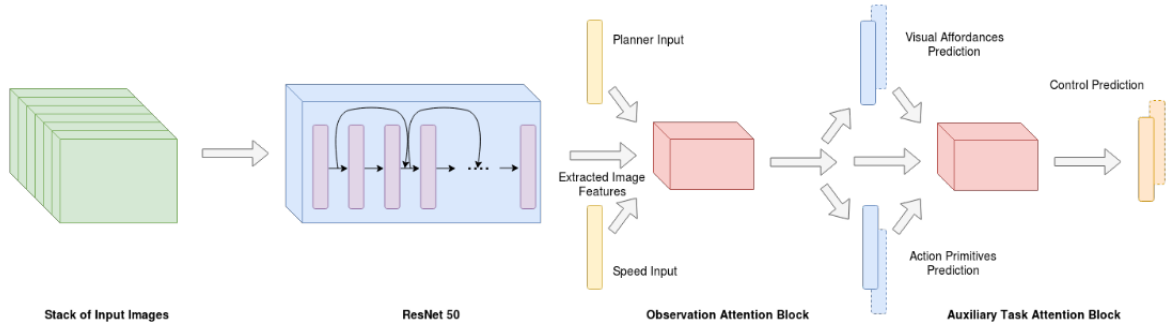


Figure 3.8: Guided auxiliary supervision architecture [Mehta 2018]

In this case the loss is a combination of auxiliary loss (one for action primitives and one for visual affordance) and of driving control loss. On this work, affordance are used also as input for the next step, so we known what is in the intermediate representation. In *Urban Driving with Conditional Imitation Learning* [Hawke 2019], Hawke et al. also use auxiliary tasks, but the intermediate representation is not directly specified (see Figure 3.9). The perception module is trained to reconstruct different interpretable outputs: semantic segmentation and depth from one RGB image, and optical flow from two consecutive frames to add temporality. However only encoded features are used for driving task. The perception module is pretrained separately on several large datasets (KITTI [Geiger 2013], Cityscape [Cordts 2016], Deeplab [L. C. Chen 2018],...). In some of the experiments, three cameras are used (a left-facing and a right-facing camera in addition to the forward-facing camera), and each image is processed independently by the perception module. Intermediate representations are merged after with sensor fusion module.

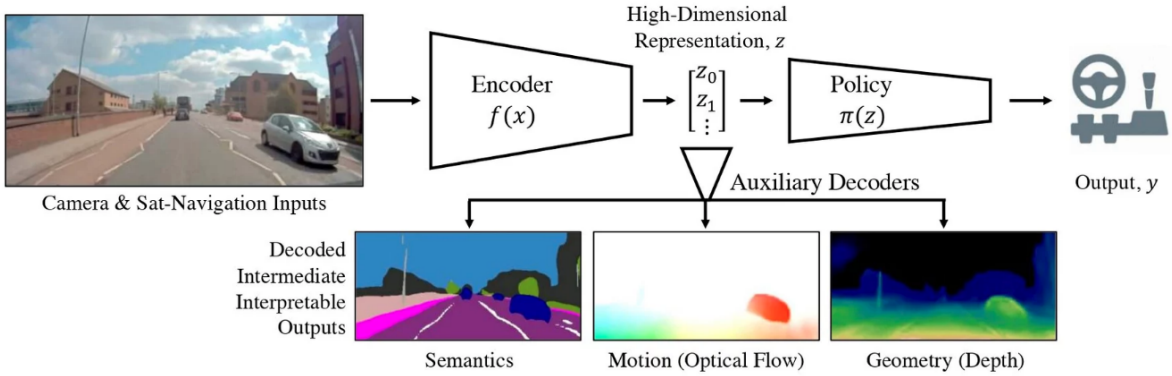


Figure 3.9: Network architecture for real world urban driving with three auxiliary tasks: semantic segmentation, depth prediction and optical flow [*wayve blog* 2019]

This work is completely realized in real world. Data are collected by driving around Cambridge, and tests are performed with a human driver ready to take control in case of a problem. In this work the authors observe different key elements for autonomous driving through supervised learning. First of all the distribution of data is very important. In an important part of the data, the vehicle is at a standstill, and when driving, it usually goes straight ahead. To prevent these predominant practices from taking over, the dataset is balanced during the learning process. They have also found that the more data that is learned, the better the system performs. Finally, they emphasize that a robust representation of the environment is an essential element for real-world driving.

The same auxiliary predictions, namely semantic segmentation and depth prediction, are used *Rethinking Self-Driving: Multi-Task Knowledge for Better Generalization and Accident Explanation Ability* [Z. Li 2018] (see Figure 3.10). However, contrary to [Hawke 2019], the perception module is trained separately and frozen during driving module training.

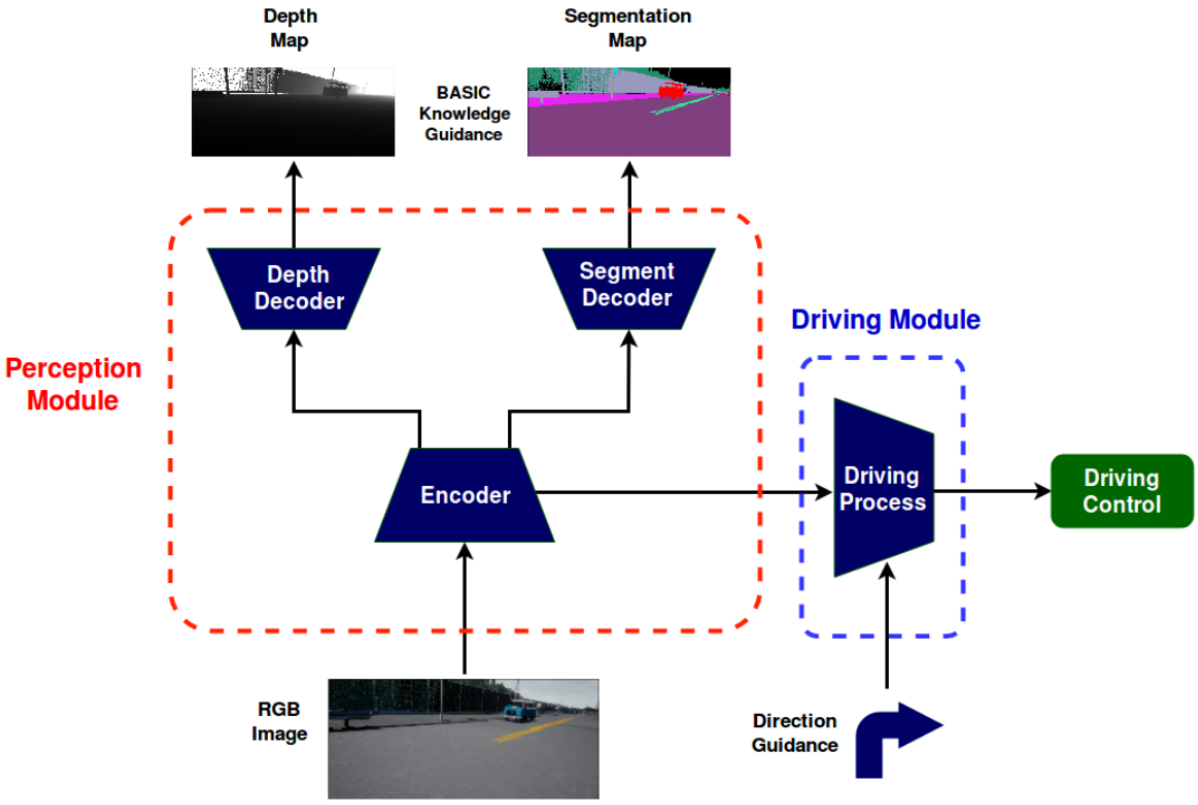


Figure 3.10: Multi task training for self driving (MT) [Z. Li 2018]

Learning to drive can be divided into two principles: understanding the environment, and learning to act on it. Using intermediate representations as presented above is one method to help the agent understand the environment. In *Learning by cheating* [D. Chen 2020], a privileged agent is trained with expert trajectories, and has access to the layout of the environment (birdview, other actors positions, etc...). This privileged agent then trains an agent that only has access to raw image input and no other groundtruth information. This decomposition allows learning in 2 steps: the privileged agent has a complete knowledge of the environment, and can therefore focus on learning to drive. The second agent will therefore focus more on understanding the image provided to him as input (called 'learning to see'). This approach is the first one to reach

a perfect score on CARLA CoRL benchmark, and shown effective generalization in the unseen town.

3.2.2 Limits of Imitation Learning

However, even with data augmentation, the problem of imitation learning, and more generally in supervised learning, is that we use human provided data. How can a car drive perfectly if we only trained it with imperfect human driving examples ? We can therefore see why it is interesting to use reinforcement learning: human bias would be avoided since the system will learn from itself. It is not told how to behave, but experiences by itself and gets positive and/or negative rewards according to how good his actions were. The point is to give the agent no human-designated features, to avoid human bias. In the case of an autonomous car, the agent is no taught to follow the lines of the road, but will be given a negative reward if it goes off the road.

In *Exploring the Limitations of Behavior Cloning for Autonomous Driving* [Codevilla 2019], the authors investigate and analyze the limits of imitation learning applied to autonomous driving. The limitations of behavior cloning that are highlighted in this work are:

- Distributional shift (also covariate or dataset shift): difference between training and test distribution. In imitation learning for autonomous driving, it is mainly due to the sequential nature of the driving task, violating the assumption that data are i.i.d. (independent and identically distributed). During test phase, unseen situations will lead to small mistakes that will pile up and cause bigger mistakes. (see Figure 3.11).

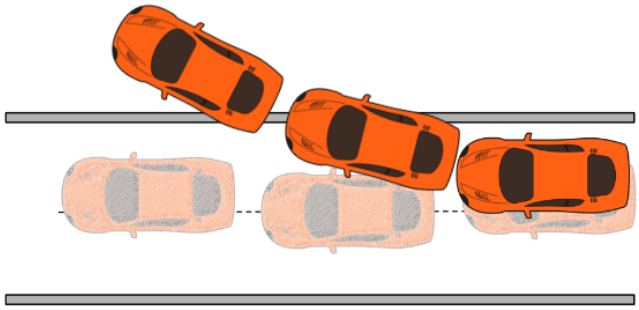


Figure 3.11: Covariate shift example between training (pale images), and test (normal images) [Shkurti 2021]

- Dataset bias: in the case of autonomous driving, collected data are mostly easy situations (like going straight). However the agent needs to learn to react in more complex situations, that are usually less present in the dataset, because they occur less often.
- Causal confusion: when the network find correlations that are only artificial in dataset.
- High variance between the results of models trained with the same parameters. Under the most difficult test conditions - many dynamic objects - the variance can reach 40% on 12 models trained with different seeds.

The authors trained a conditional network like in [Codevilla 2017]. Some changes are made to add more robustness: one module to predict the speed is added, and L1 loss is used instead of classical MSE. A new benchmark (called NoCrash) is proposed to explore the behavior of the agent in different traffic conditions (empty, regular and dense). To explore the limitations, models

have been trained with 2, 10, 50 and 100 hours of demonstration, with the best performance achieved when using only 10 hours. The dataset indeed lacks diversity, so adding more data will only lead to overfit. The authors also noticed that the agent reacts better when encountered vehicles have common model and color, which underlines the dataset bias. Identical models have also been trained with different random seeds to explore the variance, and results on the NoCrash benchmark differs highly between the two seeds. However the variance can be reduced by pretraining the CNN (here on ImageNet). The last studied effect is called inertia problem, and is due to causal confusion. Indeed in the dataset, when the agent is stopped, there is a very high probability that it will remain unmoving - because of traffic light or other vehicle ahead. Therefore in the test, there are many episodes where the agent simply stops and never starts again. The speed prediction branch reduces this problem, but does not completely eliminate it. This work points out many of the difficulties of supervised learning applied to autonomous driving, as well as research directions.

Increasing the Dataset

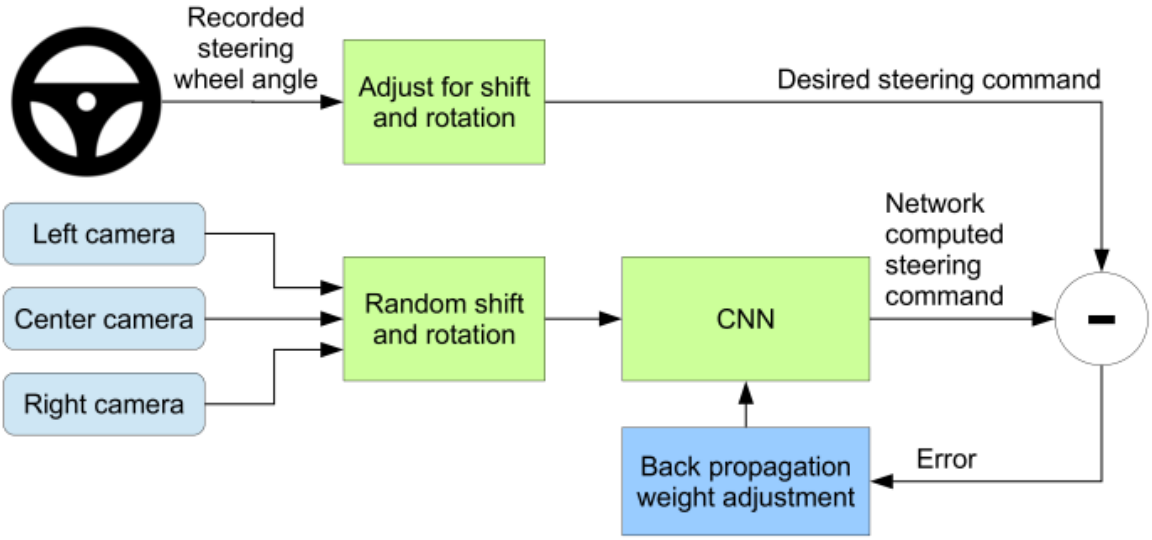


Figure 3.12: Training framework with 3 camera for data collection [Gandhi 2017]

One possible solution to distributional shift is to increase the dataset. The main problem with imitation learning is that the system doesn't know how to recover from its mistakes. When the system will commit a small mistake (and it obviously will at some point), errors will pile up since the situations have not been seen in the training data. Some have found ways to bypass this problem by extending their training datasets. In [Gandhi 2017; Codevilla 2017; Bojarski 2016], datasets are increased to add errors trajectories and the correct behavior in these cases, in order to make the learning agent more robust. These three papers deal with autonomous driving (with respectively drone and car), using a single camera. In *Learning to Fly by Crashing* [Gandhi 2017], they crashed a drone more than 11.000 times to collect a huge crash database, using a naive and random algorithm (to avoid human bias in crashes). In both *End to End Learning for Self-Driving Cars* [Bojarski 2016] and *End-to-End Driving via Conditional Imitation Learning* [Codevilla 2017], two extra cameras are added to collect more samples, and more specifically, "bad" samples: one on the left and one on the right. The images from these camera are labeled

according to the rotation of the cameras to provide bad samples during training. Moreover, the data are augmented with random shift and rotation of the recorded images (and of course groundtruth is also modified accordingly) (see Figure 3.12).

In *End to End Vehicle Lateral Control Using a Single Fisheye Camera* [Toromanoff 2018] Toromanoff et al. propose a method to increase the dataset a posteriori, without adding any additional sensor. A single fisheye camera is used to collect data. The image used to predict the steering angle is not directly the one given by the camera: a cylindrical projection is used and the image obtained is close to the image of a simple camera. The fisheye camera image is used to augment the dataset with lateral offset images, of which cylindrical projections are used. This method has the advantage of being more flexible than using several conventional cameras for data collection, but requires post-processing of the data.

Increasing data seems to be an unavoidable way to robustify a framework that uses behavior cloning. In *ChauffeurNet: Learning to Drive by Imitating the Best and Synthesizing the Worst* [Bansal 2018], not only perturbations are added, but the classical imitation loss is also modified to add additional losses to deter unwanted events. Perturbations are created to modify the trajectories in the dataset and show the agent how to recover from a wrong situation. Additional losses are: a collision loss to reduce collisions, a on road loss to encourage the agent to stay on the road and a geometry loss to encourage the agent to follow the same trajectory regardless of speed. The model also predicts the location of the other dynamic objects on the scene, so auxiliary losses are used regarding this auxiliary predictions. But contrary to lots of work presented in this section that mostly use one front facing camera, the inputs used in this framework are mid-level inputs: bird view of road map, traffic lights, boxes of the dynamic objects, route, etc., and the model outputs a driving trajectory. However, this work shows the limits of behavior cloning, and suggests ways to improve and apply it to autonomous driving.

3.2.3 Autonomous driving with Reinforcement Learning

Despite many advantages such as its speed and ease of implementation, imitation learning has many drawbacks. With the emergence of reinforcement learning and its application to increasingly complex problems, it began to be applied to autonomous driving. Because of the trial and error essence of RL and the need of a very large amount of data to converge, very little work is done directly in real world. Most of the works are carried out in simulation, especially when dealing with urban navigation.

Lane Following

As for imitation learning, some are only interested in the lateral control of the vehicle, leaving the speed control to an independent controller [Wolf 2017; D. Li 2019]. Whereas [Wolf 2017] focuses on end-to-end, [D. Li 2019] breaks down learning process in one perception module that predict track features, and one control module that uses these features as well as ego-vehicle information like current speed to compute steering angle (see Figure 3.13). The perception module is trained with multi-task supervised learning to output 3 track features: distance to lane markings, angle between track and vehicle heading, and high level road direction (left curve or right curve or straight). The track features are then used to train the control module with reinforcement learning with DPG (Deterministic Policy Gradient [Silver 2014]). In *Learning how to drive in a real world simulation with deep Q-Networks* [Wolf 2017] on the contrary, steering angle is directly computed from image input without any intermediate representation. Deep Q learning algorithm is used in this work, with then discrete actions. However, in these two works, the speed is not controlled with reinforcement learning, and thus requires an additional controller.

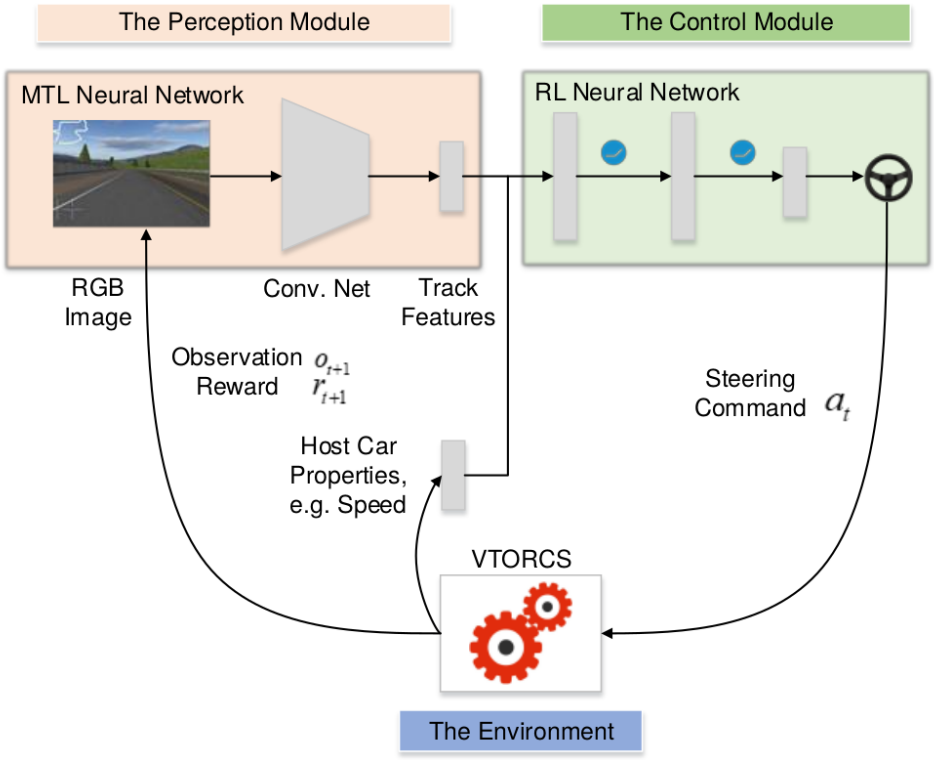


Figure 3.13: Lateral control framework [D. Li 2019]. Perception module is trained with supervised learning to computed track features that will serve as input to the RL control module.

Still in the lane following task, other works compute speed control commands (acceleration/brake) as part of a racing game [Sallab 2016; Sallab 2017; Perot 2017; Jaritz 2018]. However whereas both [Perot 2017] and [Jaritz 2018] use road image as input, [Sallab 2016] and [Sallab 2017] use simpler input like position of the track borders. Based on visual inputs, [Perot 2017; Jaritz 2018] uses A3C algorithm with discrete actions and add LSTM in the network to add temporality. The agent is trained with different driving conditions (different weather, road structure or adherence, ...) with the racing game World Rally Championship 6 (WRC6) [*World Rally Championship* 2016], and is able to generalize in unseen tracks.

Hierarchical Reinforcement Learning for Specific Driving Tasks

Others focus on tasks closer to driving in an urban environment: lane changing [Y. Chen 2019; Mirchevska 2018] and traffic light management [J. Chen 2018]. In *Deep Hierarchical Reinforcement Learning for Autonomous Driving with Distinct Behaviors* [J. Chen 2018], Chen et al. apply hierarchical reinforcement learning to handle traffic lights. Hierarchical reinforcement learning [Barto 2003] is based on the decomposition of a policy into sub-politics. In this case, the general policy is driving, and the sub-policies for traffic light management are: pass or stop. The first level policy will decide if the car must pass or stop, and the sub-policies will compute driving commands (see Figure 3.14).

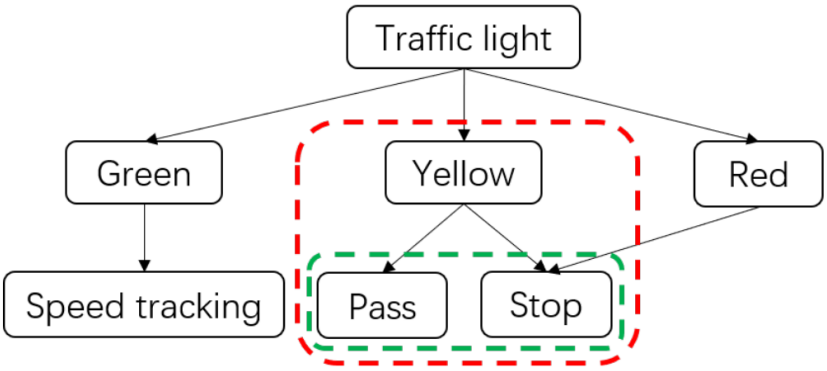


Figure 3.14: Hierarchical policy architecture for traffic light management [J. Chen 2018]

The advantage of using hierarchical reinforcement learning is that it allows smooth transfer to other task, and more interpretability and white box compared to flat reinforcement learning. The first step consists in training the primitive controller (pass and stop) individually. They output acceleration/brake intensity according to the distance to the crosswalk. Then these controllers are assembled in the structure showed in Figure 3.14, and finetuned together to handle traffic light scenario. However the state of the system is composed of the current speed of the car, the distance to the crossing line, and the time for the traffic light to turn red, the two latter not being available in real world driving scenarios. Nevertheless this work compared hierarchical reinforcement learning training with the same task trained with flat reinforcement learning that shows no good result on the traffic light scenario.

Hierarchical reinforcement learning is also used in *Attention-based hierarchical deep reinforcement learning for lane change behaviors in autonomous driving* [Y. Chen 2019] for lane change with image as input (see Figure 3.15).

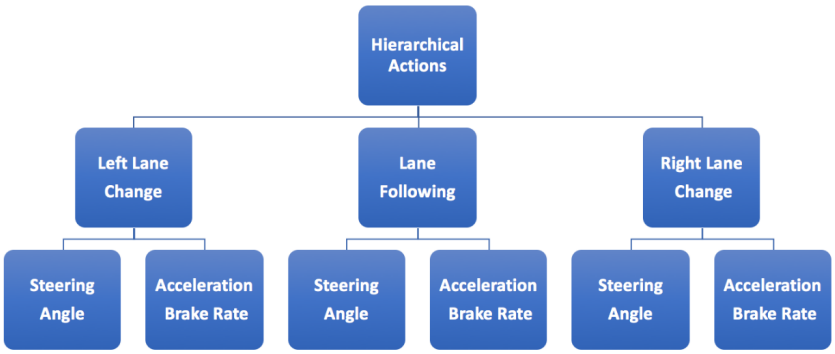


Figure 3.15: Hierarchical policy architecture for lane change [Y. Chen 2019]

In this work, the reward is constructed so that the car should remain in its lane, but also runs at a preferred speed (35 km/h), and is encouraged to overtake if the front vehicle is less than 100 meters away. The agent must first take the decision to change lane (high level command), and then compute the corresponding driving command (low level command). Training is based on DDPG, with the actor outputting the high level command from current state, and critic outputting driving command from current state and high level command. Like [J. Chen 2018], a comparison with flat reinforcement learning is performed and shows that hierarchical RL performs better on lane change task. Higher results are also achieved by adding spatial and temporal attention, as well as LSTM for temporal information. In *High-level Decision Making for Safe*

and Reasonable Autonomous Lane Changing using Reinforcement Learning [Mirchevska 2018], reinforcement learning is used to perform only high level decision: stay in line, or change lane right or left. The reward is only based on the speed of the agent (the closer to desired speed, the better), and input of the agent is composed of his current speed, as well as the relative speed of adjacent vehicles, and the distance to adjacent vehicles. Despite the fact that this information is not available in the real world, this work shows that an agent is capable of making the decision to pass a car that is moving slower than it is. Moreover what is interesting in these two works [J. Chen 2018; Mirchevska 2018] is that the agent decides by itself whether to change lane or not.

Driving in Complex Urban Environments

All of the approaches presented above apply reinforcement learning to autonomous driving, but focus on the task of following the road, without worrying about intersections which are an important part of driving in an urban environment. The first to experiment with conditional driving with reinforcement learning are Dosovitskiy et al. in *CARLA: An Open Urban Driving Simulator* [Dosovitskiy, Ros, 2017]. Together with the baseline training in imitation learning and modular pipeline, they present the first end-to-end RL training for conditional driving with vision-based input. The training is performed with A3C for 10 million iterations. This first complete autonomous driving baseline in an urban environment produces much lower results than those produced by supervised learning or the modular pipeline. Under training conditions, for the navigation task, the agent trained with reinforcement managed to complete only 14% of the episodes, compared to 86% under supervised training.

A more recent approach, *CIRL: Controllable Imitative Reinforcement Learning for Vision-based Self-driving* [Liang 2018], combines both supervised learning and reinforcement learning by pretraining the network with supervised learning (see Figure 3.16)

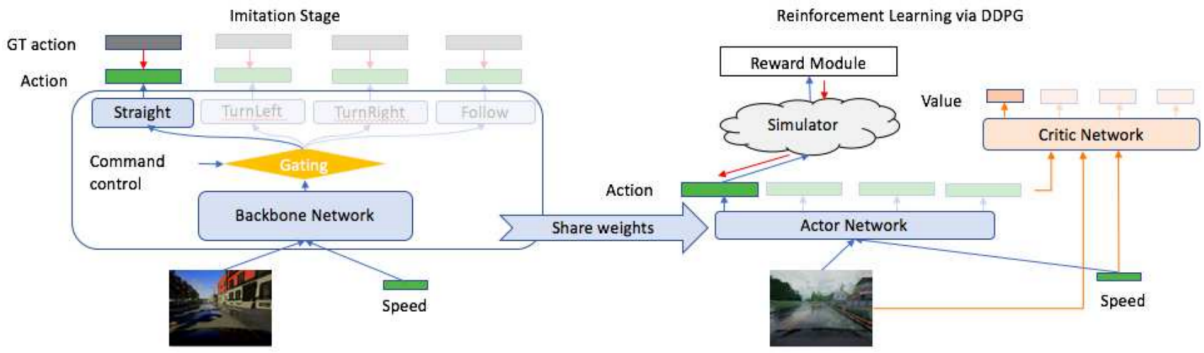


Figure 3.16: CIRL training framework [Liang 2018]. The backbone network is first trained with supervised learning, and then used to initialize the weight of reinforcement learning training.

Like [Dosovitskiy, Ros, 2017], CIRL uses a branched network to take the driving command into account, and experimental settings are identical is these two works. The actor-critic algorithm Deep Deterministic Policy Gradient [Lillicrap 2016] is used to train RL agent. Actor and critic are two separate networks, and only the actor uses the pretrained weights from imitation learning as initializer. The imitation part is trained with the same 14 hours of driving as [Dosovitskiy, Ros, 2017], and RL finetuning is performed for 0.5 million steps (against the 10 Millions used to train RL from scratch in [Dosovitskiy, Ros, 2017]). CIRL outperforms both supervised and reinforcement learning training on Carla CoRL benchmark.

The work of Rhinehart et al. in *Deep Imitative Models for Flexible Inference, Planning, and*

Control [Rhinehart 2020] also proposes a combination of supervised and reinforcement learning for conditional driving. But unlike CIRL where reinforcement learning is model-free, a model-based approach is used. As before, the combination of supervised and reinforcement learning allows for a shorter and safer training. In fact, reinforcement learning learns by trial and error, by collecting data during training, with a model that is not optimal, which can lead to dangerous behaviors. In [Rhinehart 2020], an imitative model is built which allows to prefer the expert's trajectory, contrary to classical reinforcement where all options are possible. The model learns from Lidar points and the few previous positions of the agent to plan a trajectory according to a given objective (a point located a few meters or even ten meters from the agent). During the test phase, the trajectory is given to a PID controller that calculates the commands (see Figure 3.17).

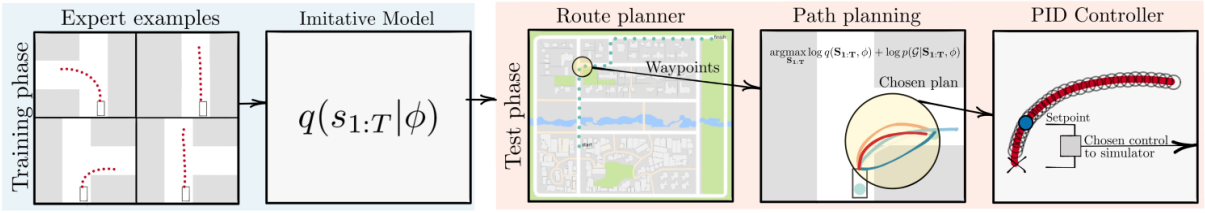


Figure 3.17: Deep imitative training and test phases [Rhinehart 2020].

Finally, *End-to-End Model-Free Reinforcement Learning for Urban Driving using Implicit Affordances* [Toromanoff 2019] also presents a reinforcement learning framework for autonomous driving divided in two steps. The encoder is first trained in a supervised way to compute affordances: semantic segmentation, state and distance to traffic light, distance to crossing, or ego vehicle angle with the road (see Figure 3.18).

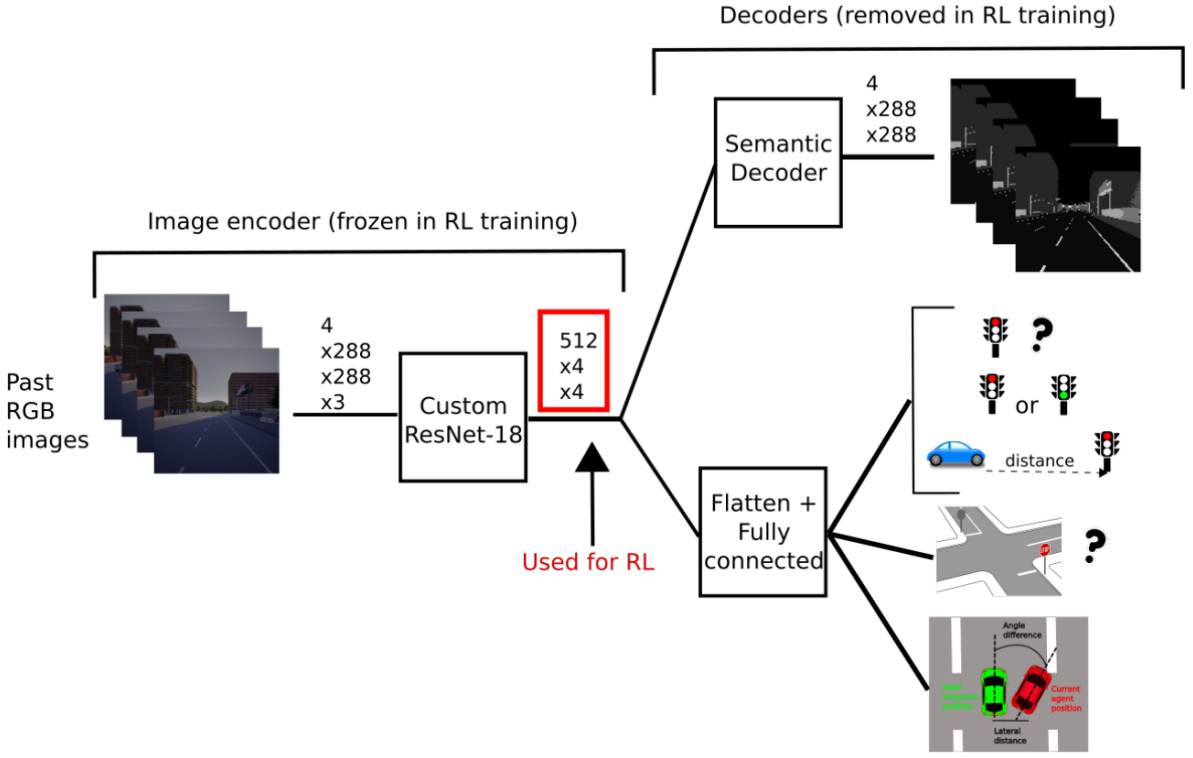


Figure 3.18: Supervised affordance training [Toromanoff 2019].

The encoder is then frozen and its output is used for reinforcement learning training. In this framework, the input is composed of 4 road images (to include temporality) and the current vehicle speed. The driving instruction is given as high level command (turn left, change lane, etc...), and to take it into account, a branched architecture like [Dosovitskiy, Ros, 2017; Codevilla 2017] is used. This work achieved very high score on CARLA benchmarks [Dosovitskiy, Ros, 2017; Codevilla 2019] as well as on the CARLA Challenge [*CARLA Autonomous Driving Challenge* 2019].

Driving in Real World

The majority of the works on autonomous driving with reinforcement is done in simulation. However, some works are performed directly in the real world. Among the first experiments with autonomous driving with reinforcement learning we can find the works of Michels et al. [Michels 2005] and of Riedmiller et al. [Riedmiller 2007]. In *High speed obstacle avoidance using monocular vision and reinforcement learning* [Michels 2005] a RL agent is trained to avoid obstacles in unstructured outdoor environment from depth prediction. The vision system is trained with supervised learning to predict depth from monocular camera input image, and the controller is trained from depth input to control command with reinforcement learning. In *Learning to drive a real car in 20 minutes* [Riedmiller 2007], the controller in also trained with reinforcement learning, using the car position and current speed as input. The robot car must follow a given track, and the reward function is based on the distance to the given track (the more the robot is in the middle of the track, the better). The robot learns to follow a given path in 20 minutes with reinforcement learning. These two experiments have however been realized with small robot cars. The scaling on a real car was carried out by Kendall et al. in *Learning to drive in a day* [Kendall 2018]. Their agent learns to perform lane following in both simulation and real world, using only images from a front-facing camera as input (as well as speed and steering measurements) (see Figure 3.19).

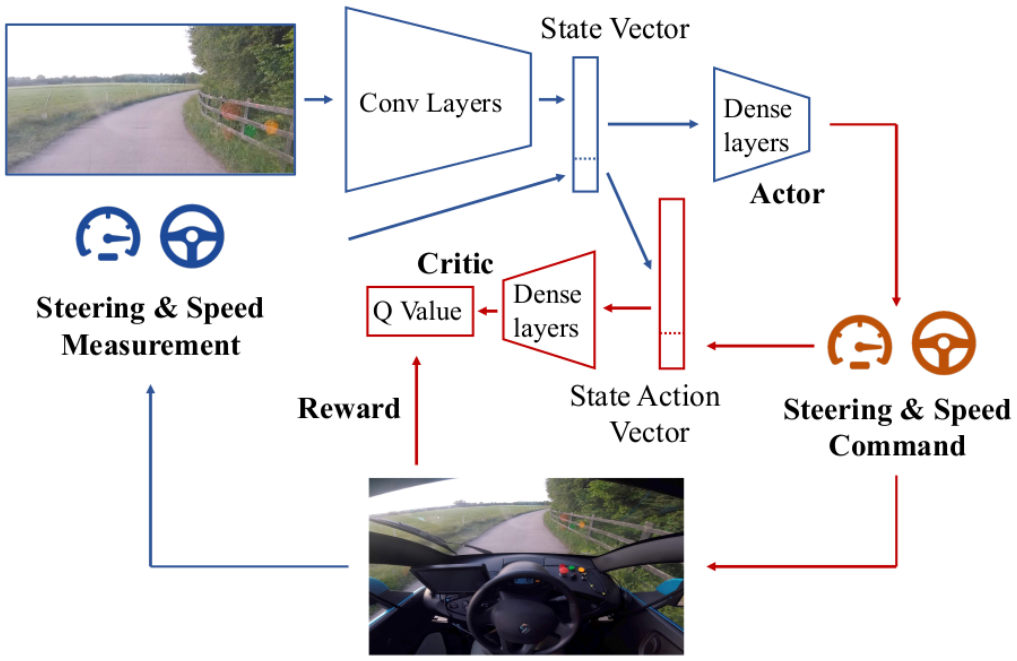


Figure 3.19: Reinforcement learning architecture to train a real car to drive in real world [Kendall 2018]

The algorithm used is DDPG [Lillicrap 2016] which is an actor critic, and the work on simulation is essentially used to tune the hyperparameters of the framework. On the real vehicles, a safety driver takes control of the vehicles when it deviates too much from the road center, and the taking over indicates the end of an episode. This work also explores the use of a Variational AutoEncoder (VAE). A decoder of the same size as the encoder is added and trained at the same time as the driving task to reconstruct the original image. The addition of an autoencoder as auxiliary task allows for much faster and more efficient learning. It is also important to notice that no transition from simulation to real world is performed, the training to drive in real world is directly realized on the real car. It is however difficult to use reinforcement learning directly in the real world. Indeed, RL is based on the interaction with the environment, and on the principle of trial and error, where the agent learns from its own mistakes. For obvious security reasons, most of the work is done in simulation, and the few works done in real world are focused on road following with simple and empty environments.

To overcome these limitations, part of the research is interested in the transition from simulation to real world. In *Virtual to Real Reinforcement Learning for Autonomous Driving* [X. Pan 2017], Pan et al. propose a translation network to bridge the gap between simulation and real world, so that an agent trained only to drive in simulation is able to drive in the real world. The translation network enables the transition from simulation to synthetic realistic image, through segmentation (see Figure 3.20). The first part of the translation network segments the simulated image to obtain semantic information. The second part transforms the semantic segmented image to realistic image. The translation network is first trained as a GAN, and then used to train a driving agent with A3C algorithm. As this driving agent only sees realistic images, smooth transition to real world can be performed.

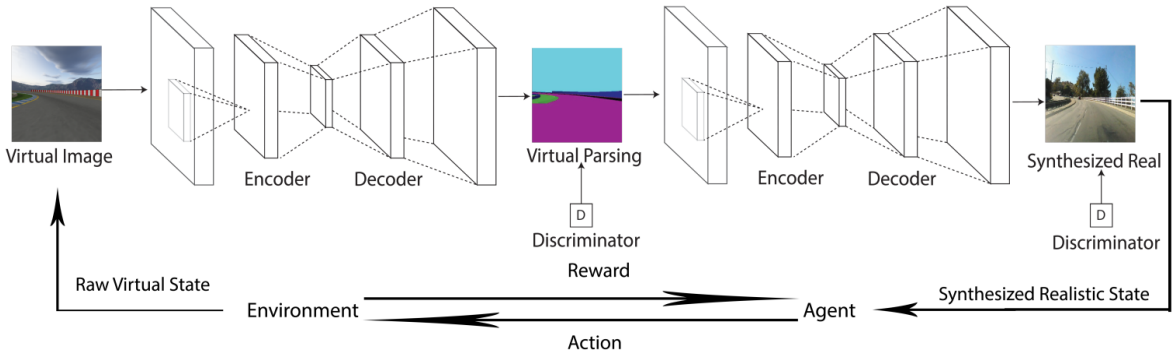


Figure 3.20: Virtual to real reinforcement learning framework [X. Pan 2017]. Translation network transforms simulated images to realistic one through semantic segmentation so that the agent learns from near-reality images and is able to drive in real world.

Extension: Driving with Birdview Input

We focused earlier on the use of a single onboard camera coupled with driving instructions. In this section we study some works using more trajectory information to teach an agent to drive. Both [Agarwal 2019] and [J. Chen 2019] use route information which could for example be provided by a gps. The agent does not need to plan where to go, which simplifies the task compared to driving with high level driving instructions. It can be noted that humans are capable of doing both: we are capable of following a high level command (and unconsciously we plan a trajectory ourselves), and also of following a gps that provides us a trajectory. These methods are not mutually exclusive.

Learning to Drive using Waypoints [Agarwal 2019] proposes a framework for autonomous driving that uses birdview segmented image and route waypoints as input. As shown in Figure 3.21, input segmented image is passed through an encoder, and its features are then concatenated with waypoints features to output steering angle and target speed. Target speed is given to a PID controller that outputs throttle/brake command. The encoder is first trained as a part of an autoencoder to reconstruct the segmented image. It is then frozen during RL driving training, while being regularly finetuned on the collected segmented images to improve stability.

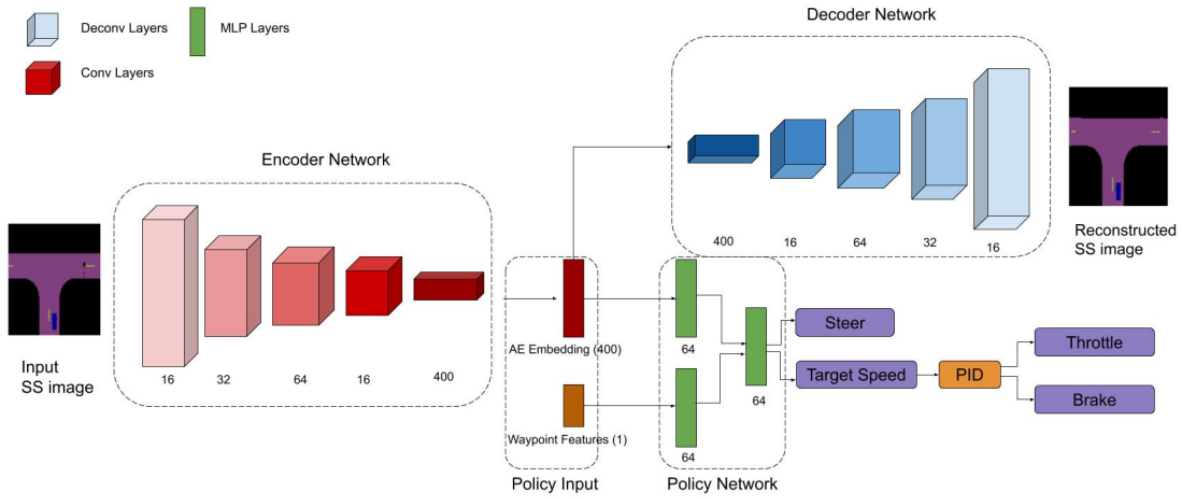


Figure 3.21: Autonomous driving framework using birdview semantic segmentation image called WRL (Waypoint based DRL) [Agarwal 2019]

This work achieved for the first time high score in Carla CoRL benchmark [Dosovitskiy, Ros, 2017] using only reinforcement learning for driving policy, with more than 90% successful episodes on navigation task (percentage that falls to 79% - or even 60% on an unseen town - on the dynamic navigation task).

The work *Model-free Deep Reinforcement Learning for Urban Autonomous Driving* [J. Chen 2019] from Chen et al. explore even more complex scenarios, with roundabounds (versus only simple crossroads in [Agarwal 2019]). Like [Agarwal 2019] they use two distinct steps. The driving agent is explicitly divided into two modules (see Figure 3.22): one perception module with the latent encoding, and one control module, the RL agent. The encoder is first trained with supervised learning using a Variational AutoEncoder (VAE), and frozen during RL training for agent driving control. In this work the input is composed of a birdview image from the map containing the routing as well as ego vehicle and other surrounding objects.

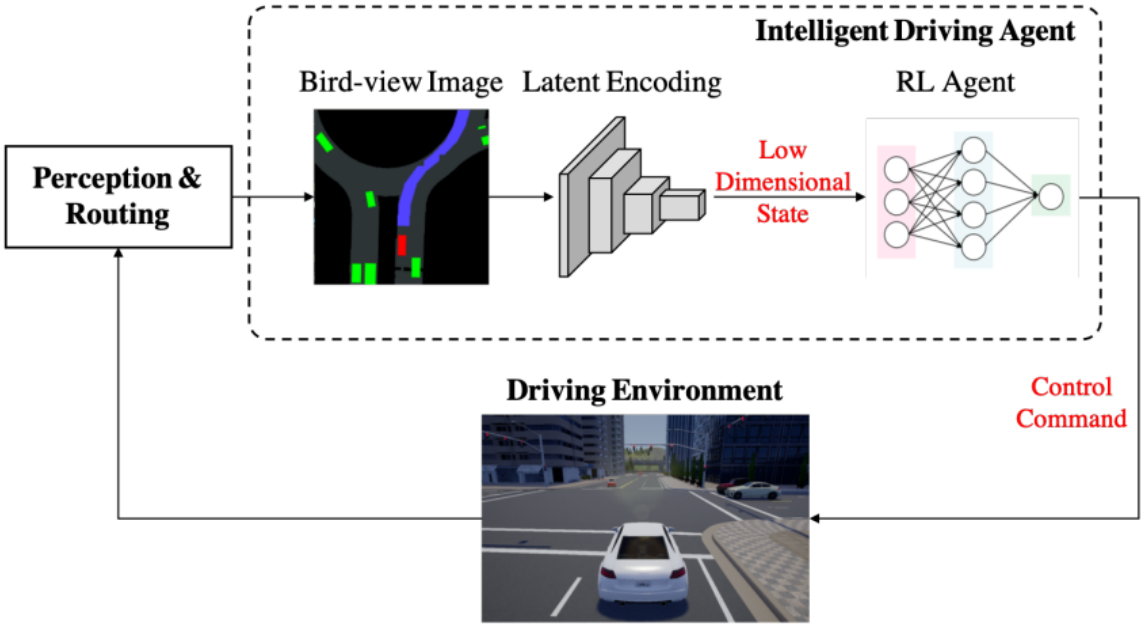


Figure 3.22: Autonomous driving framework using birdview image [J. Chen 2019]. The birdview image contains the map, as well as the route to follow, the position of the agent and of the others dynamics objects around. The latent encoder is trained via a VAE, and frozen during RL driving training.

However, even if we can imagine having the road given as input (using GPS), getting the positions of all the objects surrounding the vehicle (as well as semantic segmented birdview image used in [Agarwal 2019]) seems more complex in the real world, and requires additional data processing before this algorithm can be used.

3.3 Applications Related to Autonomous Driving

In the previous section we presented several works on autonomous driving with visual input. In this section, we introduce related work that are not strictly speaking autonomous driving, but which ideas can be easily transferable. In particular, we will focus on image based tasks. The first to successfully apply reinforcement learning to a vision-based task are Mnih et al. in *Playing Atari with Deep Reinforcement Learning* [Mnih 2013]. Deep Q learning is successfully applied on 7 Atari games, and later in [Mnih 2015] on the 49 Atari games, and outperforms human levels on a large number of them.

3.3.1 Navigation in Maze / Vision-Based Game

Slightly more complex game environments than Atari are also used: Deep Mind Lab [Beattie 2016] for maze navigation in [Jaderberg 2016; Mirowski 2016], and Vizdoom [Kempka 2017] which is a FPS (First Person Shoot) game in [Lample 2017] (more details about the environment in the section 3.4). These three works [Jaderberg 2016; Mirowski 2016; Lample 2017] use reinforcement learning with visual inputs, and all use auxiliary tasks to improve the performance of their agents.

In *Learning to Navigate in Complex Environments* [Mirowski 2016], the authors compare several architectures on different navigation tasks, ranging from a small static maze to a large

maze with a random goal. The differences between the compared architectures are the following (see Figure 3.23): a simple feedforward model, addition of recurrent layer, addition of additional input (current speed, previous action and reward), and finally additional outputs (loop closure and depth prediction). These additional outputs, computed as auxiliary tasks, are intended to teach the agent an intrinsic representation of its environment. Indeed, calculating the depth allows a better representation of the space. Finally, calculating the loop closure allows the agent to learn how to detect the places where it has already passed, and thus to allow an efficient exploration. Training is realized with A3C algorithm [Mnih 2016], and experiences show that training with auxiliary task allows a better resolution of the most complex tasks

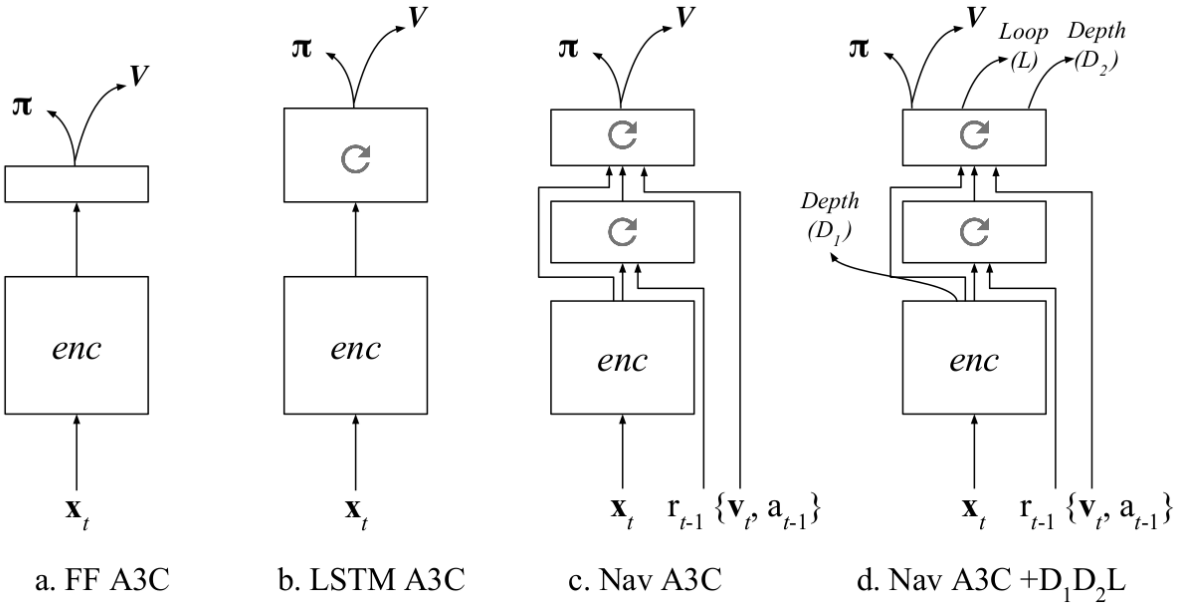


Figure 3.23: Different architectures: (a) is a convolutional encoder followed by a feedforward layer and policy (π) and value function outputs; (b) has an LSTM layer; (c) uses additional inputs (agent-relative velocity, reward, and action), as well as a stacked LSTM; and (d) has additional outputs to predict depth and loop closures. Figure and caption from [Mirowski 2016]

Using auxiliary task to help an agent is also used in *Playing FPS Games with Deep Reinforcement Learning* [Lample 2017] on Vizdoom FPS game. The goal of the agent in this case is to navigate in the game to find and kill enemies. The task is then divided in two subtasks: navigation and shooting. The model used is a small CNN followed by a LSTM layer, and is trained with DQN. Simple training did not produce good results, so an auxiliary task is added to predict game features (more precisely the presence of certain entities - enemy, health pack, weapon... - in the considered frame). This auxiliary prediction is shown to improve the performance of the agent.

Finally using additional task to improve maze navigation is reiterated in *Reinforcement Learning with Unsupervised Auxiliary Tasks* [Jaderberg 2016], on the same maze environment DeepMind Lab [Beattie 2016] like [Mirowski 2016]. The navigating agent called UNREAL is trained with A3C, together with two types of auxiliary tasks (see Figure 3.24). Auxiliary tasks are added to enable a faster, more robust and efficient training, and are called unsupervised as they do not need extra supervision or signal than the base agent. First of all, auxiliary control tasks are used, which produce additional rewards to the agent. They are composed of pixel changes (which measures the difference between the pixels of two consecutive images, and encourages the

agent to explore) and network features (which encourages the agent to activate its hidden layers as much as possible). The second type of auxiliary task is reward prediction, where the agent learns to predict the next reward with historical context (i.e. previous image inputs). The replay buffer is also used to perform value function training to enable a faster value iteration. Contrary to [Mirowski 2016], the auxiliary tasks used do not need extra features (like depth for instance). The auxiliary tasks are internal to the agent.

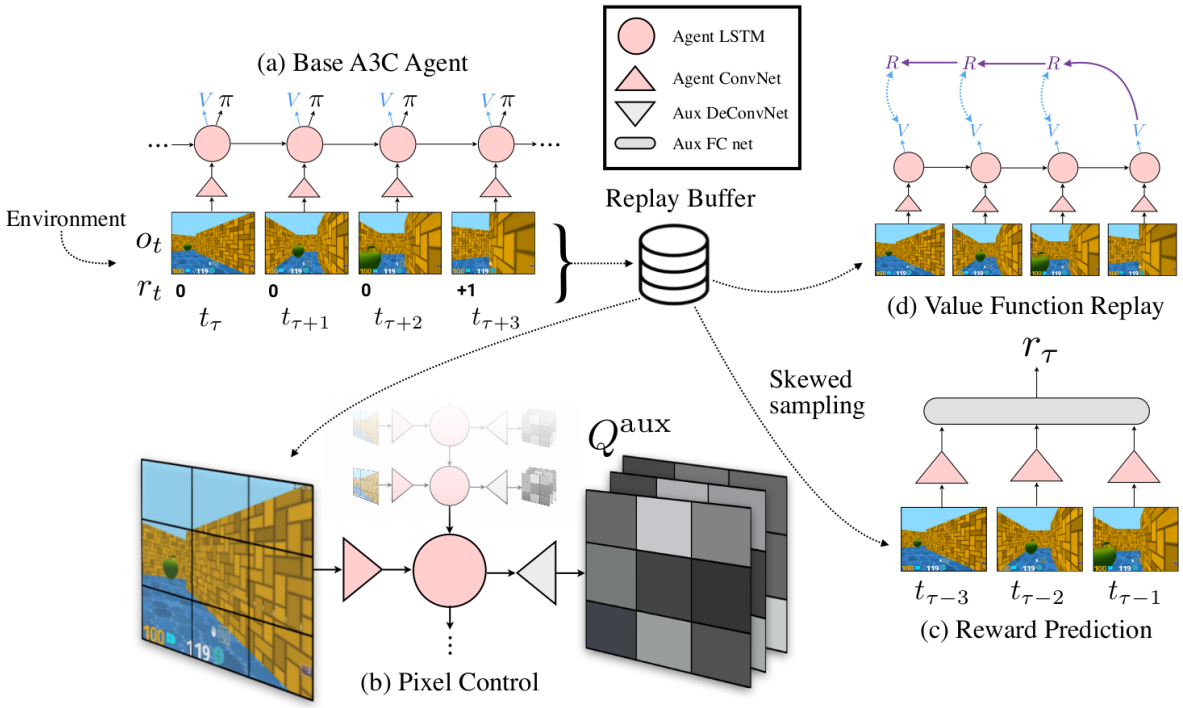


Figure 3.24: UNREAL agent [Jaderberg 2016]

However, gaming environments are simpler than those related to driving (especially in urban areas). In particular the size of the images used by the agent is relatively small ($84 \times 84 \times 3$ for [Jaderberg 2016; Mirowski 2016] and $108 \times 60 \times 3$ for [Lample 2017]).) compared to that used for autonomous driving ($200 \times 88 \times 3$ for [Dosovitskiy, Ros, 2017], $384 \times 160 \times 3$ for [D. Chen 2020], or $4 \times 288 \times 288 \times 3$ for [Toromanoff 2019])

3.3.2 Target Driven Navigation

The work in *Target-driven Visual Navigation in Indoor Scenes using Deep Reinforcement Learning* [Zhu 2017] introduces a target-based navigation. The agent's goal is to find a specific object in a room. To do so, the authors Zu et al. use an indoor navigation simulator, the AI2-THOR framework [Kolve 2017b] (see section 3.4 for details). A network is trained with the image of the target and the current image seen by the agent as inputs. This makes the algorithm more flexible to change of target. A siamese network with shared weights is developed containing a 50-layers ResNet pre-trained on ImageNet, and frozen for navigation training. After the siamese network, feature from both observation and target images are fused and passed through scene specific layers, i.e. branches for the different rooms (see Figure 3.25 for architecture details). The scene specific layers are used to catch the characteristics of every scene, and the authors show that the agent has better results when it contains specific layers for each scene

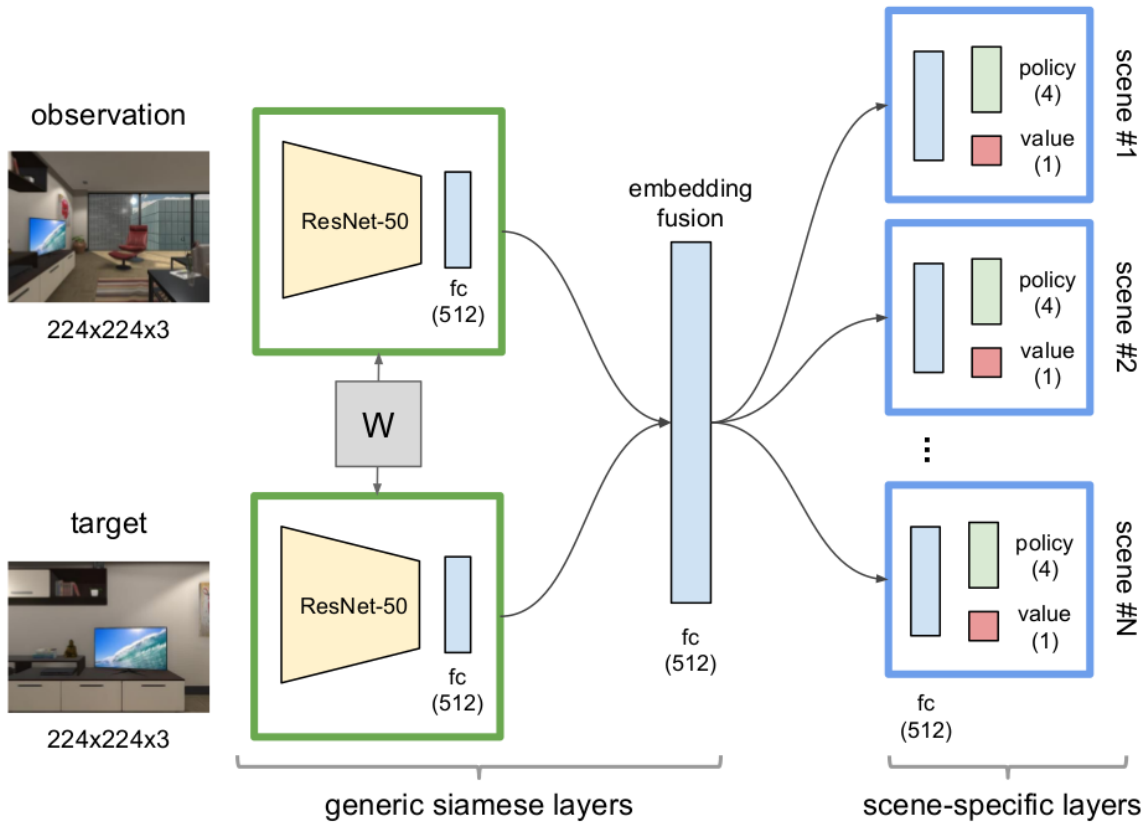


Figure 3.25: Target driven navigation network architecture with scene specific layers [Zhu 2017].

Possible actions are discrete: moving forward, moving backward, turning left, and turning right, and the agent is trained with 20 different indoor scenes with A3C algorithm. In *Vision-based Navigation Using Deep Reinforcement Learning* [Kulhánek 2019], the authors tackle the same problem as [Zhu 2017], increasing complexity. In their case their CNN is neither pre-trained nor fixed, and their framework is tested in several environments: AI2-THOR [Kolve 2017b], but also DeepMind Lab [Beattie 2016] and House3D [Wu 2018]. To improve the performance of their agent, auxiliary tasks are added: both unsupervised tasks like UNREAL: pixel control and reward prediction [Jaderberg 2016], but also supervised auxiliary tasks: depth map, observation segmentation and target segmentation (see Figure 3.26).

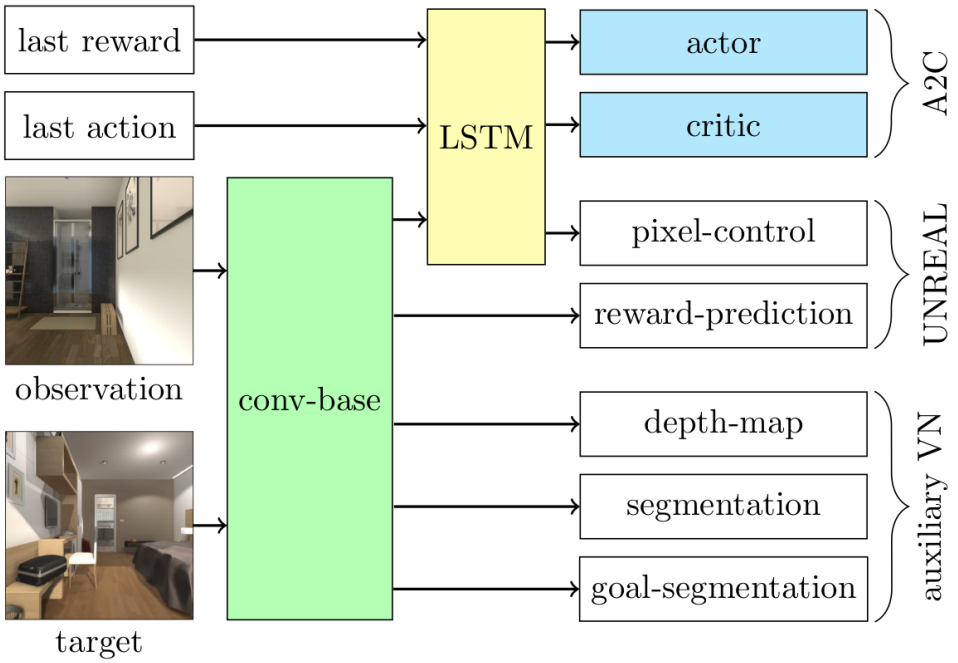


Figure 3.26: Target driven navigation network architecture with supervised and unsupervised auxiliary tasks [Kulhánek 2019]

The two previous papers indicate the target with the help of an image: the agent has a visual input of the object or place to be found. Others have looked at language-guided navigation. In both [Chaplot 2018] and [Wu 2018], the instructions are given in common language. In *Gated-attention architectures for task-oriented language grounding* [Chaplot 2018], the agent learns to navigate in a Doom environment following given instruction (eg. 'go to the green torch', 'go to the smallest blue object'), whereas in *Building generalizable agents with a realistic and rich 3D environment* [Wu 2018], navigation takes place inside a house. Wu et al. propose a new dataset for indoor navigation, and a task consisting in giving the agent a semantic concept (e.g. go to the kitchen), and considering the task fulfilled when the agent has reached the given room. In this environment, the agent can chose between a finite number (12) of high level actions. The indoor target-based navigation task, called RoomNav, is trained with A3C. Both [Chaplot 2018] and [Wu 2018] use a gated attention mechanism to follow the instruction (see Figure 3.27). This module takes features from image input and from instruction representation, and computes the Hadamard product (element-wise matrix product).

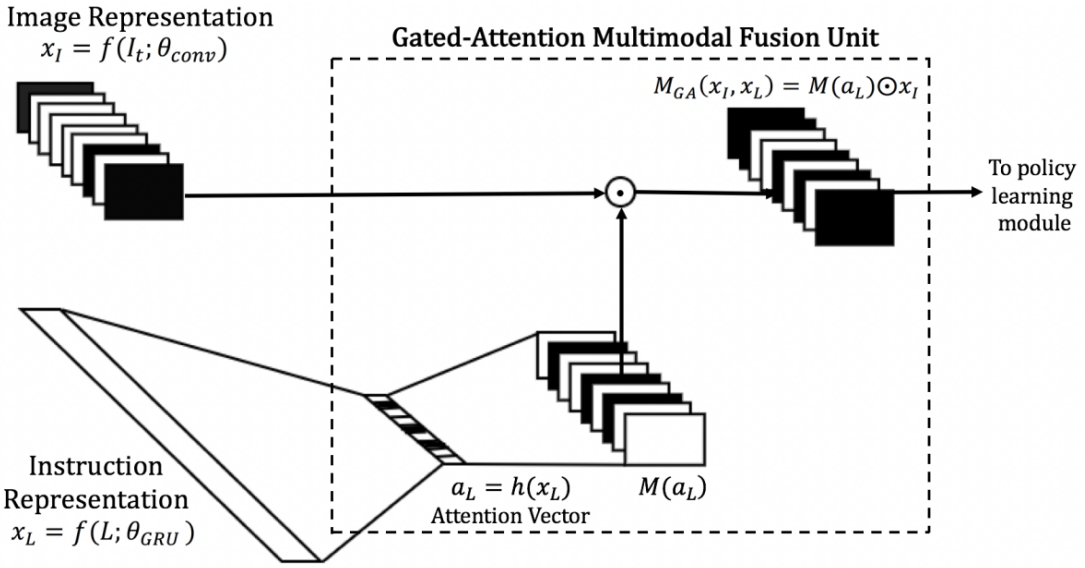


Figure 3.27: Gated attention architecture [Chaplot 2018]

Finally in *Learning to Navigate in Cities without a Map* [Mirowski 2018], a team from DeepMind has developed an agent that can solve the Courtier task. This task consists in navigating without a map, and is actually what humans do in their neighborhood for instance. When we want to go to the bakery, we find our way with visual help from our environment. The agent navigates in Google Street View interface, and can chose between a fixed number of actions (5 actions are possibles, go straight or rotate with $\pm 22.5^\circ$ or $\pm 67.5^\circ$). It has two inputs: the current image, and the goal description. The goal is described with its proximity with a set of landmarks, that are specific to each city. To solve the Courtier task, a goal-dependent reinforcement learning algorithm is used.

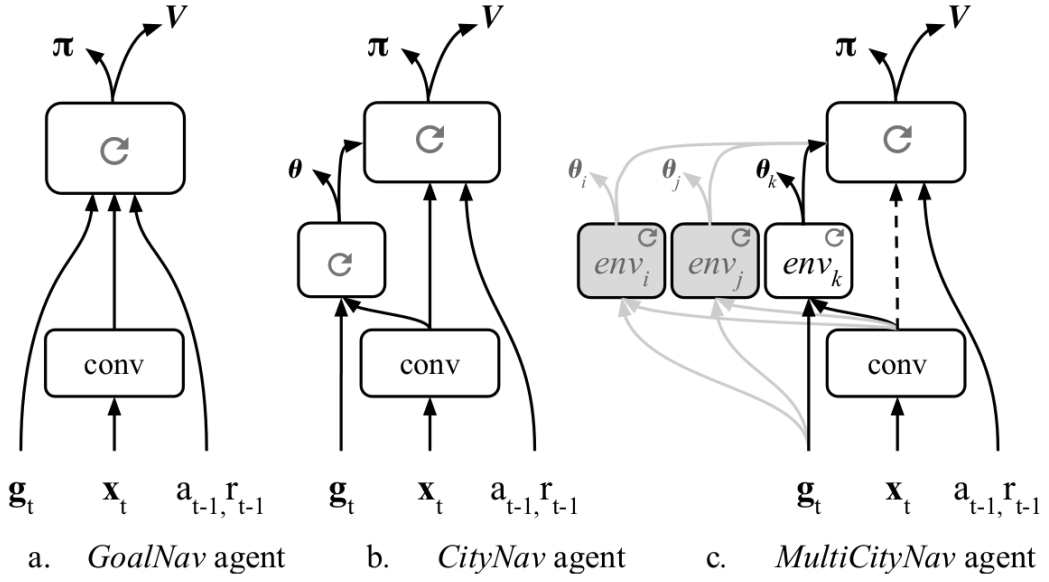


Figure 3.28: Overview of goal directed navigation architectures [Mirowski 2018].

In the Figure 3.28, the different architectures of the agent are shown. GoalNav is composed of a convolutional encoder plus a LSTM with goal description input, CityNav is a single city architecture and adds a goal LSTM. The idea is to have two parts in the learning: the global part, the encoder and the LSTM, and the locale-specific, the goal LSTM. The last architecture, MultiCityNav, enables to learn in different city. The encoder and the general LSTM are general enough for all the Cities, and the goal LSTM is City-specific. As a result, the trained agent can navigate through a lot of big cities (Paris, London, New York), and can learn to navigate in new cities without forgetting the previously learned ones.

3.3.3 Summary

In this section we have presented work dealing with vision-based navigation. Several interesting ideas emerge. The first one is the use of auxiliary tasks, either unsupervised - internal to the agent - like reward prediction, or supervised like depth prediction. As for urban driving, we also find the idea of branches to navigate in different places (different rooms or different cities), while keeping a common encoder for feature extraction.

3.4 Tools: Available Simulation Environments

In this section we present a non exhaustive list of available environments to work with. We divided the environments in 3 categories: games, indoor environments, and driving environments - simulator or datasets.

3.4.1 Games

Games are probably the simplest and most widely used environments for reinforcement learning based on visual input. Fast, they allow for effective agent-environment interaction. The first to be used are Atari games [Bellemare 2013], of which Enduro is the closest to driving. However, it is very simple, the possible actions are very limited (see Figure 3.29).

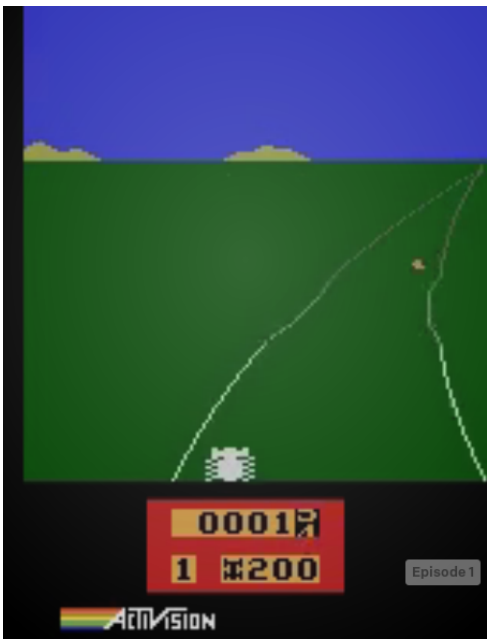
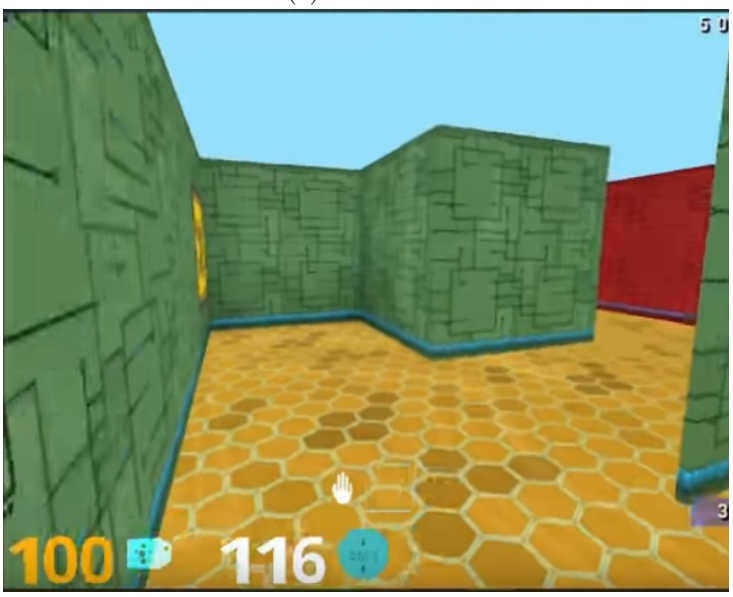


Figure 3.29: Enduro

ViZDoom [Kempka 2017; Wydmuch 2019] and DeepMind Lab [Beattie 2016] are games designed specifically for AI research. DeepMind Lab is a maze environment, and ViZDoom is a FPS (first person shoot) game. Both use first person view and 3D environments, and can be customized by the user. But whereas DeepMind Lab is only a maze, with sometimes rewards to collect, ViZDoom is a game where the agent has to move and kill its enemies at the same time. However these games remains virtual environments with no realistic rendering, and no action that can easily be transfered in real world.

(a) ViZDoom



(b) Deep Mind Lab

Figure 3.30: DeepMind Lab and ViZDoom

3.4.2 Indoor Environments

(a) MINOS (b) AI2 THOR

(c) HoME (d) House 3D

Figure 3.31: Indoor Simulator

Before focusing on autonomous driving, we had considered navigation in a more general way, hence the overview of indoor environments. To handle indoor navigation, many indoor simulators or datasets are available in literature (see Figure 3.31). SceneNet RGB [McCormac 2016] and SUNCG dataset [Song 2017] are two examples of indoor datasets, with a high number of images. Both are synthetic, and offer semantic segmentation, however SceneNet RGB [McCormac 2016] is photorealistic. Minos [Savva 2017], House3D [Wu 2018], HoME [Brodeur 2017] are indoor simulators where the agent can navigate in a house, and finally, AI2 THOR [Zhu 2017], [Kolve 2017a] contains 3D photorealistic indoor scene where the agent can act on the environment, e.g. move/push objects.

3.4.3 Driving Environments

With the development of autonomous driving, many learning environments have been developed. They can be classified in 2 categories: datasets and simulators.

One of the most famous dataset for vision-based driving task is the Kitti dataset [Geiger 2013; Geiger 2012]. It provides a large number of images collected in Karlsruhe, Germany, with image and groundtruth data for stereo vision, optical flow, visual odometry, 3D object detection and 3D tracking. This dataset proposes real world images, in a lot of challenging situations (eg. median strip, narrow road with parked cars, tramway line, etc...).

Kitti Dataset presents many derivatives: [Menze 2015] with dynamic scenes, Kitti-Road

Figure 3.32: Image from Kitti Dataset [Geiger 2013]

[Fritsch 2013] for road and lane detection. Virtual Kitti dataset [Gaidon 2016; Cabon 2020] was proposed as a virtual extension of Kitti. The scenes are cloned from the Kitit dataset, and extended to add more variety in weather or lighting conditions, as well as groundtruth data for depth map, semantic segmentation, optical flow and object detection and tracking.

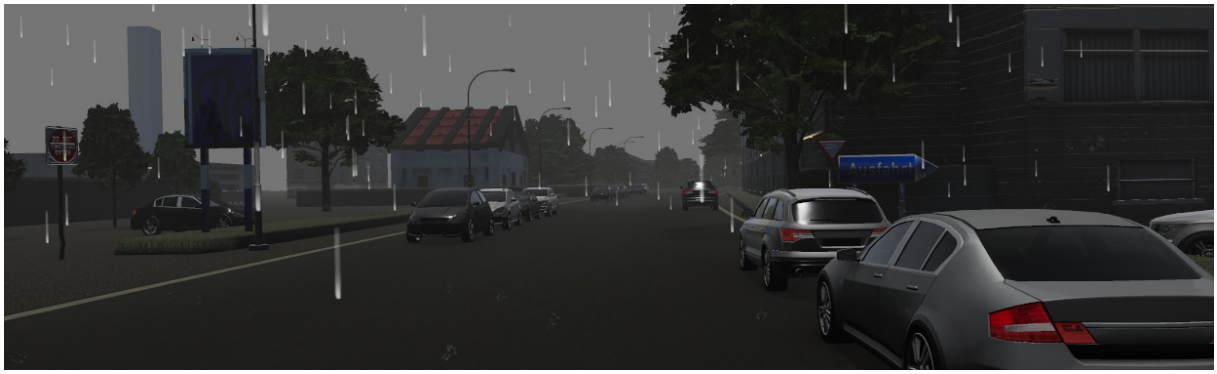


Figure 3.33: Image from Virtual Kitti dataset [Gaidon 2016]

SYNTHIA [Ros 2016] (SYNTHetic collection of Imagery and Annotations) is a large photorealistic driving dataset with semantic segmentation and depth prediction. SYNTHIA proposes a large amount of data (more that 200 000 labeled images), with different lighting conditions, weathers and seasons (see Figure 3.34), as well as different types of cities (European town, modern city, highway, green areas). Cityscape [Cordts 2015; Cordts 2016] is also a urban dataset for semantic segmentation, but with real world images, collected in many city in Germany.

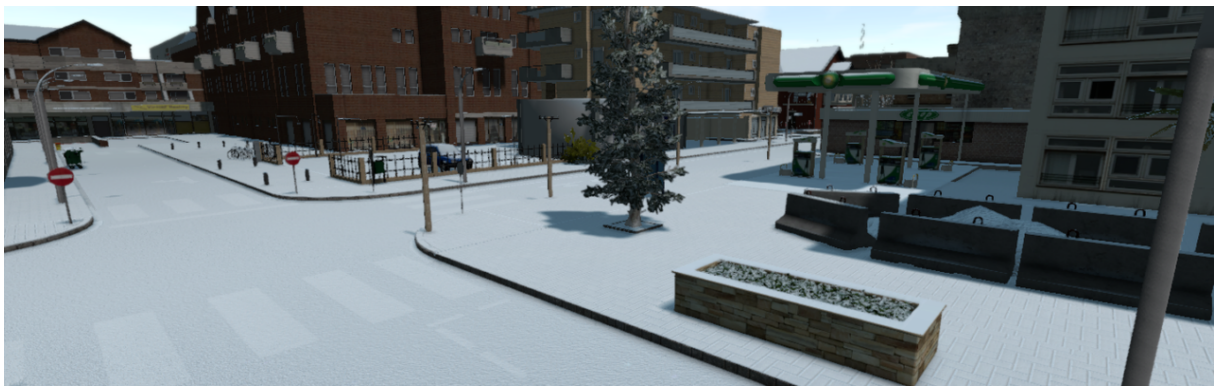


Figure 3.34: Images from SYNTHIA dataset [Ros 2016]

Figure 3.35: Segmented image taken in Stuttgart from Cityscape dataset [Cordts 2015]

Recently Waymo released their own dataset [Sun 2020] collected in three US cities: San Francisco, Phoenix, and Mountain View. It provides camera and Lidar data on a huge amount of real world images. We can mention many other datasets for autonomous driving: CamVid [Brostow 2009], Torontocity [S. Wang 2017], Robotcar [Maddern 2017], Audi Autonomous Driving Dataset (A2D2) [Geyer 2020], Appoloscape [Huang 2020], Mapillary [Neuhold 2017], Playing for Benchmarks [Richter 2017], nuScenes [Caesar 2020] and the very recent BDD100K - DeepDrive Dataset from Berkeley [Yu 2020].

These datasets, although very complete, do not allow interaction with the environment. For that it is better to turn to driving simulators. The simplest is Javascript racer [Gordon 2012], a driving game. It has the advantage of being very simple, and therefore to run quickly on a computer. However the design is very simplified and far from a real image. Torcs [Wymann 2014], still a racing game, offers a more photorealistic design, and is used in many research projects.

(a) Javascript Racer (b) TORCS

Figure 3.36: Racing games Javascript racer and Torcs

Finally for urban driving, we can cite two open-source simulators: Carla [Dosovitskiy, Ros, 2017] and Airsim [Shah 2017]. Airsim is a simulator designed by Microsoft AI & Research,for both drone flight and car driving, and Carla is a urban driving simulator sponsored by Intel and Toyota. These two simulators offer photorealistic images, and many features. They both propose complex urban environments (see Figure 3.37), with different lightning and weather conditions. Several sensors are available: camera, lidar, radar, semantic segmentation.

(a) CARLA (b) AirSim

Figure 3.37: Airsim and Carla simulators

3.4.4 Summary

We have presented in this section different environments. From simple games to driving simulators, the literature presents a rather large choice of working environments. We summarize in the following table the different characteristics of the simulators. As the datasets are not comparable in the same way, we do not include them in this table.

Environment	3D	Photo -realistic	Large -scale	Fast	Customizable	Actionable	Physics
Atari [Bellemare 2013]				√			
VIZDoom [Kempka 2017]	√			√	√		
Deep Mind Lab [Beattie 2016]	√			√	√		
AI2 THOR [Zhu 2017] [Kolve 2017a]	√	√		√	√	√	√
MINOS [Savva 2017]	√		√		√		
House3D [Wu 2018]	√		√	√	√		
SceneNet RGB [McCormac 2016]	√	√	√	√			
HoME [Brodeur 2017]	√		√	√	√		√
Javascript Racer [Gordon 2012]				√			
Torcs [Wymann 2014]	√			√			√
AirSim [Shah 2017]	√	√	√				√
CARLA [Dosovitskiy, Ros, 2017]	√	√			√		√

Table 3.1: Summary of environment - table inspired from [Savva 2017], [Kolve 2017a], [Wu 2018] and [Brodeur 2017]. **3D**: Supports 3D Settings. **Photorealistic**: life-like images. **Large-scale**: Many environment available. **Fast**: High FPS possible. **Customizable**: Adaptable with different tasks. **Actionable**: An agent can act on the environment (move objects...). **Physics**: The environment is subject to the force of gravity and to other external forces

3.5 Synthesis and Research Trails

In this state of the art, we have studied different frameworks for autonomous driving, and more broadly for navigation. The methods we presented used supervised learning, reinforcement, or a combination of both.

It is interesting to note that very few approaches focus on a complete image-based end-to-end with reinforcement learning, and the few that do are limited to lane following [Kendall 2018; X. Pan 2017]. Those who focus on more complex driving problems, i.e. urban driving, simplify their state space by using segmented birdview images [Agarwal 2019], or by pre-training the agent with supervised learning (CIRL [Liang 2018]), or finally by pre-training a part of the network in a supervised way to predict affordances that will be used as inputs for a RL agent [Toromanoff 2019]. In none of these cases did the agent learn directly from image and with reinforcement learning. In this thesis we want to study the feasibility of end-to-end learning with reinforcement learning.

We also notice in the literature a wide use of intermediate affordances or representations, both in IL [Sauer 2018; C. Chen 2015; Al-Qizwini 2017; Mehta 2018] and RL [Toromanoff 2019;

D. Li 2019]. They can be classified in two categories: implicit or explicit. When they are explicit, affordances usually are features specific to driving (distance from the previous car, center of the road, etc.). These affordances allow more explicit learning, by reducing the black-box effect of neural networks. On the other hand, these are made by and for humans, and their choice can bias the agent. Indeed, is it useful to predict that the car in front is at 5.5m, or is it sufficient to say that there is a car in front and to know if it is rather far or rather close? Intermediate representations are in a way implicit affordances, the network is trained for a certain task (semantic segmentation for example, or reconstruction of the original image using a VAE), and it is the intermediate representation that is used by the agent for autonomous driving. There are several ways to train these affordances / intermediate representations. They can be trained at the same time as the driving task, so they are called auxiliary tasks, like in [Mehta 2018; Hawke 2019]. Affordances can also be trained separately, in a supervised way, which represents the large majority of cases [Z. Li 2018; D. Li 2019; J. Chen 2019; Toromanoff 2019; Sauer 2018; C. Chen 2015; Al-Qizwini 2017]. They are then used by a PID controller, or as input for a driving agent. This allows to reduce the state space, especially when learning to drive is done in RL [Toromanoff 2019]. However, we believe that decoupling perception and control may reduce the performance and robustness of the driving agent, since there is no direct mapping between image perception and driving control. Used in RL for navigation in maze [Jaderberg 2016; Mirowski 2016], end-to-end training using auxiliary tasks seems to us to be a promising trail, still little exploited in a complex task such as autonomous driving with RL in an urban environment. Auxiliary tasks allow to train an agent using end-to-end while guiding the training. Other ideas have been considered to assist reinforcement learning, all of which are presented in the appendix A.

In this thesis, we will focus on reinforcement learning with raw image input and continuous actions for conditional urban driving. The choice of continuous action was made to bring ourselves closer to reality and not to depend on a given discretization. However, in view of the difficulty of the driving task, we will study methods to accelerate learning and increase its performance, in particular by adding auxiliary tasks. Because of the trial and error essence of reinforcement learning, its need for interaction with the environment and therefore for obvious safety reasons, our work will be done in simulation. We decided to work with Carla simulator [Dosovitskiy, Ros, 2017]. Indeed Carla is an urban driving environment with many possible sensors as well as a certain variability of the environments. Moreover Carla proposes an autopilot mode, and is easy to handle. Finally Carla offers two evaluation benchmarks (CoRL and NoCrash) on which more and more researchers compare their algorithms, and also proposes the Carla challenge [*CARLA Autonomous Driving Challenge* 2019], where complex driving situations are evaluated (handling intersections, overtaking, multi lane driving, etc...). Carla also benefits from a large and growing community (some of which implement their own maps, such as interactive traffic scenarios in [Błażej 2020]). In Table 3.2 we summarize the major approaches from the state of the art, and their results on Carla CoRL benchmark. It is important to note that very few approaches use reinforcement learning, and none of them perform end-to-end RL with RGB visual input. Some of the work presented in the table 3.2 is also done concomitantly with the work presented in this thesis.

Training	Details	Navigation				Dynamic Navigation			
		Training conditions	New weather	New town	New town & weather	Training conditions	New weather	New town	New town & weather
MP [Dosovitskiy, Ros, 2017]	Modular Pipeline	80	94	24	47	77	89	24	44
RL [Dosovitskiy, Ros, 2017]	End-to-End RL using A3C	14	2	3	6	7	2	2	4
CIL [Codevilla 2017]	End-to-End IL	86	84	40	44	83	82	38	42
CIRL [Liang 2018]	IL followed by RL finetuning with A3C	93	86	53	68	82	80	41	62
CILRS [Codevilla 2017]	CIL + speed prediction	95	96	69	92	92	96	66	90
MT [Z. Li 2018]	Multi-task IL that predicts semantic segmentation and depth	81	88	72	78	81	80	53	62
CAL [Sauer 2018]	Supervised training of affordances (distance to vehicle, red traffic light, ...) fed to a controller	92	90	70	68	83	82	64	64
ILA [Cultrera 2020]	End-to-End IL with attention module	91	92	53	67	89	92	40	56
LSD [Ohn-bar 2020]	Situational Driving: supervised mixture of models + context embedding + CMA-ES finetuning			99	**100**			98	98
WRL [Agarwal 2019]	Waypoint based DRL: birdview segmented image input + waypoints to indicate direction + speed PID controller	99	99	94	94	79	79	60	60
LBC [D. Chen 2020]	Learning by Cheating: two stages training with privileged agent having access to environmental layout, and teaching a vision-based agent	**100**	**100**	98	**100**	**100**	96	99	**100**
MaRLn [Toromanoff 2019]	RL. Encoder pretrained in a supervised manner with affordances (segmentation, distance to crossing, traffic lights states, ...) and fixed during navigation RL training (DQN)	**100**	**100**	**100**	**100**	**100**	**100**	98	**100**

Table 3.2: Summary of state of the art approaches results on Carla CoRL benchmark - navigation and dynamic navigation tasks.

Chapter 4

End-to-End Autonomous Driving on Circuit with Reinforcement Learning

Contents

4.1 Introduction

The purpose of this chapter is to highlight some of the parameters necessary for the convergence of a reinforcement learning algorithm in the case of autonomous driving. The objective here is to show that an agent is able to drive with the help of a RL algorithm using continuous actions. Reinforcement Learning algorithms are known to be extremely sensitive to hyperparameters, this is why we decided to first place ourselves in a simpler environment than the complete CARLA maps. Working in a simpler environment allowed us to make first choices of global design of our framework. The work in this chapter therefore focuses on the lane following task, conditional navigation (with intersections) will be added in the next chapter. We will therefore deal with the network model, the inputs and outputs, the design of the reward function as well as the hyper parameters that allow a RL algorithm to converge. The objective is first to understand the influence of the different parameters, justify the choices we have made, and detail them in order

to ensure the reproducibility of our work. The main contribution of this chapter is therefore to detail the different elements that play a key role and their influence on the performance of the driving agent. First we will present the simplified environment we use. It is a circuit in which we will navigate in optimal conditions, i.e. weather with good visibility. It is in this environment that we will study the different parameters essential to the good convergence of PPO for autonomous driving. Later we will add some difficulties, such as different weather conditions, data augmentation, and some obstacles.

4.2 Background

In a first part we will summarize the state of the art from Chapter 3 concerning the lane following task from visual input with reinforcement learning. We will take the opportunity to detail the reward functions used for autonomous driving in RL. We will then insist on the importance of the choice of hyperparameters, and on some good practices. Finally we will present some upstream choices, such as the choice of a framework.

4.2.1 Lane Following with Reinforcement Learning

Kendall *et al.* are among the first to take an interest in real-world driving with reinforcement learning in *Learning to drive in a day* [Kendall 2018]. Their approach combines training an autoencoder that reconstructs the image to reduce the state space, and learning to drive. The work is done on a real vehicle, with a driver who regains control if the car goes off the road - which ends the episode. The algorithm used is DDPG. This work showed for the first time that it is possible to use reinforcement learning for autonomous driving in real environment. However, the speed limit is set at 10km/h, and to explore higher speeds, a lot of work is done in simulation. Wolf *et al.* in [Wolf 2017] use DQN, with discrete actions, for driving in a fully simulated circuit at constant speed. The input images are however very simple, a mask with the lines of the road. To handle the more complex images of Torcs simulator, Li *et al.* propose a two-step training in *Reinforcement Learning and Deep Learning Based Lateral Control for Autonomous Driving* [D. Li 2019]. The first step is to extract track features from the image: distance to lane marking, the heading angle difference, and track heading - which indicates whether the road is straight or whether it turns left or right. The network for extracting these features is trained in a supervised way. Then these features are used as input for a RL agent that computes the steering wheel angle using DPG. Like [Wolf 2017], the speed is also imposed, and varies between 60 and 75km/h. Other works also manage the speed control, in addition to the steering wheel angle. This is the case in both [Perot 2017; Jaritz 2018]. These two approaches involve a small CNN (containing 3 layers of convolution) with LSTM, with A3C and discrete actions, for racing driving on WC. Finally in *Virtual to Real Reinforcement Learning*[X. Pan 2017] an agent is trained on realistic images to allow transfer to the real world. A translation network is trained using a GAN to transform simulation images - Torcs - into realistic images (through semantic image segmentation). These realistic images are used as input/state for the RL agent which is trained with A3C.

Overview of Reward Functions

We present in table 4.1 a brief overview of the rewards function used in lane following, as well as the reward used in original Carla work [Dosovitskiy, Ros, 2017] - since we are using this simulator and we want to move to urban driving afterwards, and also because this reward can also be applied to lane following.

Table 4.1: Summary of reward functions used for lane following with reinforcement learning

Reward	Details	Use	Ref
$r =$distance driven before the safety driver takes control back	Speed limited to 10 km/h	Driving in real world	[Kendall 2018]
$r = \begin{cases} (v \cdot cos(\alpha) - d) \cdot 0.006 \\ -0.025 \quad \text{when collision} \end{cases}$	• v: current speed • α: angle between vehicle & trajectory • d: lateral distance to road center	Racing game, control of speed and angle	[X. Pan 2017]
$r = v \cdot (cos(\alpha) - d)$	Same notation as above	Racing game	[Jaritz 2018]
$r = max(r_{dist}, -2) + r_{action}$ *with* $r_{dist} = 2 - (\lvert d \rvert + 1)^6$ *and* $r_{action} = \begin{cases} +1 & \text{when reducing the angle } \alpha \text{ between the road and the agent's direction} \\ -1 & \text{when increasing } \alpha \text{ or counter-steering in curves} \\ 0 & \text{else} \end{cases}$	Same notations: • d: lateral distance • α: angle	Simulated circuit, agent's speed is constant	[Wolf 2017]
$r = \begin{cases} cos(\alpha) - \lambda sin(\lvert\alpha\rvert) - d/w & \text{if } \lvert\alpha\rvert < \pi/2 \\ -2 & \text{otherwise} \end{cases}$	• α: angle between the vehicle and the trajectory • λ: parameter to adjust the lateral or horizontal influence of α	Racing game, agent only focusing on lateral control.	[D. Li 2019]
$r =1000(d_{t-1} - d_t) + 0.05(v_t - v_{t-1})$ $- 0.00002(c_t - c_{t-1}) - 2(s_t - s_{t-1})$ $- 2(o_t - o_{t-1})$	• d: traveled distance • v: speed • c: collision, • s : intersection with the sidewalk • o: intersection with opposite lane	Urban simulated driving	[Dosovitskiy, Ros, 2017]

The works on lane following presented above have either the objective of driving as fast as possible, or the speed is controlled independently and the agent focuses only on lateral control. We will present in detail the rewards for conditional driving in the next chapter, however it is interesting to note that most of the rewards for urban driving are additive, while they are sometimes multiplicative for race driving. However, in spite of a certain disparity in the formulas of the rewards functions, we generally find the same components:

- Speed and/or traveled distance
- Deviation from the center of the road
- Angle between vehicle's direction and desired trajectory.
- Punishment in case of collision / offroad / countersteering

4.2.2 Hyperparameters Importance & Best Practice

Replication of results is essential to allow for comparison and improvement. Both papers [Henderson 2018; Islam 2017] investigate the reproducibility of results in reinforcement learning. These two works highlight the variance that exists between different identical runs of the same algorithm. In the Figure 4.1 we can see the results of 10 runs of the TRPO on the HalfCheetah-v1 environment, separated in two sets of 5 runs. We can see the variance that exists between identical runs where the only variation is the random seed. Usually the results presented in the literature show the best results obtained, or an average of a small number of trainings.

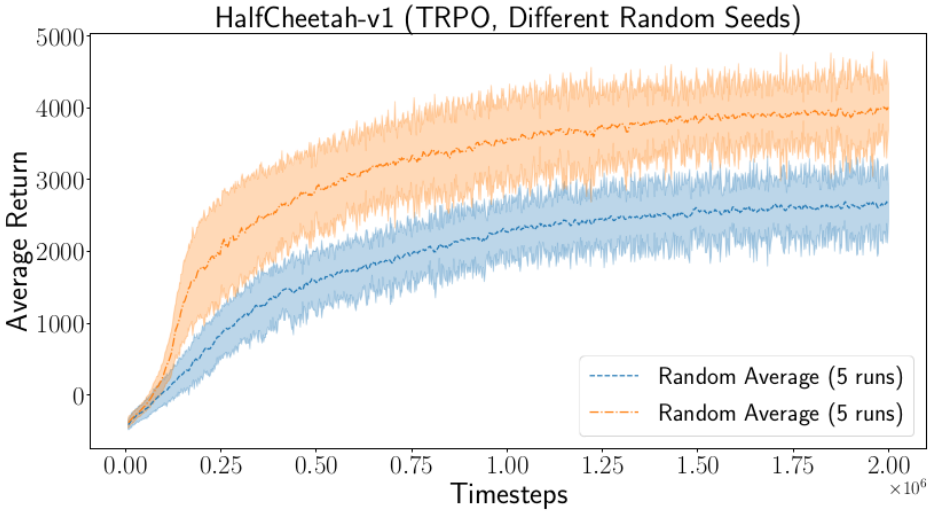


Figure 4.1: Average return on HalfCheetah environment of 10 identical trainings with different random seeds

The results also vary greatly depending on the network policy and activation features. The activation functions that seem to perform best are the relu and leaky relu. The importance of reward scaling is also highlighted in [Henderson 2018], and a badly chosen scaling can prevent convergence. Several techniques can be found in literature:

- Rescale reward with a factor: $r_{agent} = \sigma \times r$ (usually $\sigma = 0.1$) like in [Duan 2016] or [Gu 2016]

- Clip reward between $[-1, 1]$ like in [Hessel 2018]
- Rescale reward between $[-1, 1]$
- Log rescaling $r_{agent} = sign(r) \cdot log(1 + |r|)$ [Hester 2017]

Figure 4.2: Comparison between log rescaling (red: $y = sign(r) \cdot log(1 + |r|)$), rescaling factor (blue: $y = 0.1 \times x$), and clipping (green: $y = \max(-1, \min(x, 1))$

In *What Matters for On-Policy Deep Actor-Critic Methods? A Large-Scale Study* [Andrychowicz 2021] is performed an extensive study on many hyperparameters and different aspects of training - losses, network architecture, optimizers... The authors then give different recommendations. We note among other things that the optimizer Adam is a good choice, that it is absolutely necessary to normalize the observation, and that using GAE (Generalized Advantage Estimation) [Schulman, Moritz, 2015] for advantage estimation works well. There are also many parameters that need to be tuned (learning rate or discount factor) depending on the environment, but this work gives a starting point.

Finally we will present two very important tricks from *DeepMimic: Example-Guided Deep Reinforcement Learning of Physics-Based Character Skills* [Peng, Abbeel, 2018] that make RL algorithm work: the initial state distribution, and early termination. First, not initializing the agent at the same initial state allows to explore more, and probably getting to a rewarding state quicker. It will also make the agent learn every part of the task independently. Indeed, in the case of DeepMimic, the initial state is sampled in the reference motion. So if the task is to learn how to do a backflip, the agent will learn to jump and land at the same time, as it may be initialized when the agent is already in the air. If the initialization was only a fixed state, it would have learned first to jump, but it would have needed to be able to land before getting rewards. So having a good initial state distribution can be necessary to make the agent learn. Early termination also enables to accelerate the learning process. The goal is to stop the episode as soon as the agent is put in a too bad situation (lying on the ground for instance for human-shaped agent, or stuck against a wall). By doing so, it avoids having a collection of useless data, and the learning will focus on better data. This early termination in used in *Learning to drive in a day* [Kendall 2018]. Episode ends as soon as the car starts getting off the track, to avoid exploring useless places.

4.2.3 Upstream Choices

We presented in chapter 2 the main reinforcement learning algorithms. As mentioned, we chose the PPO algorithm for several reasons. First PPO is an actor-critic algorithm that allows the use of continuous actions. It is simple to implement, and allows the sharing of parameters between the policy and the value function, as well as with potential auxiliary tasks. PPO also has high performance on Atari games, and in particular far surpasses A3C and ACER on the Enduro game, the Atari game that is closest to autonomous driving.

We chose not to implement it ourselves and to use a framework. Reinforcement learning algorithms are sensible to the implementation. In *Deep Reinforcement Learning that Matters*, the

authors compared the results of different implementation of the same RL algorithms and noticed huge variance in results. That is why we wanted to work with a tried-and-tested implementation. We chose stable baselines [Hill 2018], a fork of Openai Baseline [Dhariwal 2017], the latter being no longer maintained. Stable baselines presents a unified interface for all RL algorithms. It also contains additional optimizations for PPO that do not appear in original paper. These additional optimization - like value function clipping, or advantage normalization - may have a major impact on the performance [Engstrom 2019]. More recently, Tensorflow proposes its own interface, TF agent [Guadarrama 2018], and we can also cite rllab [Duan 2016], which is not maintained anymore but refactored by garage [contributors 2019].

Finally, we chose to work in simulation with the Carla simulator. We chose to use the best possible image quality (two choices are possible, Low or Epic quality), and to work at 20 FPS. There is a wide variation in the choice of frame per second in the literature, from low frequency (10 FPS for [Toromanoff 2019; D. Chen 2020], 12 in [Rausch 2017]), to intermediate (30 FPS in [Bojarski 2016]) or even high frequency for race and high speed driving (60 FPS [Y. Pan 2017]). At the moment, it is not possible to use the Carla simulator at less than 10 FPS[1] . The physic engine is clamped at a minimum of 10FPS, so if the simulator is set to less, it is no longer synchronized with the physic engine, and therefore the simulator measurements become inaccurate or even false ($speed \neq \frac{distance}{time}$). We are not in the case of a speed race, but focus on urban - or maybe highway - driving. The speed will therefore be between 0 and 100km/h approximately. 50 km/h corresponds to about 13.8m/s, and 100km/h at 27.7m/s, so at 10 FPS, when the vehicle drives at 50 km/h, there is 1.4m between each frame, and almost 3m at 100km/h, which seems a bit too much. We decided to divide these figures by two, and therefore to work at 20 FPS.

4.3 Experiments

4.3.1 Environment

Driving well is a very complex task. When we drive, we follow a certain path, we pay attention to others - vehicles, pedestrians,... and we comply with traffic regulations. This leads to an ensemble of tasks that are very difficult to implement. Therefore we have split the tasks, starting with simple driving without obstacles, and we gradually increase difficulty (adding fixed and moving obstacles, intersection, navigation with a given path, ...). The first objective is then to measure how well the car is able to follow a road, without moving obstacles (like other cars or pedestrians), and no crossing. To do so we built our own map (see Figure 4.3) which is a closed-loop circuit, with one lane in each direction. We designed our map so that there are various environment conditions: straight road, sharp turns, trees on the side of the road, buildings, ... We integrated this map into Carla simulator to run our experiments. The first step consisted in training our agent to follow its lane, and drive at correct speed. The creation of new maps - with RoadRunner software - and their integration into the simulator is a tedious work, that is why only one training circuit was built (we will see in chapter 6 the second circuit created to study generalization). However, it was important to start with a circuit in order to validate the training setup in a simpler environment, especially without crossings management, which we will study in the next chapter.

[1] `https://github.com/carla-simulator/carla/issues/695`

Figure 4.3: Training circuit

4.3.2 Model

In this section we will first present the neural network used in a detailed manner, then the hyperparameters that have proven to be necessary. The network used is based on a Resnet34 [He 2016]. The Resnet18 and Resnet34 networks are classically used for the perception module for autonomous driving [Toromanoff 2019; D. Chen 2020; Hecker 2019], and the work [Codevilla 2018] have shown good performances for Resnet 18 and 34, and a drop in performance when switching to Resnet50. Figure 4.4 shows the neural network used by the PPO algorithm. The inputs are an image $I_t \in \mathbb{R}^{W*H*C}$, and a vector containing the measurements - current speed and current steering angle. The network outputs both action and value - since PPO is an actor-critic. Details are given in the input/output section 4.3.4. Table 4.2 and Figure 4.5 show details on the Resnet34 architecture. The activation functions used are ReLu.

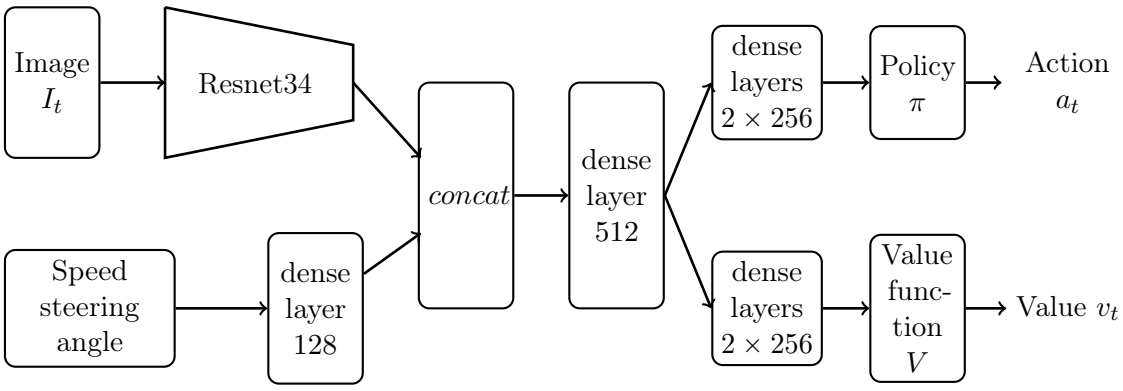


Figure 4.4: Reinforcement Learning Architecture

	Output size		
Input	[64,192]		
First layers	[32,96] [16,48]	conv[7*7], 64, stride 2 maxpool[3*3], 64, stride 2	
Encoder block 1	[16,48]	$3 \times \begin{cases} conv[3*3], 64 \\ conv[3*3], 64 \end{cases}$	no strides
Encoder block 2	[8,24]	$4 \times \begin{cases} conv[3*3], 128 \\ conv[3*3], 128 \end{cases}$	stride 2 on first conv
Encoder block 3	[4,12]	$6 \times \begin{cases} conv[3*3], 256 \\ conv[3*3], 256 \end{cases}$	stride 2 on first conv
Encoder block 4	[2,6]	$3 \times \begin{cases} conv[3*3], 512 \\ conv[3*3], 512 \end{cases}$	stride 2 on first conv

Table 4.2: Resnet34 as Encoder

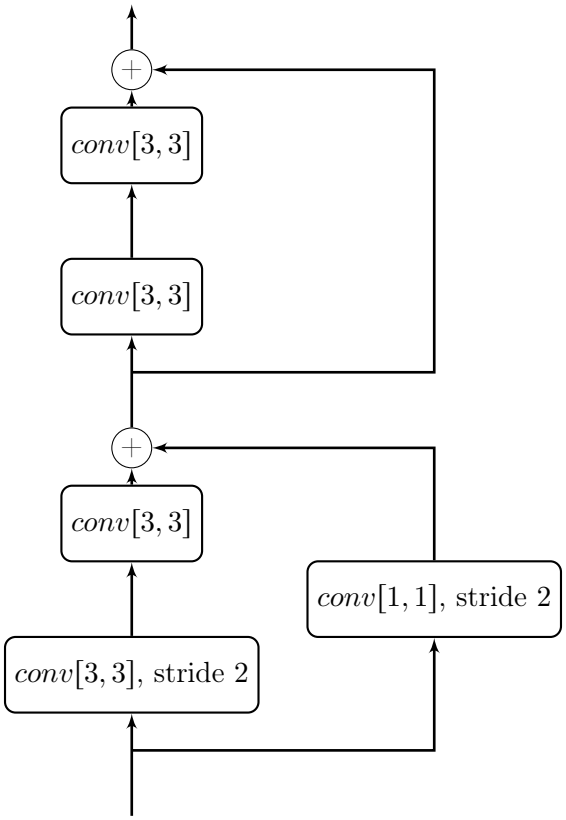


Figure 4.5: Encoder block architecture for block 2,3,4. Encoder block 1 has no strides so residual are both simple additions

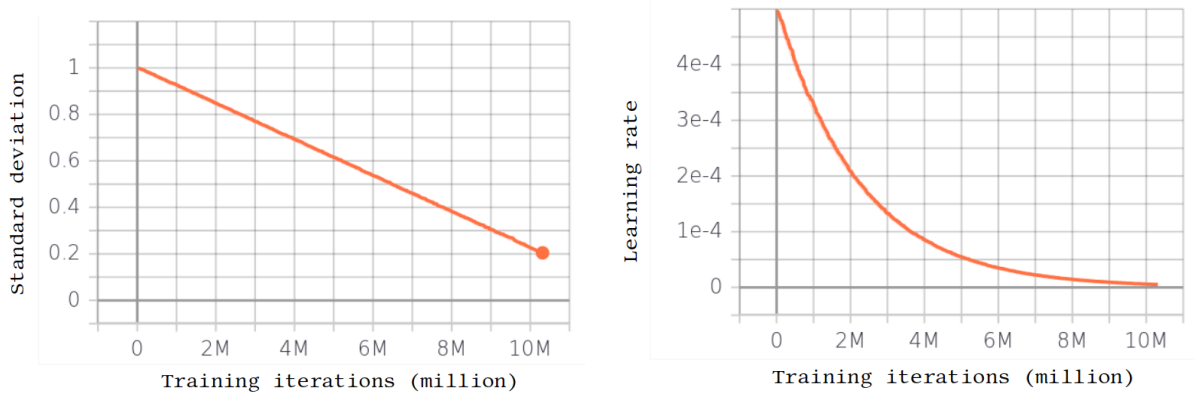
Hyperparameters

Once the algorithm and the network have been chosen, the hyperparameters must be set. We based ourselves on the hyperparameters given in the original PPO paper [Schulman 2017], in particular

for the discount factor γ set to 0.99, the clipping parameter $\epsilon = 0.2$ and the value function loss weight $c_1 = 0.5$ (see equation 2.58 for full PPO loss). Since PPO is an on-policy algorithm, it alternates between data collection, and policy/value function update with backpropagation. So at every iteration, a batch of steps is collected with all the actors running. To adjust the batchsize, we performed some quick tests on the Enduro environment. We found, all other things being equal, that the learning converged for a batch size of 1024, but not for 512 or 128, therefore we set the batchsize to 1024. We had 8 simulators running in parallel to collect data , so each one collected $T = 128$ training steps. Once this batch is collected, update takes place. The whole batch is shuffled to avoid correlated data, and split in mini-batch for backpropagation. We chose the minibatch size of 128 - so the batch is split in $M = 8$, and the batch will be gone through 4 times (number of epoch = 4). The policy network outputs μ which is the mean of a Gaussian distribution used to compute the action a_t:

$$a_t \sim \mathcal{N}(\mu(t), \sigma(t)) \tag{4.1}$$

We decided to set the variance σ linearly decreasing from 1 to 0.2 over training, but it can also be learned as an output of the policy (see Figure 4.6a). Variance decay enables high exploration of the agent at the beginning of the learning process, and a controlled decrease of this exploration during training. Training is realized with Adam optimizer and a exponentially decreasing learning rate, from $5 \cdot 10^{-4}$ to $5 \cdot 10^{-6}$ over training (see Figure 4.6b).



(a) Standard deviation

(b) Learning rate

Figure 4.6: Evolution of standard deviation and learning rate over training

Table 4.3.2 summarizes the principal hyperparameters and their values, using notations from [Schulman 2017] ,

hyperparameter	value
clipping parameter (ϵ)	0.2
discount factor γ	0.99
value function loss weight c_1	0.5
GAE parameter λ	0.95
nb of environments N	8
nb of minibatch M	8
nb of collected data per environment T	128
nb epoch	4

4.3.3 Training and Evaluation Benchmark

Since we created our own environment, the benchmark presented by CARLA authors cannot be applied [Dosovitskiy, Ros, 2017]. We were inspired by it to create our own benchmark that measures how well the car is driving. Relevant measurements are the following: average speed during one episode, traveled distance in one episode (as there is no ending point, episode ends when car crashed or with time-out), average cross-track error (*cte*), average angle between car and road (*atr*) (this will check if the car oscillates or is aligned with the road), and the number of episode that end with a collision. Evaluation will be made with the same conditions as training (same circuit, same weather condition). We will explore generalization on chapter 6, on both circuit and full town. The circuit length is around 2.5km, so we chose 6 starting points around the circuit (3 in each direction), and run episodes of maximum 2000 steps, which amounts to about 1km at 40km/h and 20 FPS. So these 6 episodes roughly cover the whole circuit. On Figure 4.7 we show the evolution of discounted reward during training over 10 Million steps. Under the simplest conditions we work with - easy circuit and weather, the trainings are stable and all converge in the same way, so we present only one learning curve. However, in more complex conditions (obstacles and difficult weather at the end of this chapter, and with conditional navigation in the next chapter), we will see that identical trainings have different performances. Table 5.4 show the results of our baseline training with optimal parameters on our benchmark.

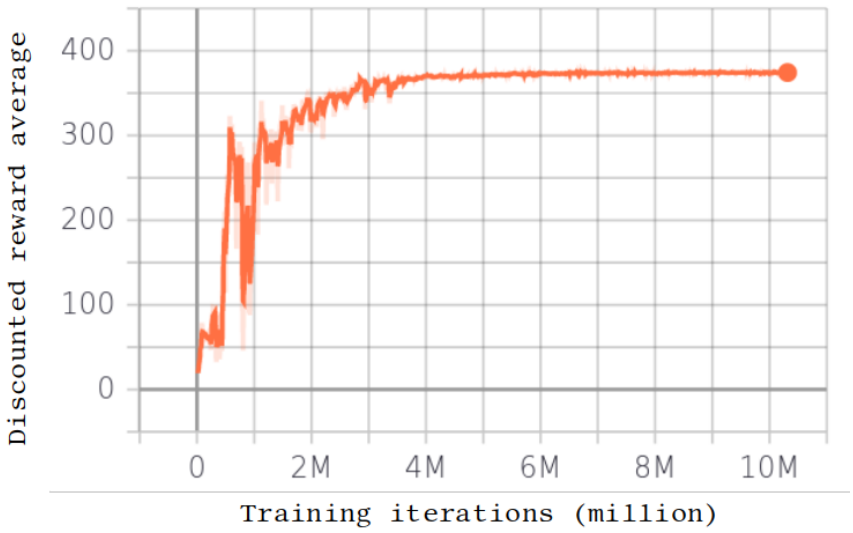


Figure 4.7: Discounted reward during training

	Baseline
speed km/h	38.40 (4.52)
cte m	0.05 (0.06))
atr °	0.98 (1.00)
distance m	1066.55 (6.19)
collision	0/6

Table 4.3: Result of our baseline training on our benchmark. For every measurement is shown the mean and variance over the test episodes.

4.3.4 Input/Output

In this section we detail the input/output of our reinforcement learning training.

Input

A compromise had to be found between a large image that makes it possible to clearly distinguish the important elements - crossroads, red lights, traffic signs - and a small image that allows large batches - and therefore faster and more efficient training. We chose an image size of $192 \times 64 \times 3$, of which an example can be seen on Figure 4.8. This image size is close to the one from original CARLA work [Dosovitskiy, Ros, 2017] with RGB image of size 288×88, but rather small compared to recent state of the art work (LBC [D. Chen 2020] use 384×160 RGB image and MaRLn [Toromanoff 2019] use 4 RGB images of size 288×288).

Figure 4.8: Example of input image - here on Town01. Size is $192 \times 64 \times 3$

The two other inputs are measurements that indicate the current state of the vehicle: current speed and current wheel angle. All inputs are scaled: RGB image is scaled between 0 and 1, and wheel angle is already $\in [-1, 1]$. Current speed is in km/h, and is scaled (by 0.01) during training. Figure 4.9 shows the comparison of the discounted reward during training with and without speed scaling. Even if both training converge, speed scaling - as well as all other input scaling - speeds up the convergence.

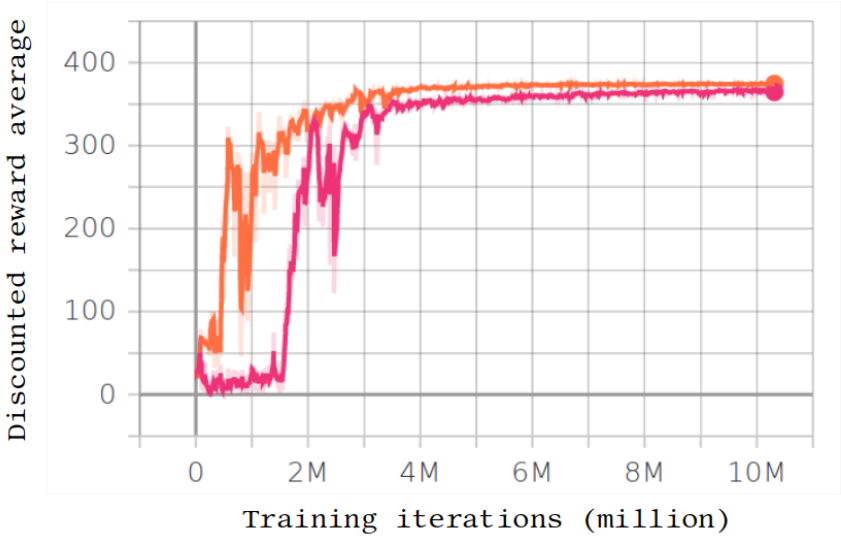


Figure 4.9: Comparison of discounted reward between training with speed normalization (orange) and without (pink).

Output

In Carla simulator, the necessary driving controls are acceleration $\in [0, 1]$, brake $\in [0, 1]$, and steering wheel angle $\in [-1, 1]$. In the literature there are two ways to represent acceleration and brake values: one common output, or two separate outputs. However, as it is not possible to accelerate and brake at the same time, we considered it simpler to have a single output for speed management. We thus combined acceleration and braking into a single output: acceleration/brake $\in [-1, 1]$ (positive to accelerate, and negative to brake).

The angle of the steering wheel can vary from -1 to 1 (which roughly corresponds to -90 to +90°). Regarding the steering wheel angle, we have found that if the output is directly the angle, the latter can vary enormously between two iterations. This causes several problems. The first one is that it is not close to reality, the steering wheel cannot turn completely in one twentieth of a second. The second one is that it produces oscillations of the vehicles if the steering wheel varies too much between two iterations. We have compared in Table 4.4 the results of two trainings, one where the output is directly the wheel angle, and the second where the output is the delta of the wheel angle. The difference on the benchmark is not huge but we can see that the angle of the car in relation to the road is higher. But above all, by observing the behavior of the agent, we can see that it oscillates much more, with a much less comfortable behavior for the passengers.

We have added a parameter, which we call the hardness of the steering wheel, which indicates how much the steering wheel can move at the most with each iteration. We have estimated that the steering wheel can rotate from one side to the other in 2 second. Therefore at 20 FPS, 40 frames are necessary for the steering wheel to go from -1 to 1, the steering wheel can then rotate to a maximum of $2/40 = 0.05$ per step. The steering wheel angle is then:

$$\theta_{t+1} = \theta_t + 0.05 \times \Delta\theta. \tag{4.2}$$

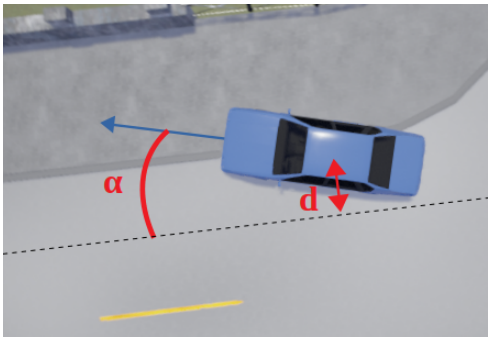
	Baseline with $\Delta\theta$ as output	Training with θ as output
speed km/h	38.40 (4.52)	39.73 (5.08)
cte m	**0.05 (0.06)**	0.09 (0.08)
atr °	**0.98 (1.00)**	1.88 (1.65)
distance m	1066.55 (6.19)	1103.67 (5.92)
collision	0/6	0/6

Table 4.4: Comparison of performance on our benchmark between training with steering angle θ as output and training with the delta of the steering angle $\Delta\theta$ as output.

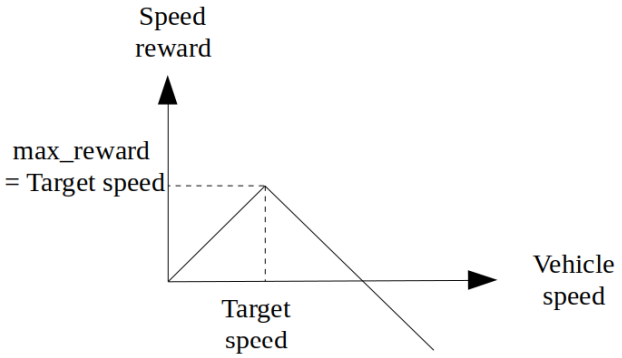
4.3.5 Design of Reward Function

As the reward is the only information the agent obtains from the environment, it is a very important component in reinforcement learning algorithm. As we have seen in the section 4.2.1, there is a wide variety of rewards. We have chosen an additive reward, we will see in the next chapter that these are the most used for city driving which is our final goal.

The reward function used is a weighted sum of several terms: the gap between current speed s_t and objective speed s_{obj} (in km/h), the cross-track error (distance to road center) cte (in m), and the angle between the car and the road atr (in degrees). There is also a negative reward when the car collides with an obstacle or runs off the track.



(a) Illustration of cross-track error d and angle α used to compute the reward



(b) Illustration of speed reward

Figure 4.10

$$r_{=} \begin{cases} w_s \times r_{speed} + w_{cte} \times r_{cte} + w_a \times r_{angle} \\ r_{punish} \text{ when collision or offroad} \end{cases} \tag{4.3}$$

with

- $r_{speed} = target\_speed - |target\_speed - speed|$): computes the gap between target speed and current speed. The target speed is set to 40 km/h
- $r_{cte} = (1 - d)$ with d the cross track error, i.e. the lateral distance of our agent to road center
- $r_{angle} = (15 - |\alpha|)$ with α the angle between the direction of the agent and the trajectory.
- w_s, w_{cte} and w_a their associated weights, respectively 1.0, 10 and 0.1, determined experimentally to balance the different terms of the reward.
- $r_{punish} = -100$

Although we have not studied in details the influence of weights on the agent's behavior, we have observed a strong influence of the reward sign. Constants are added in r_{cte} and r_{angle} so that the corresponding reward is positive if the car is roughly aligned with the road (i.e. $\alpha < 15$ °) and not too far from road center ($d < 1m$). Figure 4.11 shows the discounted reward with and without these constants. On the red curve the constants were set to zeros, and we can observe training takes much longer to stabilize. In Table 4.5 we compare the results of the training with mostly positive reward and the training with negative reward - that differ only from two constants in reward function. The results at 10 Million steps are comparable, even if we observe a much higher variance when the reward is negative. At 7 millions steps, it is obvious that the training with the negative reward is still learning, whereas the baseline is already stable. It is also interesting to note that the baseline at 7 million has a higher speed, because our agent learns first how to accelerate, and later it learns to brake when it is able to drive around 40km/h.

Figure 4.11: Comparison of discounted reward during training between mostly positive (orange) and negative (red) rewards

	Baseline 10 M	Baseline 7 M	Neg reward 10M	Neg rewards 7M
speed km/h	**38.40 (4.52)**	**40.71 (4.73)**	35.85 (13.65)	19.54 (24.81)
cte m	**0.05 (0.06)**	**0.06 (0.05)**	0.15 (0.18)	4.45 (9.94)
atr °	**0.98 (1.00)**	**0.92 (0.95)**	3.18 (5.56)	33.03 (36.15)
distance m	**1066 (6.19)**	**1130.83 (5.92)**	995.93 (262.76)	450.75 (279.15)
collision	**0/6**	**0/6**	**0/6**	2/6

Table 4.5: Comparison of performance evolution between positive and negative rewards

We also scale our reward during training. We tested the four options presented in Section 4.2.2, namely scaling factor (we tested $\sigma = 0.1$), reward clipping in $[-1, 1]$, reward rescaling between $[-1, 1]$ and log rescaling: $r_{scale} = sign(r) \cdot log(1 + |r|)$. With no normalization, our training could not converge. We tested these four options, and both rescaling with 0.1 factor and log rescaling gave satisfactory results, whereas clipping or rescaling between $[-1, 1]$ did not work out well. We decided to use log rescaling since (even if it flattens bigger values).

4.3.6 State Initialization, Early Termination and Partial-Episode Bootstrapping

We applied both *reference state initialization* and *early termination* from [Peng, Abbeel, 2018] we presented in Background (see Section 4.2.2), adjusted to our problem. First we will sample initial states around the whole circuit, the car won't start from the same starting point. Moreover, the agent will not be placed always at the center of its lane: we shift it randomly so that it is not in the middle, and we also rotate it slightly (between -30 and 30 degrees). This way, we encourage the agent to first learn how to correct its trajectory to follow the road. The termination will occur if the car collides, but also if is goes off the road - and also in case of a timeout.

We also added partial-episode bootstrapping from [Pardo 2018], which consists in modifying the reward for a terminal state if the termination is due to time limit (see eq. 4.4). Indeed for the lane following task, as we work in a closed loop circuit, there is in theory no limit for the driving,

and a timeout is only set to diversify experiences. Therefore, reaching this limit should not be penalized and we give the expected long term return instead of the actual reward to compensate.

$$r_{terminal\_state} = \begin{cases} r & \text{at environmental termination} \\ r + \gamma \times value(s') & \text{at timeout} \end{cases} \tag{4.4}$$

Figure 4.12 shows the comparison of the value loss with and without using Partial Episode Bootstrap, and we can observe that convergence of value loss is much better when using Partial Episode Bootstrap.

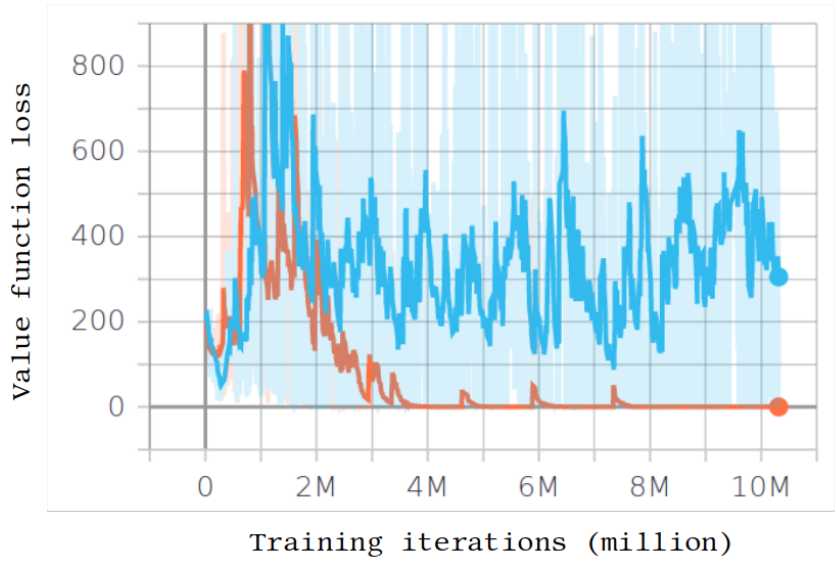


Figure 4.12: Comparison of value loss between with (orange) and without (blue) partial episode bootstrap

The influence on the performance of partial episode bootstrap on the circuit was not visible on the car behavior, but we did notice a huge effect when working in town.

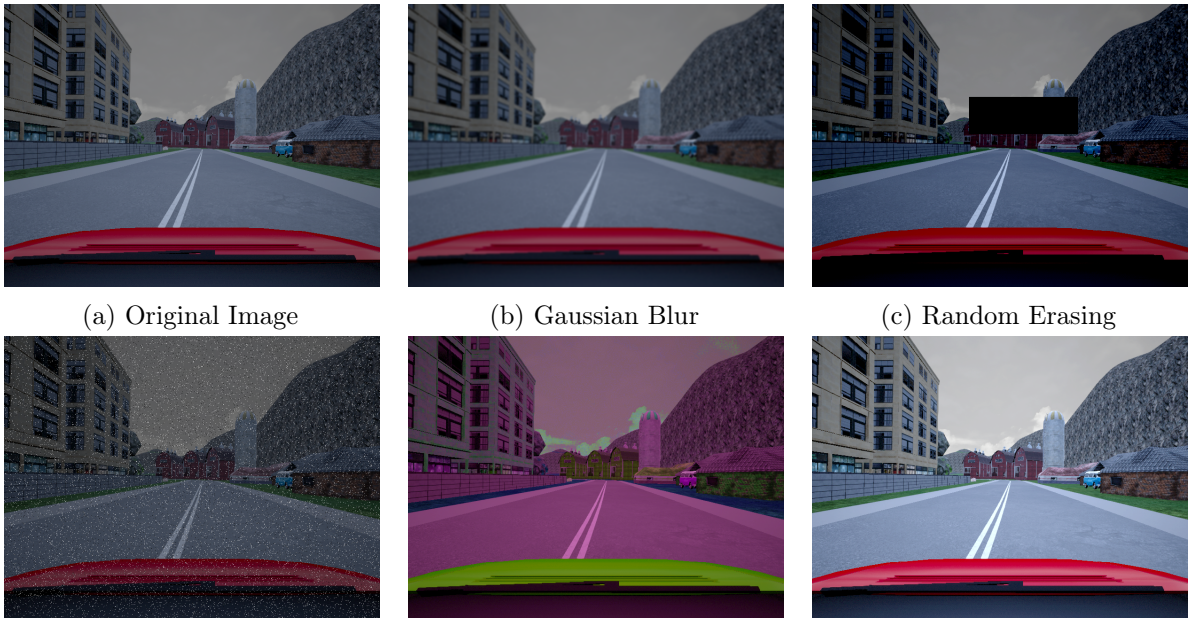
4.3.7 Increased Complexity: Diverse Weathers and Bad Parked Cars

We have previously placed ourselves in the simplest possible conditions for hyperparameters tuning - ClearNoon preset. We are going to increase the difficulty in this section, both by training with different weather conditions and adding data augmentation, and by adding fixed obstacles on the trajectory of our agent.

Different Weathers and Data Augmentation

To add robustness, diverse weathers and times of the day are used during training, as well as data augmentation on images, as commonly used in supervised deep learning. Data augmentation includes random variation in hue and saturation , brightness and contrast, along with conversion to gray scale, partial erasing of the image salt and pepper noise, and Gaussian blur (see Figure 6.3). We used the python library imgaug [Jung 2018] to augment the images. The data augmentation is applied to each image independently, so it changes along an episode. We also randomly changed the color on semantic element of the image during training, we will see on the chapter on generalization (Chap 6) that it improves generalization capabilities. Our benchmark is extended

to add new weathers. We used 6 training and 6 test weathers, and each episode will be run with each weather, for a total of 36 training episodes and 36 test episodes.



(a) Original Image (b) Gaussian Blur (c) Random Erasing

(d) Salt and Pepper Noise (e) Changes in hue / saturation (f) Change in brightness

Figure 4.13: Example of data augmentation

Addition of Bad Parked Cars

To teach our agent to avoid obstacles, we randomly placed cars that overflow slightly on the road, like badly-parked cars, and trained the model from scratch. The cars are placed so that our agent meets them regularly (about 3 or 4 per kilometer, but their placement and occurrence is random). We present on Table 4.6 the results on training conditions with and without obstacles. We can see that our agent drives perfectly when there are no obstacles, in both training conditions and new weathers, but that is collides more often in unseen weather conditions when we add bad parked cars. We can see that the agent is less efficient when the difficulty of the task increases (addition of obstacles), but also that it has difficulties to generalize (with different weather conditions).

	Training condition		New weather	
	With cars	No cars	With cars	No Cars
speed km/h	36.4 (7.8)	37.3 (6.3)	36.9 (6.3)	36.2 (8.8)
cte m	0.2 (0.3)	0.1 (0.1)	0.2 (0.3)	0.1 (0.2)
atr °	1.7 (1.5)	1.1 (0.2)	1.7 (1.3)	1.0 (0.3)
distance m	978.7 (274.3)	1035.3 (175.0)	881.1 (332.7)	1005.2 (243.8)
step	1900.7 (399.3)	2000.0 (0.0)	1724.4 (545.1)	2000.0 (0.0)
collision	3/36	0/36	10/36	0/36

Table 4.6: Performance of training with bad parked cars as obstacles: results with and without obstacles, on training and new weather conditions

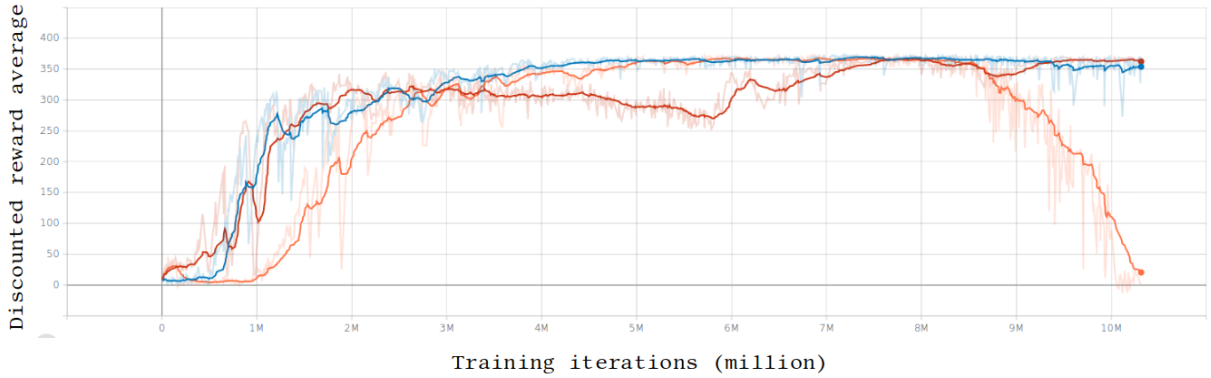


Figure 4.14: Discounted reward during 3 trainings with bad parked cars as obstacles.

In the learning curves (Figure 4.14), we notice on one of the curves a drop of the rewards at the end of the training. There can be several reasons: instability of reinforcement learning in general, or an ill-adapted standard deviation used for Gaussian policy distribution. However the training without obstacles all converge, so the reason may lie in their addition. The probable explanation is that our agent learns to drive closer and closer to the road center, so that it rubs the other cars (Figure 4.15). When it overtakes a car, as soon as it passed it, our car pulls back and may hit the other car. The problem is that our car don't see the other anymore, as we only have one front camera, and no memory in our model (neither recurrent network, nor framestacking...).

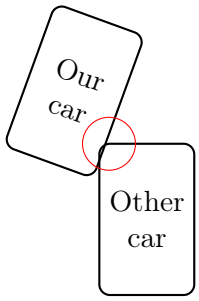


Figure 4.15: Drawing of a collision where our agent pulls back too quickly.

Hereunder on Figure 4.16 is an example of the last observation of a failing episode. Our car scratched the other one by folding too early to the right, so the episode stops and indicates a collision. However, we observe that the other car does not appear on the on-board camera anymore, so the agent doesn't know it is here anymore.

(a) Top view (b) On-board camera

Figure 4.16: Example of a collision with a car invisible from the onboard camera

4.3.8 Adding memory

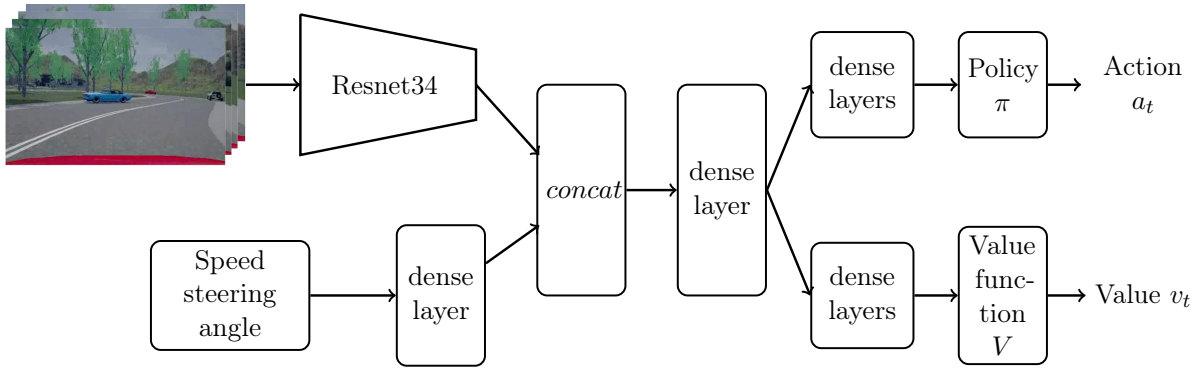
We identified a problem when our agent overtakes obstacles and pulls back too quickly. To overcome this issue, we explored several tracks of memory addition (with a time limit of one month, since we wanted to move to Carla towns, and not stick to the circuit too long):

- Framestacking: we stacked 4 images as input of our CNN to add temporality (not necessarily the 4 consecutive frames, but with some delay to add some distant memory)

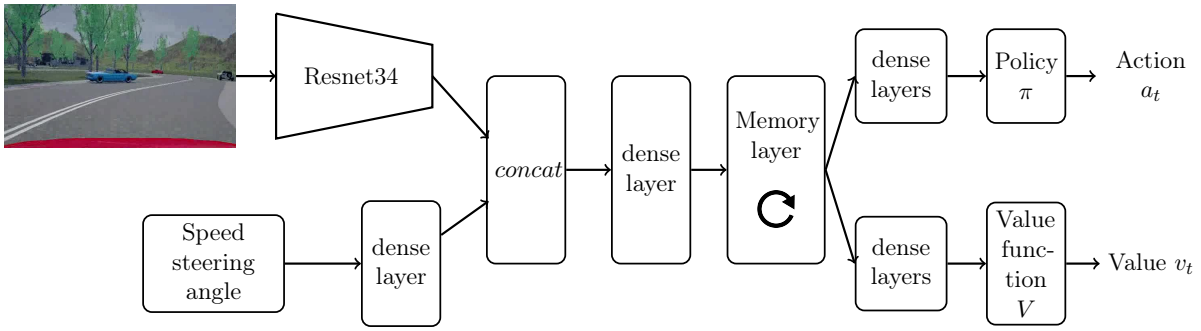
- Representation aggregation: 4 frames were taken, each independently passed through the CNN, and the resulting representation were either passed through conv1D (conv 2+1 D [Tran 2018]), or concatenated, and then passed thought the last dense layers

- Lstm/Gru: addition of recurrent network layers in the CNN

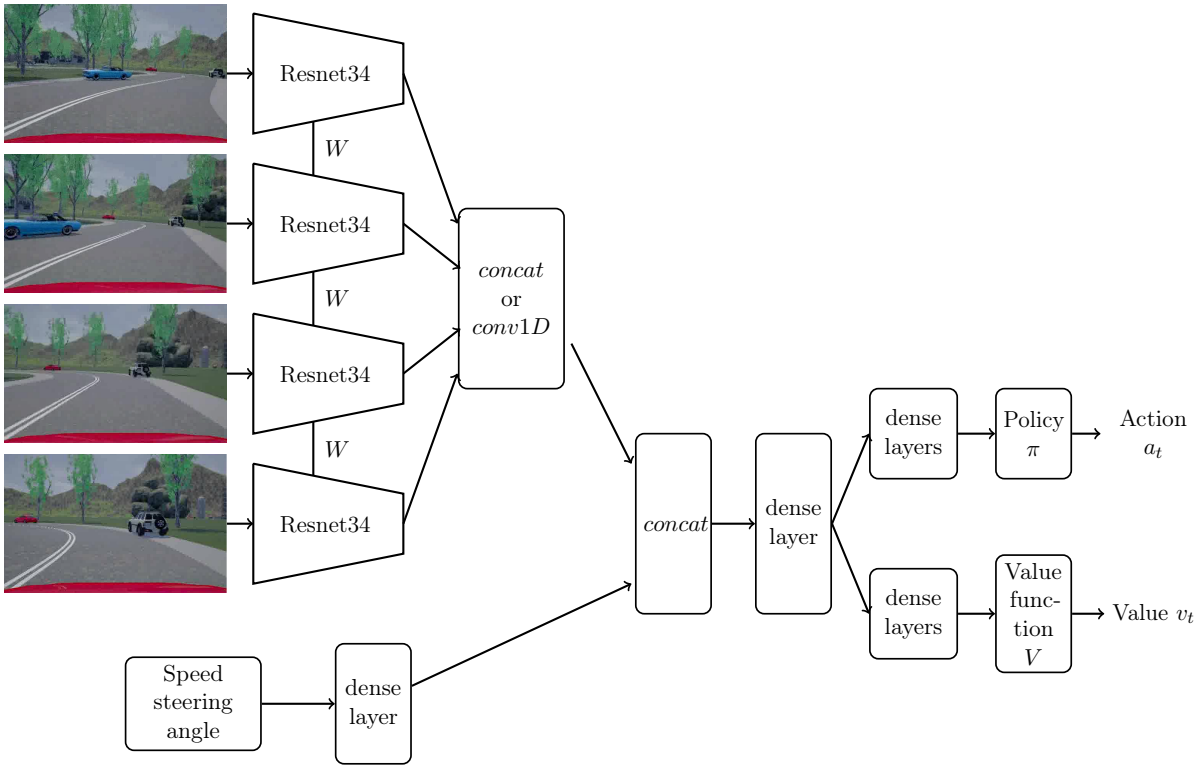
There are also other ideas (adding side and/or back cameras, or adding the history of previous actions as input) that we have not explored. On Figure 4.17 are represented the different architectures that we have studied for memory addition.



(a) Framestacking



(b) Memory layer (gru or lstm)



(c) Data aggregation. Four consecutive frames are separately passed through the encoders - with shared weights - are their features are aggregated, either with a concatenation, or with a convolution 1D.

Figure 4.17: Comparison of three architectures with memory component for conditional driving.

The network with LSTM/Gru layers were very difficult to converge, adding even more

instability to our problem. We tried several methods, pre-training the encoder with semantic segmentation, changing the batch size, but only rare configurations converged (training from scratch with LSTM - we failed to converge the network with a GRU layer). And in the rare case the network with memory layers did converge, the driving results were much poorer than our baseline. The mean speed was barely higher than 20km/h - for a target speed of 40km/h, and collision rate is around 1/3, which is not smaller than the baseline. Training with LSTM layer was also much slower than the baseline (5 days vs 3). Due to the complexity of training a network with memory layer, we have abandoned this track. Despite our efforts, we have also not been able to converge any training with aggregation. Since these training sessions are also long, we did not dig in that direction.

The only one that produced results comparable with the baseline was image framestacking. We stacked 4 frames as input to the encoder, taking 1/5 of the frames (consecutive frame would have been too close in time). Taking 1/5th of the frames allows to cover 1 second of driving time - we set the simulator at 20 FPS. However, it did not solve the problem of hit cars (maybe because this event was too rare to be taken in account), and was slower to converge than the baseline (from 3 days for the baseline to 3-5 days with framestacking).

In total we launched about 60 trainings in an attempt to add memory to the network (each taking at least 3 days to give results). None of the solutions we investigated did improve the results. Therefore we went back to no memory network since we did not want to spend too much time on memory addition. We start to see here a limitation in the choice of PPO as an algorithm - and more generally pure on-policy algorithms. The absence of replay buffer prevents us from seeing more often the rare events and thus from handling them correctly.

4.4 Conclusion

In this chapter we have focused on the task of lane following for a first study of autonomous driving with continuous actions. We placed ourselves in a simplified environment that we designed, i.e. a circuit with variations: straight lines, chicanes, trees, high and low buildings, etc... We built a training network, decided on the inputs, their normalization and the outputs, and chose in particular to produce only a variation of steering wheel angle to limit the oscillations of our agent. Numerous trainings were carried out to adjust all the hyperparameters of the PPO algorithm. We sketched out our reward function so that it could be adapted to urban driving. We found, as expected, that the normalization of the reward had an important role, but also, what was less expected, the influence of the sign of this reward function on learning. To evaluate our agents and make decisions, we have built our benchmark applicable to in-circuit driving. Finally, we have increased the complexity of our environment by playing on both the difficulty of perception - adding more diverse and complex weather conditions - and on difficulties specific to driving - avoiding obstacles. Our agent has learned how to avoid fixed obstacles, and is capable of navigating in weather conditions that it has not seen during training. However we identified a problem when our agent overtakes other cars: it pulled back too early and sometimes hit the obstacles it was avoiding. With only one front-facing onboard camera, and no memory in the network or in the input state, our agent cannot handle these cases. We tried different techniques to add memory to our network, but none of these methods solved this problem, probably because it happens too rarely to be handled by an on-policy algorithm. One solution would have been to add a lot of obstacles to make this event less rare, but we didn't linger on the circuit to quickly move on to urban driving (in the next chapter), while having a prior idea of the limits of our framework. We also identified a generalization problem. If our agent is able to drive in the same conditions as during the training, it seems to have difficulties to drive in weather conditions that

were not encountered before. Generalization will be more specifically studied in the chapter 6.

Chapter 5

From Lane Following to Robust Conditional Driving

Contents

5.1 Introduction

In the previous chapter, we studied autonomous driving on a circuit with reinforcement learning. In this chapter, we will study the transition from lane following to conditional driving. In particular, we will look at how to adapt both the reward function and the model to take into account the driving instruction. Conditional driving presents many challenges. The agent must now manage intersections, which must be crossed safely, while following high-level instructions that indicate which way to go. In addition, one of the problems, especially with on-policy reinforcement learning, is that intersections can be considered rare events, as there are many more times when the agent simply has to follow its lane.

In this chapter we will also study the robustness of autonomous agents. In particular, we are interested in the reaction of the agent if a wrong command is given to it (see Figure 5.1). We believe the agent must be robust in the sense that the information in the environment must keep priority over the high-level order. This can be for example the case where the gps signal indicates

to turn, but the street is blocked for road works, or the case where the gps signal is lost due to bad connection and it does not detect that the vehicle is approaching an intersection.

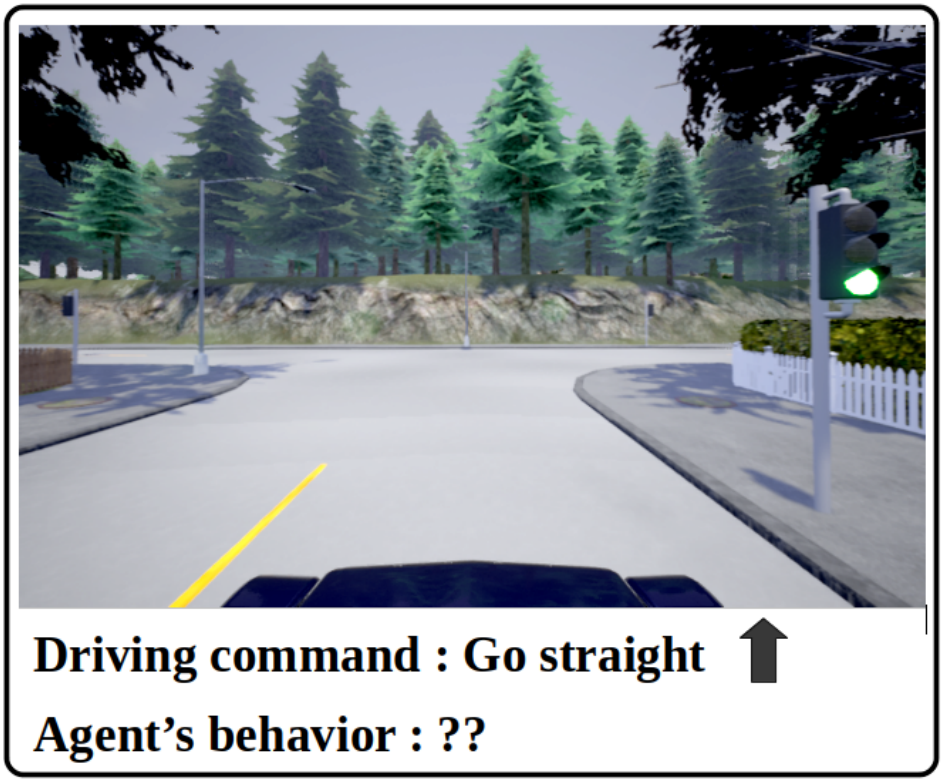


Figure 5.1: What to do if the GPS breaks down?

To summarize, the main contributions of this chapter are:

- Study of how to give the command to an agent in the case of an RL agent using on policy algorithm, and the changes to be made to switch from lane following to conditional driving.

- Implementation of a benchmark to assess the robustness of a driving agent. This benchmark evaluates the reaction of an agent if an erroneous high level command is provided. This work has been subject to the following publication accepted at Intelligent Vehicles Symposium (IV21) :
  [Carton 2021a]: Florence Carton, David Filliat, Jaonary Rabarisoa, and Quoc Cuong Pham "Evaluating Robustness over High Level Driving Instruction for Autonomous Driving". In: accepted to Intelligent Vehicles Symposium (IV21), 2021.

5.2 Background

In this section, we present a summary of the state of the art in conditional driving and robustness. In a first part, we study the different ways to indicate to an agent the path to follow, then we will examine the existing studies on robustness.

5.2.1 Conditional Driving

Early work on autonomous driving focused mainly on lane following [Kendall 2018; Perot 2017; Jaritz 2018]. With the recent development of urban driving simulators (Carla [Dosovitskiy, Ros, 2017] and Airsim [Shah 2017]), a lot of work is now focusing on conditional driving, where the agent must follow a given path. The question of high level order tracking therefore arises: how to add the order in the architecture so that the agent takes it into account? The simplest way is to treat this command as a classical input, and to concatenate it with the features of the other inputs. This is what [Codevilla 2017] and [Hubschneider 2018] do. However, these two works focus on supervised learning, where it is possible to balance the dataset. And even despite this balancing, [Codevilla 2017] found that the command was sometimes ignored.

Introduced in [Dosovitskiy, Ros, 2017], the architecture with branches is widely used in conditional driving. After the CNN, the network is split into several branches, and each branch corresponds to a high-level command (see Figure 5.2). The output used is the one from the corresponding branch. This forces the agent to take into account the command given to it. This method has been used in Atari games for action-dependent video prediction in [Oh 2015]. The action, which is discrete, is represented by a one-hot tensor, and multiplied by the output of the previous module. The rest of the network is therefore specific to the action. Recently, many approaches working on conditional driving, both in reinforcement learning and supervised learning, use this branched architecture [Dosovitskiy, Ros, 2017; Codevilla 2017; Toromanoff 2019; Cultrera 2020].

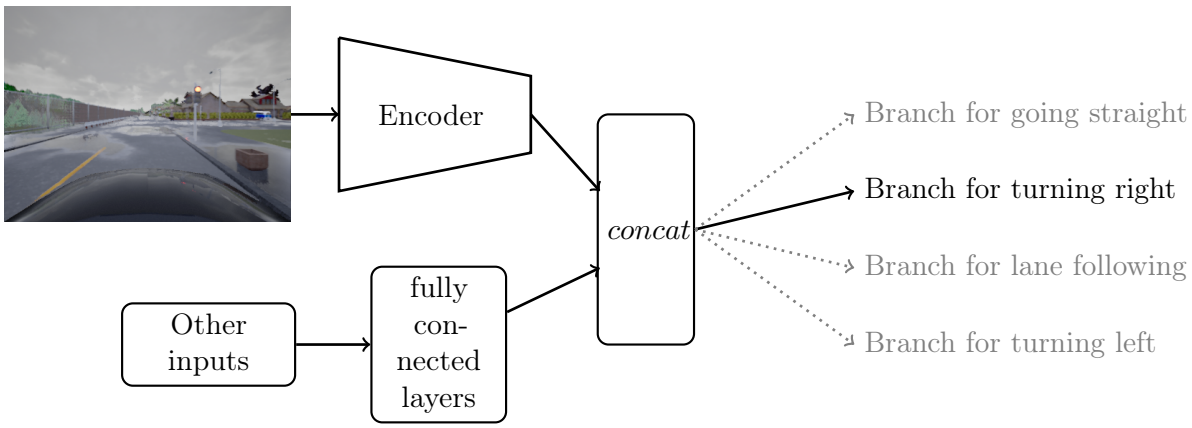


Figure 5.2: Architecture with branches

It is also possible to give a command in the form of a sentence, and both [Chaplot 2018; Wu 2018] use an attention mechanism to take the instruction into account. The instruction is given in natural language, preprocessed, and merged with features from the observation by a gated-attention module. The features are multiplied together using Hadamard's product (term-to-term matrix product).

Finally other approaches do not use the high-level command, which can be restrictive because it is discrete. Some approaches use directly a birdview image of the route to be followed, such as [J. Chen 2019], or tells the agent where to go with an objective, such as [Rhinehart 2020], or a sequence of waypoints to follow, such as [Agarwal 2019].

These different approaches are not necessarily mutually exclusive. We humans are able to follow both a high-level command (turn right), a complex sentence ('Take the second turning on the left') and a path given to us (by a gps). In the following we will focus on conditional driving using a high-level and hence discrete command.

Reward for conditional driving

In the previous chapter we present an overview of the rewards used for lane following. We present here some of the rewards used for conditional driving. The first reward used for conditional driving in Carla is the one from [Dosovitskiy, Ros, 2017]. It is a weighted sum of deltas between current and previous states: traveled distance d, speed v, collision c and intersection with the sidewalk s or the opposite lane o (both $\in [0, 1]$, 0 means no intersection, 1 that the agent is fully outside its lane). We can note the terms about distance and speed are redundant. This leads to the following reward function (Equation 5.1).

$$r = 1000(d_{t-1} - d_t) + 0.05(v_t - v_{t-1}) - 0.00002(c_t - c_{t-1}) - 2(s_t - s_{t-1}) - 2(o_t - o_{t-1}) \quad (5.1)$$

This reward encourages the agent to drive as fast as possible. The later work CIRL [Liang 2018] proposed a different formula, still a weighted sum of five terms, but with significant differences. The reward here depends among other things on the high level command $c \in$ {Followlane, TurnLeft, TurnRight,Straight}.

$$r = r_s + r_v + r_r + r_o + r_d \quad (5.2)$$

with

$$r_o = -100 \text{ when overlapping the other lane} \quad (5.3)$$

$$r_r = -100 \text{ when overlapping the sidewalk} \quad (5.4)$$

$$r_d = \begin{cases} -100 \text{ if collision with other vehicles and pedestrians} \\ -50 \text{ other collisions} \end{cases} \quad (5.5)$$

$$r_s = \begin{cases} -15 \text{ if } |\text{steer}| \text{ in opposite direction for c = TurnLeft or TurnRight} \\ -20 \text{ if } |\text{steer}| > 0.2 \text{ when c = Straight} \end{cases} \quad (5.6)$$

$$r_v = \begin{cases} min(v, 35) \text{ for c= Followlane} \\ min(v, 25) \text{ for c = Straight} \\ v \text{ if } v \leqslant 20 \text{ for c = TurnLeft or TurnRight} \\ 40 - v \text{ if } v > 20 \text{ for c = TurnLeft or TurnRight} \end{cases} \quad (5.7)$$

Contrary to the previous work of [Dosovitskiy, Ros, 2017], this reward has a clear dependance on the high level command. More precisely, with this reward it is very likely that the agent will prefer to collide (which ends the episods with a negative reward of -50 or -100 according to the what it collided into), than simply going into the wrong direction at a crossing (which leads to a sum of negative rewards r_s for going into the opposite direction). In *Learning to Drive using Waypoints* [Agarwal 2019] the reward is much simpler (see Equation 5.8).

$$r = r_s + r_d + I(c) \times r_c \tag{5.8}$$

with $r_s = s$ the speed reward directly proportional to the current speed s, $r_d = -d$ inversely proportional to the cross-track error d, and $r_c = -250 \times s - 250$ in case of offroad or collision, inversely proportional to the speed s. The function $I(c)$ indicates if there is a collision or if the agent goes offroad. This reward also encourages the agent to go as fast as possible, but the collision/offroad punishment is also proportional to the current speed, with higher punishment if the collision occurs at high speed.

Finally the reward from [Toromanoff 2019] is composed of three terms. The first is proportional to the difference with desired speed (maximum when the agent drives at target speed, and linearly decreasing to zero), the second is maximum when the agent is at the center of the road, and the last is inversely proportional to the angle of the agent with the road direction. Episode is terminated with negative punishment if the agent collides, goes offroad, or stops for no reason.

In general, the same terms can be found in the reward function: one on speed and/or acceleration, one on distance to the center of the road, and possibly one on the angle to the path. Also, the conditions of termination are collisions, leaving the road, and when the agent stops without reason.

5.2.2 Robustness

In recent years, there has been a great improvement in the performance of autonomous vehicles. Most of the time, the agents are evaluated on their performance and their generalization capacity (driving in an unknown environment, with different weather conditions, or different luminosity [Dosovitskiy, Ros, 2017], or with dense dynamic agents [Codevilla 2019]). However, there are relatively few evaluations that focus on the robustness of these agents. This is sometimes evaluated, as for example in the Carla challenge, where one of the scenarios simulates a loss of control of the vehicle - due to a very wet road. Some have focused on the robustness on the perception module of driving agents [W. Zhou 2019; Michaelis 2019]. In *Improved Robustness and Safety for Autonomous Vehicle Control with Adversarial Reinforcement Learning*, Ma et al. [Ma 2019] propose a framework to train a robust agent with reinforcement learning, and evaluate it by applying disturbance. The disturbances are applied on acceleration and steering angle. In the same way in ChauffeurNet [Bansal 2018], the agent is trained to be able to handle a trajectory disturbance. We note that no evaluation is done on the reaction of the agents if an incorrect driving instruction is given.

5.3 Crossing Management

In this section we will look at the changes needed in both the reward function and the model to move from lane following to conditional driving.

5.3.1 Reward and model adaptation

We selected several crossings in the training city, and started learning about small episodes with only one intersection, with easy weather. The idea was to verify that our model is capable of learning to turn. We therefore selected episodes where the agent is asked to turn left, and episodes where the agent is asked to turn right. If we trained our agent on these episodes separately, it was able to follow the right path, but we found that if we trained on all possible intersections, the agent preferred to stop before the intersection. The reason for this is the following: we penalize

the agent if it goes too far from his route, with a negative reward and ending the episode. But during the learning process, the agent explores and may takes the wrong road, it does not know yet how and where to turn. So our training ends up in a local optimum, and the agent rather stops in front of an intersection. The penalization when going offroad was too important, therefore the agent would rather just stop and have no reward than risking a big negative reward. To solve this problem, we added a new episode termination condition called bad stop: if the agent stops when it should not, the episode ends and the agent is penalized.

The reward function is then slightly modified and becomes:

$$r_{=} \begin{cases} w_s \times r_{speed} + w_{cte} \times r_{cte} + w_a \times r_{angle} \\ r_{punish} \text{ when collision, offroad, bad stop.} \end{cases} \tag{5.9}$$

with

- $r_{speed} = target\_speed - |target\_speed - speed|)$. The target speed is modified compared to training in the circuit so that the agent slows down at crossing. The normal target speed is 35 km/h as a general rule and 15 km/h at crossings.

- $r_{cte} = (1 - d)$ with d the cross track error, i.e. the lateral distance of our agent to road center

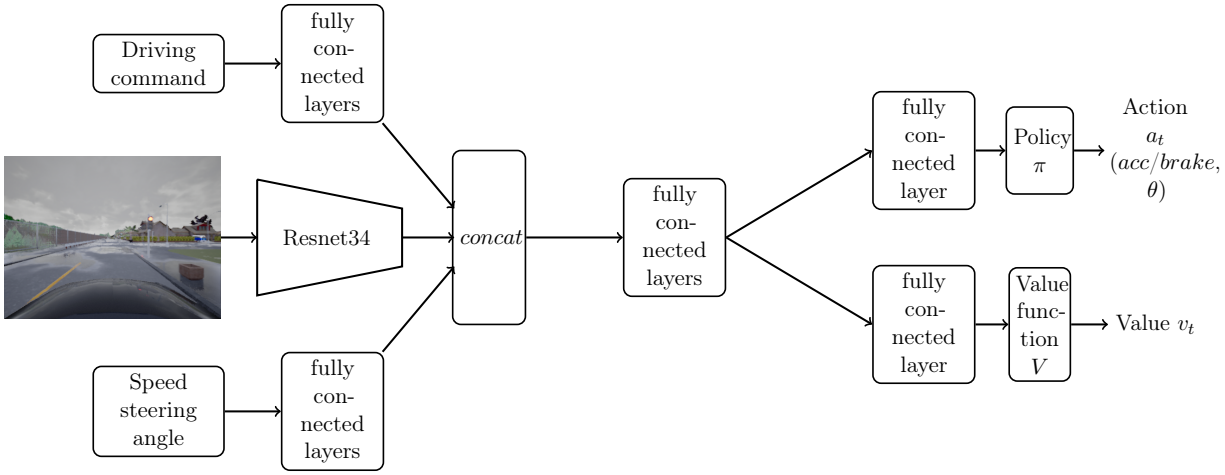
- $r_{angle} = (15 - |\alpha|)$ with α the angle between the direction of the agent and the trajectory.

- w_s, w_{cte} and w_a their associated weights, respectively 1.0, 10 and 0.1.

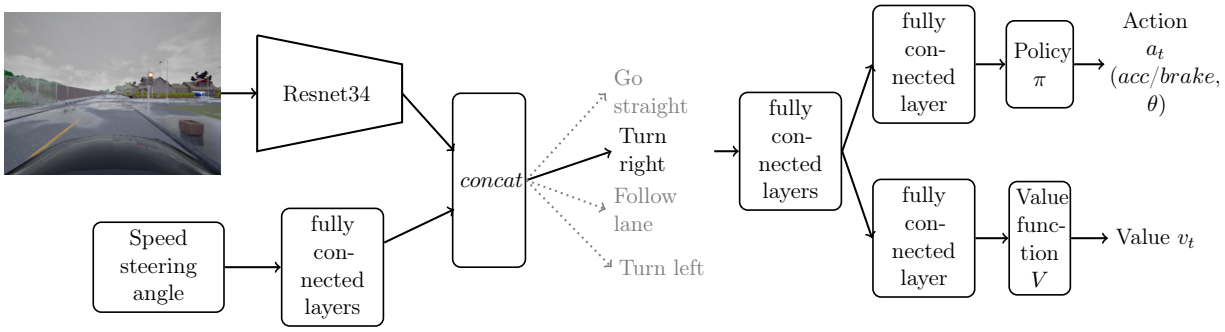
- $r_{punish} = -100$

We have kept the reward function from the previous chapter, adding these modifications. We have seen among other that the positivity of the reward has an impact on the performance of our agent. The weights balance the different terms of the reward, giving the highest weight to the speed term, so that the agent is not tempted to drive at low speed.

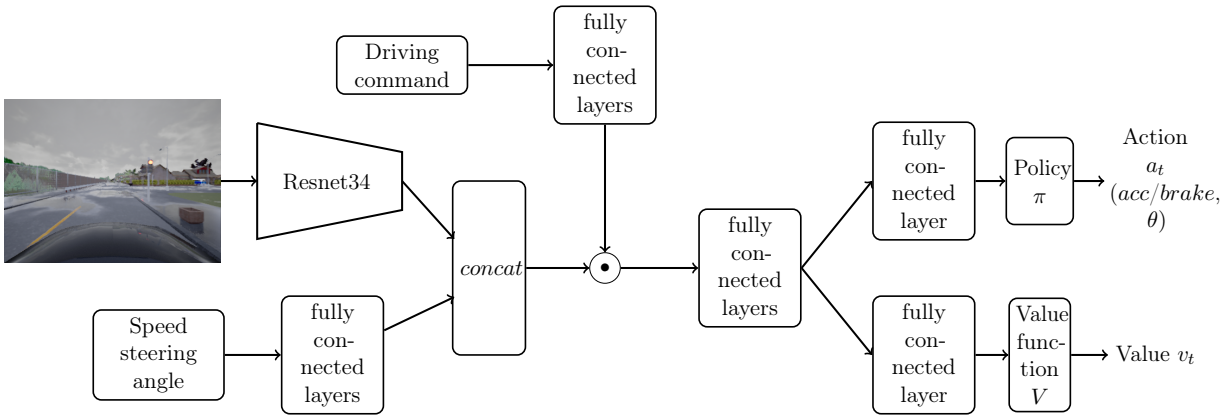
At the same time, we studied the different possible architectures to provide the agent with high level control. To take into account a high level command, we identified three possible architectures (see Figure 5.2). In all three cases the command is discrete, represented by a one-hot tensor. The first architecture is the most classical; it is the simple concatenation of the features coming from the command with those coming from the image and the different measurements. The second is the branch architecture, with specific branches for each command. The last one is an architecture using an attention mechanism. The features coming from the commands are multiplied by those coming from the image and measurements using Hadamard matrix product (element-wise matrix product).



(a) Architecture with concatenation



(b) Architecture with branches



(c) Architecture with gated attention

Figure 5.2: Comparison of three architectures for conditional driving. The architectures differ in the way the high-level command is taken into account. (a) : the features from the command are concatenated with the features from image and measurements, (b) : the command is used to create command-specific branches, (c): the features from the command are multiplied with the others using Hadamard product (gated attention).

We did not test the first architecture using simple concatenation, because the state of the art shows weaker performance than architecture with branches, with a command sometimes

ignored [Codevilla 2017]. We have tested and compared the two following ones: with branches and with an attention mechanism. For our experiments, we used 26 training episodes, and 20 test episodes, with an equal distribution of crossroads where the command is to go straight, turn left or right, and a few episodes with just a road that turns without a crossing. Training episodes are sometimes on the same intersection, but with different driving instruction or starting points. Test episodes are located on intersections that have never been seen during training. Unsurprisingly, we found a big overfit on the training episodes, which is not surprising given the low number of episodes used for learning. But what we found was that the agent with the architecture with attention mechanism tended to ignore the command in a situation it hadn't seen, and simply continue straight ahead. The agent with the branch structure, despite a difficulty to generalize, showed that the command was taken into account.

5.3.2 Discussion

We have studied different possible architectures to add a high level command to the navigation task. In the simplest case of concatenation, the command is not always taken into account, and we found the same problem with the architecture with gated attention. This problem is all the more present because we have chosen an on-policy reinforcement learning algorithm. This means in particular that there is no replay buffer, and therefore no way to balance the dataset to have an equivalent number of each command. In our case, intersections and associated high-level commands are rare events, and it is not possible to modify - even artificially - their occurrence. This is why we decided to work with a model using branches. It allows to have a separate policy for each high-level command, and to force the agent to take into account the requested instruction.

However, the model with branches has 2 major drawbacks. The first is the lack of continuity between branches. When the command changes from following the road to turning right, the agent suddenly changes branches. So it has to learn the same things several times - braking if there is an obstacle, for example, because it does not depend on the given command. The second disadvantage is related to the choice of the high level command. Using a discrete high-level command forces us to have a finite, defined number of possible instructions and instructions like 'turn to the 2nd right' become impossible. So we limit the possibilities: how to do at a roundabout? Or at more convoluted intersections?

It would be interesting for future work to test the attention mechanism for taking into account the command with an off-policy algorithm, such as DQN or SAC. Indeed, the use of branches for each command allows to counterbalance the fact that the commands are not equitably seen during training, and thus not equitably used in the data. Most of the time, the agent has indeed to go straight, crossing situations are rarer than following the road. By removing these branches, the agent tends to perform the very dominant action in the data, namely going straight. Switching to off-policy would allow the rarer data to be used more often.

5.3.3 Training and Results on CoRL Benchmark

In order to train our agent on the navigation task, we used the same parameters as for the circuit for the most part. The main difference is the training duration: to drive in the city, our agent is trained up to 20 Million iterations, against 10 in the circuit. To speed up the training, 16 simulators in parallel are used for data collection, compared to 8 in the circuit. We kept collecting 128 step per environment, which resulted in a batch twice as large (2048), that we split in 16 minibatch of size 128 for backpropagation. Our first trainings were following the framework from the original Carla paper [Dosovitskiy and Koltun 2017], also used in CIRL [Liang 2018], and indicated the high level command corresponding to the closest waypoint. In other words, the

driving instruction was given at the last moment, especially when the agent was at a crossing. We found that this was not robust at all, as the agent would turn as soon as the order changed. So we decided, as with more recent approaches (LBC [D. Chen 2020] or MaRLn [Toromanoff 2019]), to give the order a few meters before the crossing (usually 15). This increases the robustness of the agent, but also the difficulty because the agent now has to learn when to turn. The control of the steering wheel is no longer directly linked to the high level control. We carefully selected training episodes. We chose 25 episodes all over the training town, by taking particular care to distribute the crossings in an equitable way. Our first tests included a lack of left turn and we found that the agent had great difficulty turning left.

Figure 5.3: Examples of training and test environments. Left: training town and training weathers. Right: test town and test weathers.

Our trainings are evaluated on the Carla CoRL benchmark, and we present here the best results. We use data augmentation and an encoder that has been pre-trained on segmentation task. We will see details on both in the next chapter, as well as more extensive results. However, it is very important to note that training is sometimes highly unstable. Identical trainings using the same parameters did not have the same results on the benchmark at all (cf Table 5.3 at the end of this section). So we did several trainings with the same parameters (at least two) to keep the best, which took time and a lot of resources.

CoRL Benchmark [Dosovitskiy and Koltun 2017] is a goal directed urban navigation benchmark. Four different tasks can be evaluated (from going straight to full navigation with dynamic obstacles), and we will focus on the navigation task (navigation without other cars or pedestrians). Dynamic navigation will be dealt with in the next chapter. CoRL benchmark evaluates the driving

capacities of an agent, as well as its generalization capabilities, by testing training conditions as well as unseen town and new weather conditions. Some examples of training and test conditions are shown on figure 5.3. The benchmark outputs the percentage of successful episodes, i.e. when the driving agent has managed to reach destination. For every episode, the trajectory is computed by the simulator and high level commands are sent to the agent with the different information (road picture, current speed, etc...).

We present in Table 5.1 the performance of our agent, and compare them to the state of the art algorithms *Learning by Cheating* (LBC) [D. Chen 2020] and MaRLn [Toromanoff 2019], and also with original RL baseline on Carla Dosovitskiy, Ros, 2017 and CIRL Liang 2018 on the navigation task in Table 5.2. We used the open-source implementations of the benchmark released by *Learning by Cheating*[1] and *MaRLn*[2]. For the latter, in order to make a fair comparison, we used the agent who was trained only on training conditions for the CoRL benchmark. The score is the percentage of success of requested trajectories. Evaluation is made with the version 0.9.6 of Carla.

Task	Training conditions	New weather	New town	New Town & weather
Straight	100%	100%	97%	100%
One turn	92%	96%	60%	68%
Navigation	87%	98%	49%	40%

Table 5.1: Results of our agent on Carla CoRL benchmark

Training	Training conditions	New weather	New town	New Town & weather
RL [Dosovitskiy, Ros, 2017]	14%	2%	3%	6%
CIRL [Liang 2018]	93%	86%	53%	68%
LBC [D. Chen 2020]	**100%**	**100%**	98%	**100%**
MaRLn [Toromanoff 2019]	**100%**	**100%**	**100%**	98%
Our	87%	98%	49%	40%

Table 5.2: Results of our agents and SOTA agent on CoRL benchmark with navigation task

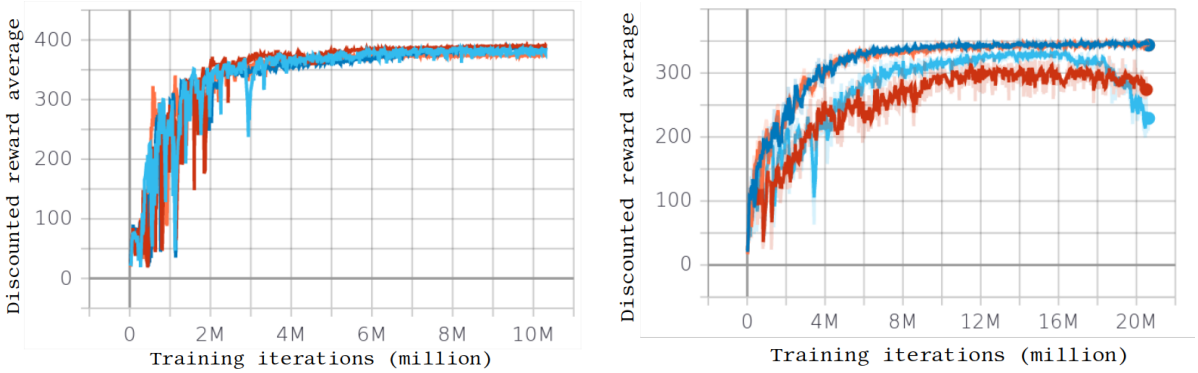
If we are competitive with the method using only reinforcement learning [Dosovitskiy, Ros, 2017], we are quite far from state of the art results. We notice that our agent overfits in the training town, and that it has trouble to generalize driving in an unknown town.

Here we also draw the limits and flaws of the on-policy algorithms. We had to choose very precisely the training episodes, and a slight imbalance (less left turns for example), led to a drop in performance. We also found another imbalance: out of the four training meteorological conditions, one of them is more difficult - *HardRainNoon*. The agent found it harder to navigate through it, so the training episodes were shorter in this configuration, and therefore there were fewer examples in the collected data. The fact that there is no replay buffer does not help to compensate for these imbalances.

[1] `https://github.com/dotchen/LearningByCheating`
[2] `https://github.com/valeoai/LearningByCheating`

Instability - High variance

In contrast to circuit driving, especially on empty circuits with simple weather conditions, city driving training is much more unstable. The figure 5.4 shows the learning curves of 4 identical trainings, differing only by their random seed, on the circuit and in town. It can be seen that urban driving training has much more diverse curves, and the performance of the agents resulting from these trainings is also very varied (see Table 5.3).



(a) Discounted reward of 4 identical trainings - different random seed - in empty circuit

(b) Discounted reward in 4 identical trainings - different random seed only - in empty town

Figure 5.4: Variance between several trainings with different random seeds, in the circuit (left) and in the town (right)

Training	Training Condition	New Weather	New Town	New Town & Weather
Orange	**87%**	98%	49%	**40%**
Dark Blue	86%	**100%**	**51%**	24%
Cyan	8%	8%	11%	12%
Red	0%	0%	7%	6%

Table 5.3: Results of 4 agents trained with different random seed on CoRL benchmark with navigation task

5.4 Robustness of Learning

It is very classic to evaluate an agent on an environment it has not already seen, to measure its generalization capacity. We will study generalization in the next chapter. On the other hand, there are few studies on the robustness of agents. Robustness is defined in terms of reliability, and assesses the capacity to withstand variations without consequences.

We propose here a benchmark that would allow an evaluation of the ability of driving agents to react in case of unexpected command, and thus measure their ability to understand the environment around them.

5.4.1 Benchmark Description

Our benchmark is inspired by the CoRL benchmark from Carla [Dosovitskiy, Ros, 2017]. In CoRL benchmark, the command is always correct. Therefore, to study the robustness of the

agents, we build a benchmark that consists in giving an incorrect command at an intersection, and evaluate the reaction of the driving agent. To evaluate only the robustness of the agent, we place ourselves in the simplest conditions, i.e. in conditions that have already been seen during training, namely the training town with ClearNoon time/weather preset. Since we only work in an easy map of Carla, there are 4 possible commands: follow lane (basic command when there is no crossing), go straight, turn left and turn right (neither lane change nor traffic circle). We want to evaluate the reaction of the driving agent if an incorrect command is given at an intersection. We can distinguish two cases: either the agent is told to go at a direction that does not exist, or the basis command (follow lane) is still given even if the agent reached a crossing. We will measure the number of unsafe behavior (ie collision, offroad, or stop at the middle of the crossing) in these cases (see Figure 5.5 for examples of unsafe behaviors). This will help to measure if the agent has achieved a certain understanding of the environment, or if it blindly follows instructions. For each crossing type, we selected 10 of them in the training town, and run small episodes starting before the crossing with erroneous command.

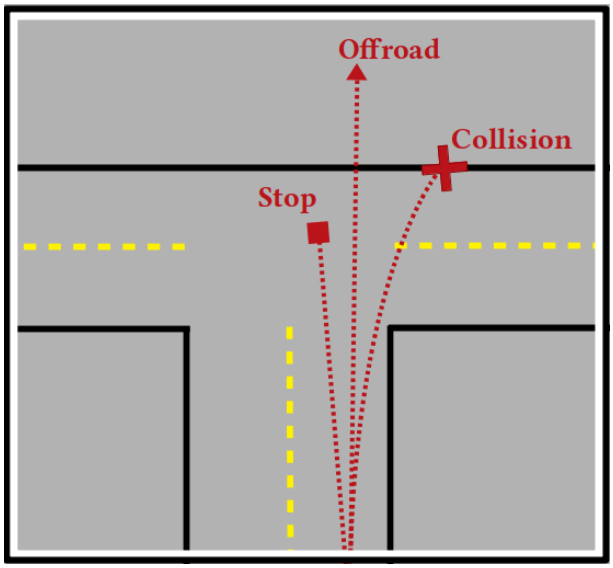


Figure 5.5: Examples of unsafe behaviors.

In the event that a wrong command is given at a junction, one can expect two behaviors from a human driver: either it stops before the junction or it randomly takes one of the proposed routes. Out of the 10 test episodes, we have therefore counted as unsafe behavior the cases where the agent stops in the middle of the crossing, and the cases where the agent collides or goes offroad. For every crossing type we counted the unsafe behavior for both wrong command (that is not available at the given crossing), and basis - follow lane - command.

In Table 5.4, we present the results of our agent by comparing it to two state of the art agents [D. Chen 2020] and [Toromanoff 2019]. Since the results measure unsafe behavior here, the smaller the more robust. In the next chapter, we will also study two other agents that we test here as well. This is an agent trained with segmentation as input, and an agent trained with segmentation auxiliary task. More details will be given in the next chapter, but for further study and discussion, we also test them here.

These agents are trained with the same algorithm (PPO), and the same reward function. Unlike our image input agent, the encoders of these two agents are trained from scratch. The differences are listed below:

- Agent with Segmentation input
  - RGB image input is replaced by segmentation mask.
  - Resnet34 encoder is replaced by smaller CNN from [Mnih 2015] which we call NatureCNN
  - Encoder trained from scratch
- Agent with segmentation auxiliary task:
  - A decoder is added to output semantic segmentation
  - Encoder trained from scratch
  - Semantic segmentation and driving are trained at the same time

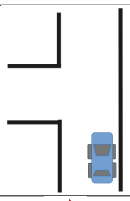
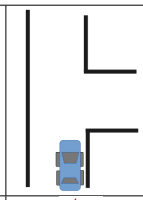
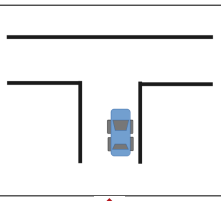
	Type of crossing				
	wrong command				Total unsafe
LBC [D. Chen 2020]	Unsafe behavior with wrong command	0	5	10	23 / 60
	Unsafe behavior with 'follow lane' command	0	0	8	
MaRLn [Toromanoff 2019]	Unsafe behavior with wrong command	2	**1**	10	13 / 60
	Unsafe behavior with 'follow lane' command	0	**0**	**0**	
RL seg input	Unsafe behavior with wrong command	0	10	9	35 / 60
	Unsafe behavior with 'follow lane' command	0	6	10	
RL img input	Unsafe behavior with wrong command	0	**1**	7	16 / 60
	Unsafe behavior with 'follow lane' command	0	4	4	
RL with auxiliary task	Unsafe behavior with wrong command	0	2	**0**	**3 / 60**
	Unsafe behavior with 'follow lane' command	1	**0**	**0**	

Table 5.4: Results of our agents and two state of the art agents on our wrong-command benchmark, showing the number of unsafe behavior on 10 trials.

5.4.2 Qualitative Analysis

We first note that the results are very disparate according to the type of crossing. For the first crossing type, where the 'turn right' command is not possible, all the agents drive correctly (with sometimes bites into the sidewalk, but overall safe behaviors). This is probably due to the fact that the sidewalk is always present to the right of the agent, and that this helps the agent to drive correctly by simply following it.

For the second crossing, where the 'turn left' command is not possible, the agents have different behaviors. The LBC [D. Chen 2020] agent deviates slightly if it is still asked to turn left, and then in half of the cases, stops abruptly in the middle of the crossing, which we considered

to be dangerous and therefore unsafe behavior. The MaRLn [Toromanoff 2019] agent keeps its course, and both [Toromanoff 2019] and [D. Chen 2020] perform well when the command remains to follow the road. Among our agents, the behavior varies greatly. The agent using segmentation as input has a very high collision rate, because it turns directly left if this command is given. The two agents using the input image react better, even though the auxiliary task seems to help to behave correctly.

Finally the third type of intersection, where it is not possible to go straight, appears to be the more difficult for all agents. The LBC agent has a very bad error management: in the vast majority of cases, it keeps going straight, even if there is a wall, and usually hits any obstacle in front of the car, and so does our agent that uses segmentation. The MaRLn agent manages to turn when the command follow lane is still given - by sometimes biting into the other lane, but keeps going straight if the command straight is given. However, unlike LBC, it rarely collides, preferring to stop in the middle of the crossing. On the other hand, our agent which has learned with an auxiliary task does not make any mistakes. It turns arbitrarily to one side or the other, even if it is asked to go straight, which shows that it relies much more on the analysis of the environment around.

5.4.3 Discussion

We present the performance of 5 driving agents: one trained with imitation learning (LBC [D. Chen 2020]), and the four others with reinforcement learning. We are not surprised by the rather low performance in terms of robustness of LBC. This is one of the drawbacks of supervised learning, since the cases of error are not present in the learning dataset, the agent never learns how to react to unexpected situations.

One might have thought that reinforcement learning, because of its trial-and-error nature, would alone have increased the robustness of learning. However, the disparity in the results of agents trained through reinforcement learning shows that this is not enough. We can compare the performance of our two agents with RGB image as input. Although using an encoder pretrained on the segmentation task, the simple agent (i.e. without auxiliary task) is less robust than the one using segmentation as an auxiliary task. The use of semantic segmentation as auxiliary task allows the agent to make the link between the semantic parts of the image and the rewards, and thus to have a better understanding of the environment. It is able to analyze its surroundings, and to make a decision in a situation it has never seen before by giving priority to the visual information over the high-level command.

MaRLn agent [Toromanoff 2019] is quite close to ours using auxiliary task, with the following differences: the encoder is pretrained to calculate segmentation as well as other affordances (distance to traffic lights, position in the road...), and frozen during RL driving training. Perception is then decoupled from driving. Our agent in contrast is trained end-to-end with both segmentation and driving. We believe that decoupling perception and control may reduce the robustness of the agent.

We have also analyzed the different experiments of our agent using only segmentation, and a possible explanation for the poor performance in terms of robustness is that it is too confident on the road marking perception. Indeed, the segmentation in Carla being perfect, this agent has never needed to use anything other than the road lines to learn to drive. In normal situations, when it reaches a crossroad, our agent starts to turn in the prescribed direction, and adjusts its trajectory with the lines of the road it joins (road markings are not present in the intersection). However, in the case of a wrong command, it starts to turn and cannot correct itself because there are no more lines - because the target road doesn't exist.

On the other hand, the agents which learned with an RGB image as input were subject to

data augmentation. In particular, road lines are not always visible in the image (very rainy weather or Gaussian blur), so the agents learn to use other elements - the sidewalk for example. These agents therefore learn with information redundancy, which is not the case if only perfect segmentation is used. One lead for future work to validate this hypothesis would be to train an agent with segmentation input, by regularly and randomly removing one or more masks. The agent would then learn to use the different elements present in the observation in a robust way.

5.5 Conclusion

In this chapter, we first studied how to add a high-level command to give a driving direction instruction. We have seen that in the case of an on-policy reinforcement algorithm, the most efficient way is to use a branch architecture. Indeed the on-policy character does not allow to balance inputs, and thus to balance the situations seen by the agent. The use of branches allows to specialize a part of the network and avoid that the agent learns only with the most frequent examples.

We also propose a first study to evaluate the robustness to wrong command of a vision based driving agent. With the development of end-to-end conditional autonomous vehicles, we believe this issue is of crucial interest and has so far not been dealt with. We found in this evaluation that an agent appears to be more robust when perception and control are not decoupled during training. We also observed a better robustness when the agent has a high-level understanding of its environment with, for example, the ability to segment the observation, but also when it learns to use the redundancy of the elements present in the observation.

As future work, we can imagine others tests to evaluate the robustness: changing the visual input by placing the camera at a different position, changing steering wheel hardness, etc... Since all the situations cannot be seen during training, we need to have agents that are capable of adaption and understanding of the environment, in the sense that they must extract enough information to avoid dangerous situations.

Chapter 6

Exploration of Methods to Reduce Overfit

Contents

6.1 Introduction

In the previous chapter, we set up a framework for reinforcement learning applied to urban vision-based autonomous driving. Our agent is however facing a big problem of generalization. If changing the weather seems to have little effect, navigating in a town that was not seen during training leads to a drastic drop of performance. In this chapter we will study different methods to reduce this overfit and increase generalization performance in the context of autonomous driving in an urban environment. We are particularly interested in semantic segmentation and different ways of using it: through data augmentation, directly as input, as pre-training or as auxiliary task.

This study has been subject to the following publication at Autonomous Vehicle Vision Workshop at WACV 2021:
[Carton 2021b]: Florence Carton, David Filliat, Jaonary Rabarisoa, and Quoc Cuong Pham "Using Semantic Information to Improve Generalization of Reinforcement Learning Policies for Autonomous Driving". In: *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV) Workshops*, 2021.

6.2 Background

In the context of autonomous navigation, the ability to generalize to an unseen environment is a crucial element. The driving agent must be able to navigate in an environment it has never seen before, and also in various weather or lighting conditions. When introducing Carla simulator [Dosovitskiy, Ros, 2017], its authors proposed the CoRL benchmark for goal directed navigation. This benchmark measures the performance of driving agents, both in training conditions, and in different conditions (new city, new weather, or both), on different tasks - from going straight to dynamic navigation with other agents (vehicles and pedestrians). A larger and more complex benchmark on Carla, called NoCrash is proposed in [Codevilla 2019]. It proposes 3 urban navigation scenarios: empty town, regular traffic, and dense traffic. Like CoRL benchmark, 4 conditions are evaluated: training conditions, new city, new weather, and new city and new weather. If the training and test towns are the same as CoRL benchmark, as well as the training weathers, one of the two test weather is replaced (WetCloudyNoon is replaced by WetSunset).

We found in previous chapter (Chapter 5) that our agent's ability to generalize is poor, and this finding can be extended to the performance of many state of the art works on the benchmarks of Carla simulator. This is particularly true for navigation and dynamic navigation tasks, as well as in NoCrash benchmark, for approaches using both reinforcement learning and imitation learning. On the same weather conditions, CIRL [Liang 2018] performance drops by about 40% between train and test town on both navigation and dynamic navigation tasks. Less drastic, *Learning to Drive using Waypoints* has a drop of almost 20% between the two cities in dynamic navigation - whereas the input used is a bird view of segmented image. In supervised learning, CIL [Codevilla 2017], MT [Z. Li 2018], CAL [Sauer 2018] and ILA [Cultrera 2020] all experience drops in performance (between 10% and 40%) when they are evaluated in the new town, and most strongly in the dynamic navigation task. We refer the reader to the Table 3.2 from Chapter 3 for an overview, where most of the work presents difficulties in generalizing. Only very recent work show perfect scores on Carla CoRL benchmark [Toromanoff 2019; D. Chen 2020], and these approaches still present difficulties on the more complex NoCrash benchmark. Despite recent progress, generalization is still an open problem for autonomous driving.

In a broader context, generalization is a key element in computer vision. However, as mentioned in *Quantifying Generalization in Reinforcement Learning* [Cobbe 2019], it is common to use the same environment for training and test in reinforcement learning benchmarks. Moreover, classical techniques employed in supervised learning are not automatically used when training with reinforcement learning. For example, it is interesting to note that in Carla original paper [Dosovitskiy, Ros, 2017], the authors automatically add methods to reduce overfit in the training with imitation learning - data augmentation and regularization techniques like dropout - whereas none is applied for RL training.

In this section we will first present a brief overview of classical techniques used in computer vision to reduce the overfit, and secondly we will study the addition of information as a way to increase both performance and generalization.

6.2.1 Classical Methods to Reduce Overfit

In supervised learning, many techniques are used to improve generalization and reduce overfit in deep neural networks, especially when applied to computer vision. The most classical regularization techniques are batchnorm [Joseph 2016], L1 or L2 regularization [Krogh 1992], and dropout [Srivastava 2014].

Batchnorm and dropout are specific layers that can be added in the network. The dropout is the temporary deactivation of a certain proportion of neurons. Carried out randomly at each

passage in the network, the objective is to reduce the co-adaptation of neurons. During the test, all neurons are reactivated, and their output is averaged according to the percentage of neurons deactivated during training (if 50% of the neurons are deactivated during training, the output of the neurons is multiplied by 0.5 during test). The batchnorm normalizes the input of an internal layer in the network, in the same way that the input of the network is normalized. The mean and the variance are calculated on the mini-batch given as input. In parallel, the batchnorm also learns a sliding mean and variance to be applied during the test on a single sample. It should be noted that dropout and batchnorm have a distinct behavior during training and testing. However, reinforcement learning algorithms interact with the environment, alternating between data collection and policy improvement. This raises a difficult question: should the dropout and batchnorm layers therefore have train or test behavior during data collection?

Easier to apply in RL, L1 or L2 regularization consists in penalizing the high values of the network weights. A regulation term is added to the global loss. On Equations 6.1 and 6.2 are shown both regularizations.

$$loss_{regL1} = loss + \lambda \sum |w_i| \tag{6.1}$$

$$loss_{regL2} = loss + \lambda \sum w_i^2 \tag{6.2}$$

These methods modify networks to improve generalization. In computer vision, inputs can also be modified to artificially vary the data that are provided to the network. Addition of noise, blur, change of colors or contrast, geometric modifications, these modifications are called data augmentation, of which the work [Shorten 2019] presents an extended overview. [Laskin 2020] focuses on data augmentation, and proposes a plug-and-play module to add data augmentation in reinforcement learning framework. The latter, although crucial and widespread in supervised learning, is not at all generalized in RL. When comparing the baselines in both imitation and reinforcement learning in autonomous driving, CARLA authors [Codevilla 2017] do apply data augmentation and regularization techniques (dropout) during imitation learning training, and not during reinforcement learning.

The work [Cobbe 2019] studies the problem of generalization and the way to apply the classical methods (data augmentation, L2 regularization, dropout...) to reinforcement learning training. They conclude that classical techniques used in supervised learning can be successfully applied in reinforcement learning with some notable success of data augmentation and batch-normalization. It should be noted that these papers [Cobbe 2019; Laskin 2020; Lee 2020] are recent and that their studies were therefore not available at the beginning of this thesis.

Close to data augmentation, domain randomization is used to enable the transfer from one environment to another, in particular to fill the reality gap. It consists in randomizing some parameters of the environment (colors, textures, lighting, position of camera, noise, ...) in the simulated environment, so that the transfer to real world is smooth. Domain randomization is applied in simulation training for robot grasping in [Tobin, Fong, 2017; Tobin, Biewald, 2017], and in drone navigation in [Sadeghi 2016], where the drone was only trained in simulation and tested in real life. In the same spirit, in both [Rajeswaran 2016; Peng, Andrychowicz, 2018], the authors randomize the physical parameters of the system (mass, length, damping, ...) to build a more robust policy.

6.2.2 Use of Additional Information to Improve Training and Generalization

In addition to classical methods, the ability of an agent to generalize increases with the number of different environments it encounters. Unsurprisingly in [Cobbe 2019], the agent increases its performance as the number of different environments seen during training increases. Similarly

in [Toromanoff 2019], an ablation study is performed to compare the performance of an agent in a city it did not see, depending on whether one or more cities were used during training. As expected, performance increases with the addition of data, but the highest performance in an unseen city peaks at 58.4% of successful episodes. This indicates that there is still plenty of room for improvement, especially in autonomous driving with RL. Another very standard practice is to add information via pretraining the neural network, either with the same task (autonomous driving in CIRL [Dosovitskiy, Ros, 2017]), or with different task (collision probability in [Sadeghi 2016] for drone navigation). However, different data and new environments are not always available. In *Exploring data aggregation in policy learning for vision-based Urban autonomous driving* [Prakash 2020] is proposed an improvement of Dagger [Ross 2011] for autonomous driving for better generalization. The general idea is to sample critical states - intersection, presence of pedestrians... - from on-policy data collection, and use a replay buffer to focus on these high complexity states. This methods aims at counter the natural bias of the dataset, and showed improved performance compared to CILRS [Codevilla 2019].

Last but not least, an auxiliary loss (or multi-task training) was first used in [Jaderberg 2016] with deep reinforcement learning, and it showed improved performance. However, no study on generalization capabilities is made. The idea is to train several tasks at the same time, the main one is trained with reinforcement learning, and the auxiliary tasks are trained with supervised or unsupervised learning. Auxiliary loss is also used with reinforcement learning in *Learning to navigate in complex environment* [Mirowski 2016] to predict depth to help an agent navigate in a maze. In autonomous driving the classic auxiliary tasks are depth, segmentation, or movement (optical flow) [Hawke 2019; Z. Li 2018; Toromanoff 2019; B. Zhou 2019]. However, these methods require additional information - for depth or segmentation - or additional calculations - for optical flow. To use what we already have at hand, we can think of the reconstruction of the original image using a VAE [Kendall 2017; Agarwal 2019], or future prediction - next frames prediction [Santana 2016; Mahjourian 2017; Vukotić 2017] or future measurement prediction [Dosovitskiy and Koltun 2017].

In this chapter we focus on data augmentation and semantic segmentation. We will see that semantic segmentation contains (almost) all the necessary elements for urban driving, and we will study how to incorporate this information during training, in data augmentation, in pre-training and as an auxiliary task.

6.3 Experiments

In this section we present the different experiments we have conducted and their results. Most of our conclusions come from the navigation task experiments, using Carla CoRL benchmark [Dosovitskiy, Ros, 2017], a part of our experiments was also carried out in a first time on circuit. We have presented in chapter 4 our circuit, created to study autonomous driving under simple conditions. To study the generalization, a second circuit dedicated for the test was built. To make it different from the first one, we varied the environment around the circuit, and also changed the color of the center line. An image of our test circuit is shown in figure 6.1.

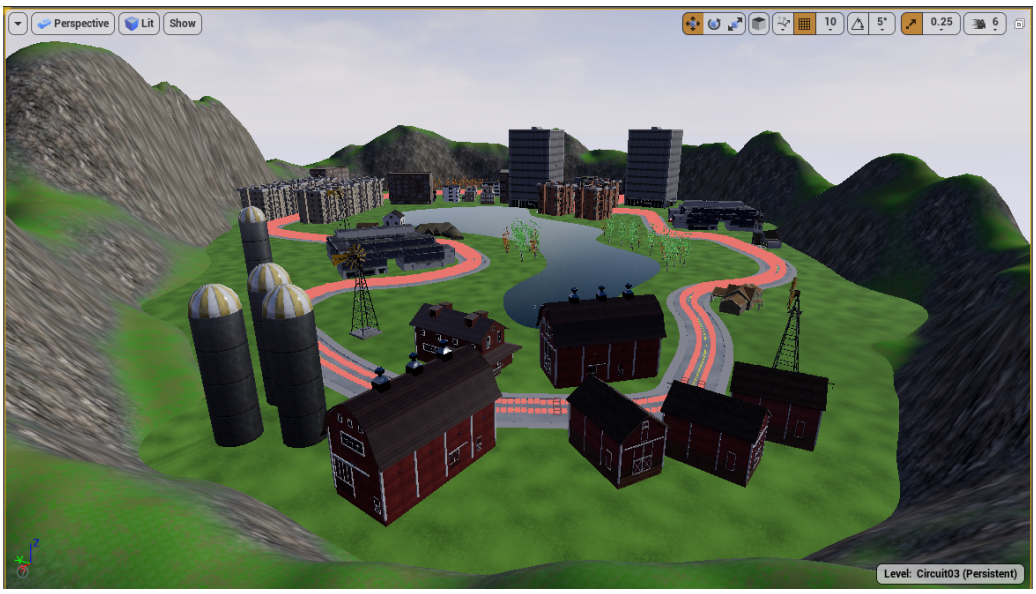


Figure 6.1: Test circuit

In a first part, we will study the impact of data augmentation on generalization and propose a data augmentation with a semantic segmentation component. Then in a second part we will study the addition of segmentation as a pre-training as well as an auxiliary task. Finally we will extend our study to dynamic navigation to compare with other state of the art methods.

6.3.1 Data Augmentation

As described in the chapter 4, in addition to using different weathers and times of day, we used data augmentation, inspired by the one used for supervised training in Carla original work [Dosovitskiy, Ros, 2017]. Data augmentation includes random variation in hue and saturation, brightness and contrast, along with conversion to gray scale, partial erasing of the image, salt and pepper noise, and Gaussian blur.

The data augmentation is applied independently on each image, and thus varies during an episode. The different data augmentation are combined in a random order, and also in a random way, the most frequent being the variation of contrast, hue and saturation (half of the time), and the least frequent being Gaussian blur and gray-scale (10% of the images).

When creating the test track, we chose a different color for the middle line of the road. As we noticed a big loss of performance of our agent when it passes on this test circuit, we had the idea to randomly change the color of the middle line during the training (see Figure 6.2), idea that we then extended to different semantic parts of the environment (see Figure 6.3 for examples in the town).

6.3.1.1 Circuit

On Figure 6.2 is shown an example of data augmentation performed in the circuit. Random erasing as well as salt and pepper noise are added to the original image, and the lane color is randomly changed. The results of the three following benchmarked trainings are presented on table 6.1: without any data augmentation - only four different training weathers, with classical data augmentation, and with classical data augmentation plus random semantic variation.

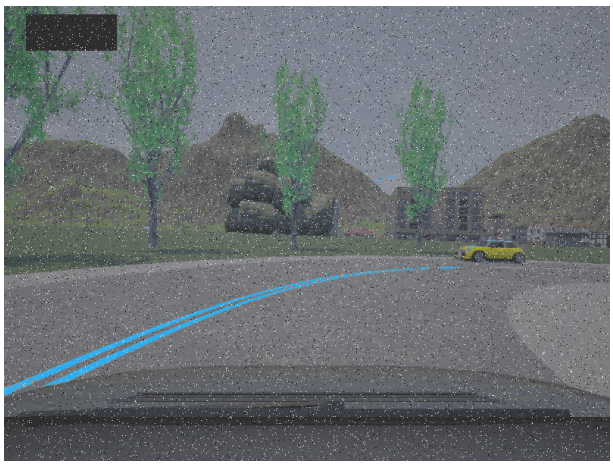


Figure 6.2: Data augmentation in circuit with random line color

	Training Condition	New Weather	New Track	New Track & Weather
speed (km/h)	29.4 (0.8)	28.0 (5.0)	24.6 (6.8)	26.2 (5.0)
cte (m)	0.3 (0.1)	0.3 (0.1)	2.2 (2.8)	2.7 (4.1)
atr (°)	1.8 (0.7)	2.0 (1.2)	12.3 (10.8)	13.9 (14.1)
distance (m)	637.2 (274.3)	578.0 (320.9)	150.3 (206.5)	135.3 (150.3)
collision	16/36	16/36	34/36	36/36

(a) No Data Augmentation

	Training Condition	New Weather	New Track	New Track & Weather
speed (km/h)	35.3 (6.1)	36.2 (6.2)	11.7(13.3)	16.4(14.2)
cte (m)	**0.2** (0.1)	**0.2** (0.1)	**4.7** (5.6)	**2.6** (2.6)
atr (°)	**1.4 (0.3)**	**1.3 (0.2)**	**22.7 (25.4)**	**14.1 (14.2)**
dist (m)	775.4(311.9)	851.0(286.8)	146.3(258.7)	222.4(322.8)
collision	16/36	15/36	**17/36**	**17/36**

(b) Data Augmentation without color randomization

	Training Condition	New Weather	New Track	New Track & Weather
speed (km/h)	**36.4** (7.8)	**36.9** (6.3)	**29.0** (9.2)	**29.5** (8.3)
cte (m)	**0.2** (0.3)	**0.2** (0.3)	4.8 (5.9)	2.9 (3.3)
atr (°)	1.7 (1.5)	1.7 (1.3)	30.8 (33.7)	25.0 (28.0)
dist (m)	**978.7 (274.3)**	**881.1 (332.7)**	**350.8(468.8)**	**359.7 (462.1)**
collision	**3/36**	**10/36**	22/36	25/36

(c) Data Augmentation with color randomization

Table 6.1: Comparison of the benchmark at 10 Millions steps with and without data augmentation. The figures presented are the mean (and variance) on different measurements

When classical data augmentation is added, performance improves, especially in the training town. The agent drives at a speed close to the target speed (40km/h) while it was 10km/h below without data augmentation. It also remains a little more in the middle of the road, and covers on average a greater distance in each episode. In the test city, collisions decrease, despite performances that remain low. Part of this poor performance is potentially due to a different colored center line, and we see in the last table an improvement in performance when we add a random variation in the color of this line. The speed of the agent gets closer to the target speed (40km/h), and despite a large number of collisions, the agent covers more distance in each episode on average. The agent generalizes a little better, and the performance is also significantly better in the training city, with a significantly lower number of collisions and a higher distance traveled.

6.3.1.2 Town Environment

We extended this study to the city to confirm our results, by comparing the results of different data augmentation on the navigation task. We extended our data augmentation by randomly changing the color of all semantic parts of the input image, examples can be seen on Figure 6.3. We observe in the table 6.2 the same trends. We first notice that learning without data augmentation is not efficient at all (we are barely higher than the RL baseline from the original CARLA paper [Dosovitskiy, Ros, 2017]), even under training conditions. Classic data augmentation allows to increase the performance, both in training conditions and in the new city, but does not really reduce the gap between the training city and the new city. If we look closer at the results of training without data augmentation, we notice that with a few exceptions, only the episodes with the easiest weather (ClearNoon) were successful. The addition of noise (classic DA) in the input image has therefore made it possible to better learn the different meteorological conditions, even if they are all seen during the training. The addition of segmentation in the data augmentation allows to reduce the generalization gap a bit, even if it is still important. Highlighting the semantic elements of the image allows the agent to make the link with the reward function, and thus to better generalize. However, even if progress is made compared to the training without data augmentation, the gap between training town and test town is still huge.

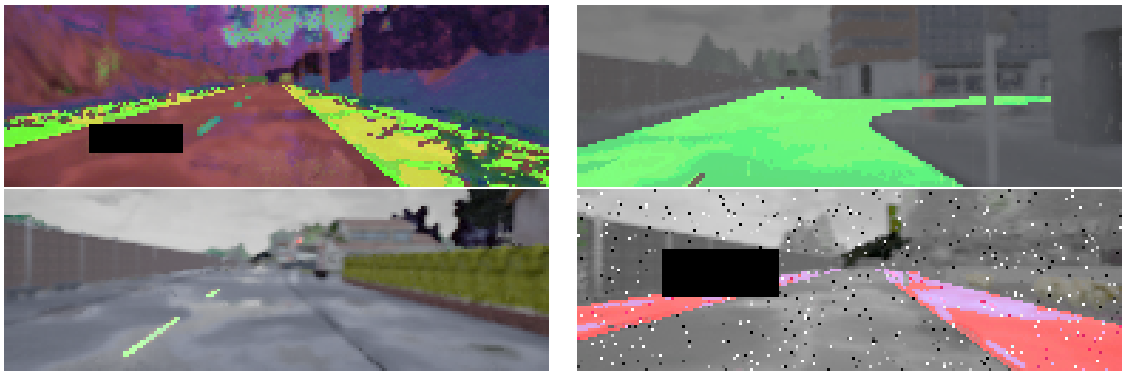


Figure 6.3: Examples of data augmentation in town

Training	Training Condition	New Weather	New Town	New Town & Weather
No DA	34%	6%	9%	2%
Classic DA	57%	60%	22%	4%
Semantic DA	**67%**	**60%**	**34%**	**28%**

Table 6.2: Baseline results for navigation task with different data augmentation

6.3.2 Semantic Segmentation as Auxiliary Task

We have seen in the previous section that data augmentation helps better learning and generalization, and that the addition of semantic information within the data augmentation itself also allows better generalization. We intuit that semantic segmentation plays an important role in generalization. Indeed, it is important that the agent distinguishes the road from the sidewalk, detects cars and pedestrians, and is able to do so in an environment that it has not seen before. In this section we will first confirm this hypothesis by training an agent with only segmentation as input. Then in a second part, we will compare the benefits of semantic segmentation if it is used as a pre-training for encoding, or as an auxiliary task.

6.3.2.1 Semantic Segmentation

We first trained an agent with the semantic segmentation as input. We trained it in the same way as a classical agent using the image, with the difference of the network. The input is indeed much simpler than an image - as it only consists of 5 segmentation masks (road, road lines, vehicles, sidewalk and backgroud). We therefore used a simpler network, namely the network used in [Mnih 2015], which we call NatureCNN, a small network composed of 3 convolution layers. We show on table 6.3 the performance of our agent trained with semantic segmentation as input on navigation task on CoRL benchmark [Dosovitskiy, Ros, 2017]. This first experiment shows the importance of semantic information in generalization. Indeed, training an algorithm with the semantic image as input allows to obtain perfect training and generalization performances, and this shows that the semantic segmentation contains the necessary information for the training. This approach is not useful in practice as perfect semantic segmentation is usually not available in the target environments, but shows that the bottleneck in generalization is mainly on the image analysis part, and not in the driving decision that generalizes perfectly.

Training	Training Condition	New Weather	New Town	New Town & Weather
Perfect segmentation	100%	100%	100%	100%

Table 6.3: Performances with semantic segmentation as input on navigation task on CoRL benchmark.

Since perfect segmentation is in reality never available, a natural solution to this problem is to train a semantic segmentation in the training conditions and rely on its generalization capacity to the new environment. We therefore wanted to test the same agent using a segmentation that we would have trained separately. To train semantic segmentation from RGB images, we used a Linknet [Chaurasia 2018] since it is based on Resnet34 we already use for RL. This training will be then also used as pretraining for the Resnet encoder in the next section. A drawing of Linknet structure is shown in Figure 6.4. Data for training are collected in training conditions only, in

Town01 with training weathers. 140 000 images are used, divided in training and validation sets with the ration 0.8/0.2. Images are collected with three cameras to perform viewpoint augmentation. We used one forward-looking camera, and two with a $\pm 30°$ angle like in [Bojarski 2016]. Using only one front-facing camera for segmentation training and using this segmentation for a RL driving agent resulted in very poor performance (less than 50% in training town). Indeed the RL agent explores a lot, and thus oscillates. If it never learns to segment image that are not perfectly aligned with the road, it will hardly learn how to drive. Moreover having different points of view also helps to manage the crossings. Five categories are trained for the mask output: road, roadlines, vehicles, sidewalk and background. Loss function is categorical cross-entropy, and optimizer is Adam. The learning rate is 10^{-3}, and we used a batch size of 32. Training is made during 100 epochs, and we use classic data augmentation (Gaussian blur, salt and peper noise, random erasing, etc...) during training.

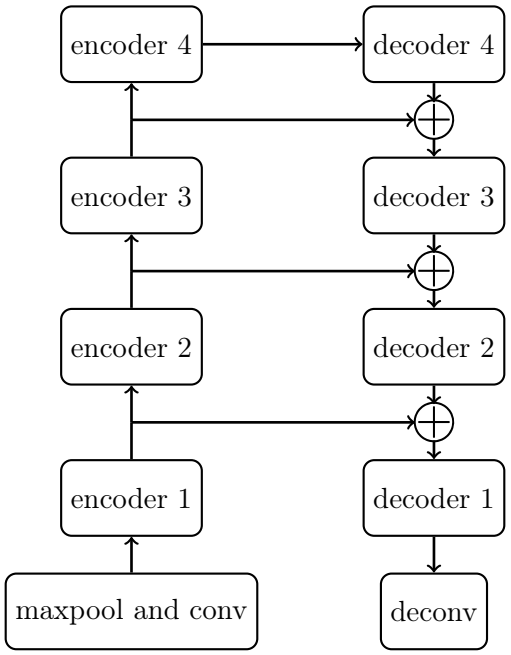


Figure 6.4: Linknet network segmentation architecture [Chaurasia 2018]

We measured the performance of our training by computing the frequency weighted intersection over union (FIoU). Indeed all the categories do not have an equivalent number of pixels in the image (road and road line for instance), therefore weighting the results with the frequency of the class equilibrates the results. IoU and FIoU are detailed in the following equations.

$$IoU = \frac{TP}{TP + FP + FN} \tag{6.3}$$

TP stands for True Positive (number of pixel belonging to the class and being correctly assigned to this class), FP and FN respectively the False Positive (not belonging to the class but assigned to it) and False Negative (belonging to the class and not assigned to it) number of pixels.

$$FIoU = \frac{1}{\sum_i t_i} \sum_{i=1}^{k} t_i * IoU_i \tag{6.4}$$

k is the total number of classes and IoU_i is the IoU for the i^{th} class, and t_i the number of pixel in class i (in groundtruth).

Our training achieved a FIoU of 85% in training conditions, and 82% in new town and weather. An example of a segmented image from the test set is shown on Figure 6.5. We can see that the segmentation is globally good, even if the road line is not perfectly segmented (however the portion of the line close to the agent is good), and the distant cars are not detected.

(a) Original image

(b) Road prediction

(c) Road groundtruth

(d) Roadlines prediction

(e) Roadlines groundtruth

(f) Sidewalk prediction

(g) Sidewalk groundtruth

(h) Vehicles prediction

(i) Vehicles groundtruth

Figure 6.5: Example of segmented test images

We show in table 6.4 the performance of our agent with this learned segmentation, and compare it with the training using the perfect one given by the simulator. The agent using our learned segmentation achieves high but not perfect performance. This approach shows that this is not sufficient to retain the generalization ability of the previous approach. This is also probably an upper limit to the generalization performance that could be achieved with this reinforcement learning approach. Moreover, we saw in the previous chapter that our agent using semantic segmentation as input was not at all robust to a wrong direction command. This is why we think it is important to continue using the image as input, and we will see in the rest of this chapter different methods to incorporate semantic information.

Training	Training Condition	New Weather	New Town	New Town & Weather
Ground truth seg	**100%**	**100%**	**100%**	**100%**
Learned seg	**100%**	**100%**	89%	86%

Table 6.4: Performances with semantic segmentation as input.

6.3.2.2 Pretraining and Auxiliary Task

First, we studied the influence of a good pre-training on the performance of our agent. We decided to continue using semantic segmentation, and used the weights of the segmentation trained encoder - see previous section - to initialize our training. The data augmentation is the same as the one used previously - the semantic data augmentation. We noticed a clear improvement of the performances on the navigation task, reported in the table 6.6. We are not surprised that a good initialization can have a great influence on the results, considering the variation of the performances when changing the random seed (see previous chapter). However, despite a certain improvement in performance, the ability of our agent to generalize remains weak. We have increased its performance in general, but without really reducing the gap between training and test town.

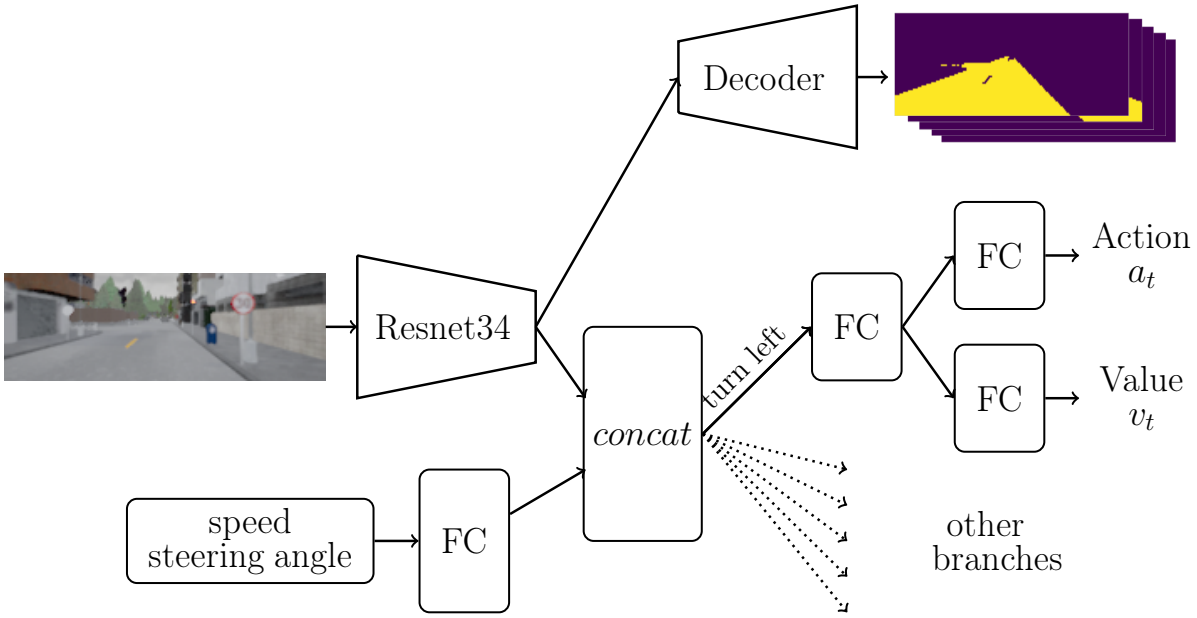


Figure 6.6: Reinforcement learning architecture with auxiliary task

In a second step, we study the effect of adding semantic segmentation as an auxiliary task. Auxiliary tasks consist in training a different task than the main one at the same time in order to improve its performance. In our case, we train semantic segmentation along with driving. The general architecture (see Figure 6.6), as well as the inputs and outputs of the reinforcement training are identical to the previous one (see Chapter 5). The Linknet [Chaurasia 2018] decoder is added to compute the segmentation of the image. The decoder outputs $S_t \in [0,1]^{W*H*c}$ with c the number of classes to segment. For a given pixel on the original image, the output is the probability to belong to the different classes. The 5 classes are the same as the previous training

of semantic segmentation: road, roadlines, vehicle, sidewalk and background. The pedestrians are not taken into account since they cannot cross the street in the Carla version we use (0.9.6). One of the challenge is to combine both learnings at the same time, and not having one prevailing on the other. The simplest loss is a weighted sum of the different losses :

$$l = \lambda_{rl} l_{rl} + \lambda_{aux} l_{aux}. \tag{6.5}$$

In equation 6.5, l_{rl} is PPO loss, l_{aux} the semantic cross entropy loss, and λ_{rl} and λ_{aux} their respective weights. We first tested this simple combination of losses, adjusting the weights using the orders of magnitude of the different losses, but our tests were not conclusive. In [Kendall 2017] is described a more efficient loss combination to perform automatic loss balance in multi-task learning. We tested it and quickly had good results, both trainings, main task and auxiliary task, converged. This loss consists in adding two trainable parameters, which will allow an automatic balancing. We set σ_{rl} and σ_{aux} two trainable parameters, and the loss function becomes:

$$l = e^{-\sigma_{rl}} \times l_{rl} + \sigma_{rl} + e^{-\sigma_{aux}} \times l_{aux} + \sigma_{aux}. \tag{6.6}$$

This loss is used in all the following experiments. In our tests in this section, 3 of the 4 blocks of the encoder are common between the segmentation task and the navigation task, the 4th and last block is duplicated and task-specific. We kept the skip-connections of the Linknet during the training with auxiliary task (see Figure 6.4 to have a global overview of Linknet architecture). We did some experiments with the skip connections removed, and the results were about the same. We did not pursue the study and decided to keep the skip connections. However, we have removed the batchnorms in the architecture (there are some in the Linknet).

We first tested the addition of auxiliary task on the circuit - used in particular to adjust the loss balance. The results with and without the auxiliary task - and always with badly parked cars as obstacles - are shown in the table 6.5. We can see an improvement in performance by adding the auxiliary task on the test track, with an average speed closer to the target speed, an agent closer to the center of the road, and less away from its trajectory. The agent also travels more distance on average per episode, and the variance of the measurements is lower overall. We note, however, that our agent is slightly worse on the training track, which is often the case when we try to reduce overfitting. Moreover, the low generalization capacity in our circuit is probably also due to the fact that the pattern is very repetitive with a not very long circuit.

	Training Condition	New Weather	New Track	New Track & Weather
speed (km/h)	36.4 (7.8)	**36.9** (6.3)	29.0 (9.2)	29.5 (8.3)
cte (m)	0.2 (0.3)	0.2 (0.3)	4.8 (5.9)	2.9 (3.3)
atr (°)	1.7 (1.5)	1.7 (1.3)	30.8 (33.7)	25.0 (28.0)
dist (m)	**978.7 (274.3)**	**881.1 (332.7)**	350.8 (468.8)	359.7 (462.1)
collision	**3/3**	**10/36**	**22/36**	**25/36**

(a) Baseline

	Training Condition	New Weather	New Track	New Track & Weather
speed (km/h)	**37.3 (5.3)**	**36.8 (5.2)**	**34.1** (9.6)	**36.1 (6.9)**
cte (m)	**0.2 (0.1)**	**0.2 (0.1)**	**2.2 (4.8)**	**1.9 (2.6)**
atr (°)	**1.4 (0.9)**	**1.7 (1.1)**	**7.2 (10.3)**	**7.5 (7.8)**
dist (m)	874.0(329.1)	770.3(397.6)	**481.6 (401.3)**	**541.1 (406.9)**
collision	11/36	15/36	30/36	**25/36**

(b) Auxiliary task

Table 6.5: Comparison of results at 10 million steps between baseline and learning with auxiliary task. The figures presented are the mean (and variance) on different measurements

Finally, we applied the auxiliary task to the navigation task on our urban training (the extension to dynamic navigation will be seen in the next section). In the table 6.6, we can compare the results of our baseline, of the agent pre-trained with segmentation, and of the agent with auxiliary task.

We also performed some experiments by adding regularization and batchnorm, both during the segmentation pre-training and RL training, and we add the results to the table 6.6. The L2 regularization was only added in the CNN. We indeed found that if regularization is added in the branches, the network has trouble learning, because regularization takes place at each iteration, while branch update is more sparse. We did 2 separate tests with the batchnorm. The first one consisted in fixing the batchnorm layers after the pretraining, and the second one in continuing to train the batchnorm layers. We noticed a surprising drop in performance when continuing to train the batchnorm, contrary to what we expected. We did not extend our tests with batchnorm and regularization because the first results were disappointing, i.e. they were not improving the generalization of the driving agent.

Training	Training Condition	New Weather	New Town	New Town & Weather
Baseline with data aug	67%	60%	34%	28%
Pretraining	82%	98%	49%	40%
Pretraining + reg L2	86%	98%	52%	32%
Pretraining + batchnorm fixed	86%	**100%**	55%	28%
Pretraining + trainable batchnorm	40%	34%	30%	16%
Auxiliary task	**90%**	92%	**78%**	**68%**

Table 6.6: Pretraining and auxiliary task performance

We can see that the auxiliary task allows to obtain high performances, but especially to reduce the gap between the training and test town. The agent learns at the same time to distinguish the different semantic elements of the image, and to map them with positive or negative rewards.

6.3.3 Application to Dynamic Navigation

In order to better compare ourselves to the state of the art, we applied these results to dynamic navigation, i.e. by adding moving obstacles. It should be noted, however, that we use Carla version 0.9.6 without further modifications, and that in this version pedestrians do not cross the street.

To add more vehicles, we had to make some modifications. The first is the adjustment of the target speed as in [Toromanoff 2019]. Indeed, without modifications, we found that our agent tries to pass the vehicle stopped in front of him, either by the opposite lane or by the sidewalk (as there is no difference in the reward, only the distance to the center of the road matters). In addition to being an undesirable behavior, we found the problem raised in chapter 4, namely that our agent having only a front camera and no memory, it pulls back too fast and hits the obstacle it is overtaking. We therefore modified the target speed in the reward function, using a target speed linearly decreasing to zero according to the distance to the vehicle in front. We also placed a lot of vehicles around our agent at the beginning of each episode, rather than randomly placing them around the training town, to make sure it sees enough. We also chose to give our agent an initial speed $\in [0, 40] km/h$ so that it quickly learns to brake. However, to continue learning to navigate, only half of the simulators have dynamic agents.

We noticed an increase in convergence time - which seems normal when increasing the task complexity. In the circuit, our algorithm converges in less than 10 Million steps. For the navigation task, the training often converged in 15 Million steps. For dynamic navigation, the highest performances are reached around 18-20 Million iterations.

We trained 3 different types of models. The first one is with semantic segmentation as input, as a baseline. The other two are both with an auxiliary task, and different simply by the number of common encoder blocks between the navigation task and the auxiliary segmentation task. The encoder, Resnet34, has 4 distinct blocks. Our previous experiments used 3 blocks in common out of the 4. We experimented here with having 3 blocks, or 4 blocks - and the whole encoder - in common between navigation and segmentation. Due to long training times (around 7 days for full dynamic navigation task) combined with computational expense of our training (16 Carla simulators are used in parallel to collect data in one training), we only launched 2 trainings of each architecture. Our models are evaluated on Carla's classic CoRL benchmark, but also on the more complex NoCrash benchmark.

Task	Training Condition	New Weather	New Town	New Town & Weather
	CoRL 2017			
Navigation	98	98	96	98
Dynamic navigation	91	98	96	98
	No Crash			
Empty	99	100	95	94
Regular	92	92	85	88
Dense	70	68	69	72

Table 6.7: Benchmark results with semantic segmentation as input

In the table 6.7 are shown the results of the segmentation alone, used as input. It can be seen that the results are high, and obviously very stable from the point of view of generalization. It can be seen that in the most complex conditions, i.e. navigation in dense terrain, the performance peaks at 70%, which suggests a maximum achievable with our method.

In both tables 6.8 and 6.9, we can see the results of our training with auxiliary task, with 3 or 4 common blocks in the encoder. We can see that even if the addition of the auxiliary task has reduced the variations between the trainings, they are still present. The evaluations are launched with the same random seed. We can see that the two apprenticeships with 3 blocks out of the 4 in common with the auxiliary task have quite similar results. On the two other trainings with the common complete encoder, the performances are more different. One of the two has lower performances - the training does not look finished because the performances increase with the previous checkpoints while the curve seems stable. The second has high performance on the CorL benchmark, but performance drops on the NoCrash benchmark, especially on the test town with new weather.

Task	Training conditions		New weather		New town		New Town & weather	
	CoRL 2017							
	training 1	training 2	training 1	training 2	training 1	training 2	training 1	training 2
Navigation	84	79	84	80	73	75	58	44
Dynamic nav	84	78	88	78	69	67	52	46
	No Crash							
	training 1	training 2	training 1	training 2	training 1	training 2	training 1	training 2
Empty	78	72	92	74	71	67	28	30
Regular	73	72	80	74	64	61	22	18
Dense	73	61	62	60	55	42	12	8

Table 6.8: Auxiliary task, 3 blocks encoder

Task	Training conditions		New weather		New town		New Town & weather	
	CoRL 2017							
	training 1	training 2	training 1	training 2	training 1	training 2	training 1	training 2
Navigation	29	100	34	96	30	79	20	70
Dynamic nav	35	86	36	84	34	60	26	42
	No Crash							
	training 1	training 2	training 1	training 2	training 1	training 2	training 1	training 2
Empty	12	94	22	64	28	83	8	4
Regular	19	89	18	46	28	73	8	4
Dense	27	78	16	30	38	56	4	0

Table 6.9: Auxiliary task full encoder

To conclude this section, we compare our best trainings with the state of the art methods using reinforcement learning on CoRL benchmark [Dosovitskiy, Ros, 2017] - and not on NoCrash benchmark since all these methods do not have results on it. Comparison is shown on table 6.10. It is important to note that we are the only ones, with the exception of [Dosovitskiy, Ros, 2017] that achieve very low performance, to perform end-to-end RL training with image input. CIRL [Liang 2018] uses supervised learning as pretraining, WRL [Agarwal 2019] uses segmented birdview images, and MarLn [Toromanoff 2019] pretrains the encoder with semantic segmentation and other affordance then freezes it during RL training.

We can compare our training with semantic segmentation as input with WRL [Agarwal 2019], and our method outperforms the latter, especially since we are using an on-board camera and not a top view like [Agarwal 2019]. However as mentioned in chapter 5, using pure segmentation seems to be not robust, and would require in any case to have access to this semantic information.

Our models with input image present good results, far surpassing the RL of [Dosovitskiy, Ros, 2017], and having results comparable or even slightly superior to CIRL [Liang 2018], knowing that no demonstration has been given to our agents. However, they remain inferior to those of MaRLn [Toromanoff 2019], which is possibly due to two major differences: the use of an off-policy RL algorithms, which allows to focus on more difficult passages, and the separate training of the encoder, with thus two separate optimizations, whereas we use an on-policy algorithm with an end-to-end training.

Training	Task	Training conditions	New weather	New town	New Town & weather
RL	Navigation	14	2	3	6
[Dosovitskiy, Ros, 2017]	Dynamic navigation	7	2	2	4
CIRL	Navigation	93	86	53	68
[Liang 2018]	Dynamic navigation	82	80	41	62
WRL	Navigation	99	99	94	94
[Agarwal 2019]	Dynamic navigation	79	79	60	60
MaRLn	Navigation	**100**	**100**	**100**	**100**
[Toromanoff 2019]	Dynamic navigation	**100**	**100**	**98**	**100**
Seg input	Navigation	98	98	96	98
	Dynamic navigation	91	98	96	98
Auxiliary Task	Navigation	**100**	96	79	70
Full encoder	Dynamic navigation	86	84	60	42
Auxiliary Task	Navigation	84	84	73	58
3 blocks encoder	Dynamic navigation	81	88	69	52

Table 6.10: Comparison with algorithms from state of the art using reinforcement learning on CoRL benchmark on both navigation and dynamic navigation

6.3.4 Preliminary Results with Ensemble Averaging

In this last section, we want to propose an opening and preliminary results on the ensemble average. Ensemble methods usually help reduce overfitting since they benefit of local optima. And as, because of our choice of methods, we have trained many models, we want to take advantage of it by averaging them. We propose here two tests: checkpoint average, and model average.

Checkpoint average, as proposed in *Checkpoint ensembles: Ensemble methods from a single training process* [H. Chen 2017], consists in averaging the output of checkpoints from the same training. Used in [Toromanoff 2019], the authors indicate that this method reduces oscillations and improves the performances. Between 3 and 8 checkpoints are used (3 in the paper, 8 in the git repository[1]). We have selected one of our model trained with auxiliary task in a dynamic environment, and average the output of 6 checkpoints (from 15 to 20M). At each iteration, the action used is the average of the actions calculated by the models from each checkpoint. We report in table 6.11 the results and compare them to the results of the two best checkpoints we measure (18 and 20M).

[1] `https://github.com/valeoai/LearningByCheating`

Task	Training conditions			New weather			New town			New Town & weather		
	18M	20M	avg 15-20M	18M	20M	avg 15-20M	18M	20M	avg 15-20M	18M	20M	avg 15-20M
Empty	66	**72**	64	74	74	**76**	**70**	67	61	24	**30**	18
Regular	**74**	72	73	**76**	74	74	61	61	**69**	**22**	18	20
Dense	66	61	**67**	56	60	**62**	**46**	42	43	**18**	8	16

Table 6.11: Checkpoint average performance on NoCrash benchmark. We compare the performance of two checkpoints (18M and 20M steps), with the average of 6 checkpoints: from 15M to 20M - every 1M.

One can see in these figures little or no improvement in performance. This suggests that in our case, the model converges to a distribution and that the successive checkpoints, although quite distant, are part of the same distribution.

Model average [Dietterich 2000] is more general. It consists in averaging the outputs of different models - usually trained on the same data. It is classical to average identical models, i.e. having been trained in the same way, but we had few that were performing good enough. We chose to test our three best dynamic navigation models. In the table 6.12 we show the performance of each model separately, and their average. We can observe a clear improvement in performance, especially in situations where all three models are struggling (new town and new weather). It is interesting to note that the agent resulting from this average tends to draw the best from each. We can conclude here that the models are not in the same local optimum. An immediate extension to this work would be to average more models, and average identical models. It would be interesting to see if models trained in the same way - with 3 encoders for example, tend to go to the same local optimum or to different ones. However, this would require a large number of models, and thus many computing resources.

Task	Training conditions				New weather				New town				New Town & weather			
	CoRL 2017															
	1	2	3	avg	1	2	3	avg	1	2	3	avg	1	2	3	avg
Navigation	**100**	84	79	95	96	84	80	**100**	79	73	75	**85**	**70**	58	44	62
Dynamic navigation	86	81	78	**95**	84	84	78	**94**	60	69	67	**87**	42	52	46	**60**
	No Crash															
	1	2	3	avg	1	2	3	avg	1	2	3	avg	1	2	3	avg
Empty	**94**	78	72	**94**	64	92	74	**100**	83	71	67	**86**	4	28	30	**38**
Regular	**89**	73	72	**89**	46	80	74	**94**	73	64	61	**86**	4	22	18	**38**
Dense	**78**	73	61	74	30	62	60	**84**	56	55	42	**66**	0	12	8	**22**

Table 6.12: Model average results on both CoRL and NoCrash benchmark. First model is auxiliary task with full common encoder, second and third are auxiliary task with 3 blocks over 4 common between navigation and semantic segmantation. Last column is the average of the output of the 3 models.

6.4 Discussion

In this chapter we have studied different methods to reduce overlearning with reinforcement learning applied to autonomous driving. We have analyzed the influence on the performances, and in particular on the generalization, of different methods based on semantic segmentation. Indeed, semantic segmentation allows an agent to learn to drive and to generalize to roads it has

not seen. However, relying solely on the generalization capability of segmentation is to separate perception and control, which is not in the spirit of end-to-end driving. We therefore experimented with different ways of adding this segmentation information: in the data augmentation, through pre-training, and finally with an auxiliary task. We have shown that only the latter method reduces the gap between training conditions and test environments. Pre-training or applying a rigorously chosen data augmentation allows us to improve the performance of our agent in general, but does not solve the generalization problem. Simultaneous learning the image segmentation as auxiliary task, by sharing part of the network with driving task, on the other hand improved generalization. What the addition of segmentation seems to indicate is that the agent learns to match the elements observed in the input image with the rewards. One could imagine going further, for example by adding more data augmentation, varying the camera position to change the point of view, or even using different environments - more cities in Carla, or even completely different environments.

One might also wonder what impact the quality of the learned segmentation has. Recent work, *Label efficient visual abstractions for autonomous driving* [Behl 2020], seems to suggest that a weaker annotation (coarse 2D boxes around the segmented object rather than a precise contour) - and thus less expensive - would be just as effective in terms of driving. It would be interesting to test this weaker segmentation as an auxiliary task to study if the same performance can be achieved.

We also applied the segmentation auxiliary task to the dynamic driving task, testing two different levels of layer sharing in the process. We can imagine as future work to test more in depth different architecture and their influence on the results - for example the skip connections of the linknet, to study in more detail the level of information shared between the main navigation task and the auxiliary task. We have not made attemps to modify the general architecture when it was set, but very recent work [Raileanu 2021] seems to suggest that decouple policy and value leads to better generalization. We can also imagine comparing the relative performance of adding different auxiliary tasks. We have focused only on segmentation, essentially due to limited time, but we can think about other driving related tasks like depth or next frame prediction, optical flow computation, direction or waypoints prediction, etc ... An interesting extension to our work would be the comparison of the simple or combined addition of these different auxiliary tasks.

Finally, we have also tested to average either checkpoints of a same model or dinstinct models. We found that using checkpoints from the same training does not improve the performance or only slightly, which indicates that the model does not oscillate between several minima. On the other hand, averaging different models led us to obtain better results than each model taken separately.

To conclude, we compare our best results, i.e. those of model averaging, with the two state of the art methods, LBC and MaRLn. In our comparison with the state of the art algorithms in table 6.13 we re-run their evaluation without pedestrians for a fair comparison (using implementation from both [2] and [3]).

[2] `https://github.com/dotchen/LearningByCheating`
[3] `https://github.com/valeoai/LearningByCheating`

Task	Training conditions			New weather			New town & weather			New Town		
	LBC	MarLn	Ours	LBC	MarLn	Ours	LBC	MarLn	Ours	LBC	MarLn	Ours
	CoRL 2017											
Navigation	**100**	**100**	95	**100**	**100**	**100**	98	**100**	85	**100**	98	62
Dynamic navigation	**100**	**100**	95	**100**	**100**	94	**99**	98	87	**100**	**100**	60
	No Crash											
Empty	97	**100**	94	84	14	**100**	**100**	98	86	**68**	26	38
Regular	99	**100**	89	90	16	**94**	**99**	**99**	86	**68**	28	38
Dense	92	**93**	74	**86**	18	84	**90**	79	66	**52**	20	22

Table 6.13: Comparison with SOTA on CoRL and NoCrash benchmarks

First, we see that none of the approaches achieves 100% success on the most complex benchmark, namely NoCrash. Moreover, LBC seems to have results that generalize better to conditions that have not been seen. These methods are very diverse and show that there is still room for improvement because in the most complex situation (dense traffic and new town and weather), the best of the agents only achieves its goal in just over half of the cases. Our agent, the only one that was trained end-to-end with reinforcement learning, shows promising results, which despite a difficulty to generalize, remains relatively homogeneous in this benchmark.

Chapter 7

Conclusion

7.1 Summary

In this thesis, we focused on the application of reinforcement learning to driving in an urban environment. Most of the existing works in the literature on end-to-end autonomous driving use imitation learning, or at least a separate perception module trained in a supervised manner. In this thesis, we have chosen to perform end-to-end training with reinforcement learning. In particular, we studied several aspects of autonomous driving: robustness in case of an erroneous directive, and the ability to generalize to an unknown environment.

After having initially set up the elements for an autonomous agent to be able to follow a road using only one embedded camera like a human driver, we first addressed the management of crossings by our autonomous agent. We studied how to give a direction command to an agent to switch from lane following to conditional driving. We also proposed a benchmark that evaluates the robustness of an autonomous agent. Comparison with state-of-the-art algorithms shows that reinforcement training improves the reaction of agents to erratic high-level commands, and that learning to segment the input image in parallel improves the robustness of the agent by giving it an understanding of its environment. Another essential element that we have identified for a robust agent is information redundancy. We found that an agent that has learned to use all the elements available is more robust. Eventually, our results suggest that an agent is more robust when perception and control are not decoupled during training.

Finally, we studied the problem of generalization to an unknown environment. We have been particularly interested in the addition of semantic information during learning, and the relative benefits depending on how it is added. We have shown in a first step that segmentation contains the necessary elements for driving in urban environment, and is thus particularly adapted to autonomous driving. We then showed that the addition of segmentation as an auxiliary task, i.e. learned in parallel with driving, by sharing part of the neural network, largely improved the generalization capacity of our autonomous agent.

7.2 Limits

In the course of our work, we identified several limiting factors.

The first is the variability of our different trainings. Training differing only in random seed could lead to completely different autonomous agents, and thus to very different performances. Even if the addition of a supervised auxiliary task reduced this variability a little, it is still present and makes the analysis of the results more complex.

The second issue is the training time and the resources required per training. This is linked to

our choice of algorithms, and in particular to the on-policy factor. Coupled with the variability of learning, having to perform a large number of long and resource-consuming trainings was a substantial limiting factor for this work.

Finally, we must also mention the limits of the simulator. Although very powerful and offering many complex situations, it is recent and still under development. Moreover, the comparison between works realized on distinct versions of the simulator is not entirely relevant because of the difference in graphics. It should also be noted that the autopilot embedded in the simulator is far from perfect. We can see on the figure 7.1 that our agent is blocked by a collision, and the autopilot barely reaches 60% of completed episodes on certain tasks on the most complex benchmark (NoCrash).

Figure 7.1: Our driving agent is blocked due to a collision of Carla's vehicules.

7.3 Future work

The work of this thesis opens the way to many perspectives and avenues of reflection.

Use of more off-policy data. We have indeed found that our agent has difficulty in handling the rare cases that can occur. This is due to the choice of PPO, which is on-policy, and treats all data in the same way. However, when driving, the most common cases are driving straight ahead, but the most complex moments are changes of direction, or the presence of other agents where we have to brake. To counterbalance this limitation of PPO, we artificially varied the situations seen, by placing our agent close to crossings, and by adding many vehicles around it. However this requires a lot of pre-processing, and it is difficult to consider all situations - and that is also only realizable in simulation. A natural solution would be to directly change the algorithm to an off-policy algorithm, like SAC. However, on-policy algorithms are in theory more stable, because the update is done with data collected by the current policy. Moreover, the data is stored in a replay buffer which can be very large in terms of size if the input used is the image. To keep the on-policy side of PPO, one solution would be to change the occurrence of the data during the update. PPO works as follows: a batch of data (state-action-reward) is collected. This batch is shuffled to make the data independent, then separated into mini-batches, and the update is done with these mini-batches. One could imagine that instead of taking all the data from the batch, the mini-batches are formed by picking the data using probability. The more complex the data (e.g. low reward obtained, or low value), the higher the probability. Another close solution would be to copy some data so that they appear several times, balancing

the batch by duplicating the rarest data. However, this solution has limits, because it only allows to balance within a single batch collection, and not on all the data seen during the learning process

Extension of auxiliary tasks. We found that adding an auxiliary segmentation task greatly helps an autonomous agent to learn, and in particular to relate rewards to the semantic elements around it. An interesting perspective would therefore be to test other driving-related auxiliary tasks, such as depth calculation, optical flow, or next image prediction, as well as their combination. In the context of a transfer to the real world, the possibility of using these auxiliary tasks on real vehicles should be considered. One can either use an existing module producing imperfect information - such as segmentation or object detection, or use an automatic annotation (of the depth by a Lidar for example), or use a task that does not require any annotation (e.g. next frame prediction).

In-depth study of perception and control coupling. In our work, we have chosen to train our agent from end-to-end, while many works decouple perception and control. The study on robustness that we carried out suggests that the coupling, and thus the use of a single optimization, increases the robustness of the autonomous vehicle. When perception is trained separately from control task, the encoder is trained with different tasks - detection, segmentation, etc, that are not directly the driving task - and the encoder has then never been optimized for driving. It would be interesting to train two different agents in the same way, using the same algorithm and the same hyperparameters, one by pre-training and fixing the encoder with a certain number of tasks, and the other by learning these same tasks in an auxiliary way, and to test these agents both for their performance, their ability to generalize, and their robustness to erroneous commands.

More environments. Generalization to an unseen environment remains an open problem. While humans are able to adapt to big changes, a significant change of its environment can cause a drastic loss of performance for an intelligent system. In particular, humans are able to handle a new environment (e.g. video game) with different graphics - because we identify semantic objects even if they are visually different from those we are used to. One could imagine training an autonomous agent in several completely different environments - using several different simulators, not just different environments of the same simulator. Coupled for example with semantic segmentation auxiliary task, this may build agents that have more in-depth understanding of the environment and that can adapt more easily to a new one. More specifically, it could also ease the transition from simulation to the real world.

Intrinsic reward. Another perspective would be to exploit intrinsic rewards. Based on the idea that an agent learns better if it is intrinsically motivated [Singh 2005], the intrinsic rewards do not come from the environment (like the distance to the center of the road, collision, etc.), but from the agent itself. One can think of curiosity to improve exploration. Adding intrinsic motivation could reduce training time, however, it seems difficult to perform autonomous driving without environmental rewards, but intrinsic rewards could be combined with external rewards.

Appendix A

Methods to Improve RL Training

Reinforcement learning is known to have a number of drawbacks, such as long training time and high computational expense. If learning starts from scratch, with no prior information, the agent must test all options to figure out the good ones. Therefore, accelerating learning is a major and still open problem. Stability is also a sensitive issue in reinforcement learning because of the high degree of randomness in training process. In this section we will present several research directions that can be used to help RL training.

A.1 Fine-tuning

In supervised learning, fine-tuning is a very popular technique. It consists in training the agent of the source domain, and retraining it on the target domain. In perception tasks for instance, a lot of models are first pretrained on image classification tasks, to develop feature representation, and then re-trained on new task, and show very good results. Fine-tuning is however less commonly used in reinforcement learning. Indeed RL is based on the principle of trial and error, and on its own exploration of the environment. If the agent is guided, the very principle of reinforcement learning is tampered.

Some, however, have used supervised pretraining before reinforcement learning training. In [Sadeghi 2016], model is pretrained to predict the collision probability. The goal is to teach an agent how to fly a drone. Training is made in simulation using CAD models from real images using [Izadinia 2016]. The agent is pretrained to predict the collision probability, and fine-tuned with RL on navigation task. In simulation, CAD^{2}RL presents a noteworthy improvement of the results obtained with Imitation learning, and in real environment the results are pretty interesting, as the drone is able to adjust its height to go through a window, or fly up stairs. Pretraining the agent can speed up a lot training, and still allows to explore, but only starting from the pretrained weight.

In this case the pretraining is made on a different task (collision probability), however it can also be made with the same task. It is the case of both [Liang 2018; Pfeiffer 2018] for navigation task. In this case the agent is pretrained with supervised learning and finetuned with reinforcement learning, and show improved performance compared to supervised or reinforcement learning alone. However using supervised learning drastically reduce the exploration of the agent, and still requires a large amount of annotated data.

A.2 Combining Reinforcement Learning with Demonstration

Adding expert demonstration during reinforcement learning training is a technique that allows to guide the agent. Training still starts from scratch, and demonstration are added during training. Intuitively it can be more powerful than simple pretraining since it does not constrain the exploration, as reinforcement learning starts at the beginning of the training, while still giving a guidance during training. Indeed one main problem of RL is the exploration part: if the exploration is not enough, the agent may never get good rewards, and thus never learn. Therefore it can be interesting to add good demonstration, combining reinforcement and supervised learning, so that the agent can know in which direction the rewards are higher.

Different methods are used to take into account the demonstrations. Demonstrations can be added to the replay buffer like [Vecerik 2017; Gulcehre 2020], and pulled with high probability thanks to the prioritized experience replay [Schaul 2016] (see Figure A.1). Demonstrations can also be processed separately, using a separate supervised loss [Rajeswaran 2017; Lakshminarayanan 2016; Hester 2017]. However, work has shown that training in the same way, i.e. with the same loss, the supervised transitions and those collected during the training allows to be more robust in case of a poor quality demonstration (cf. *Reinforcement Learning from Imperfect Demonstrations* [Gao 2018])

Finally, demonstrations can be incorporated into the reward like in *DeepMimic: Example-Guided Deep Reinforcement Learning of Physics-Based Character Skills* [Peng, Abbeel, 2018]. The agent is taught to perform physical tasks on simulation (like spinkick or cartwheel), and is given reference motion. The reward function is composed of two terms: the first one encourages the agent to follow the reference motion, and the second to perform a specific task (hit a ball). This allows the agent to learn to perform some specific task that were not given as example while being guided by a reference movement.

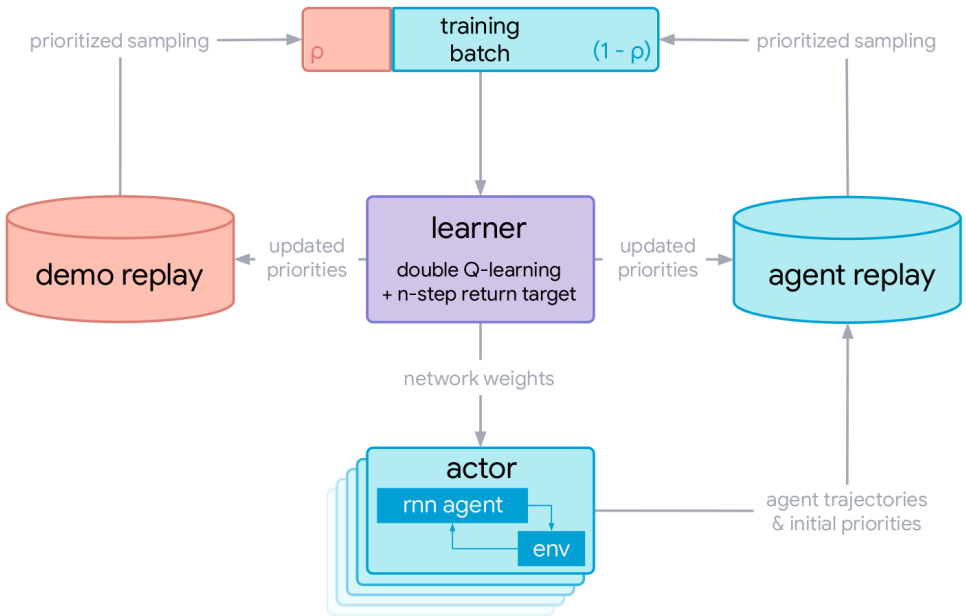


Figure A.1: Prioritized experience replay with two separate buffers: one for expert demonstrations, one from the learning agent [Gulcehre 2020].

A.3 Progressive Networks

Developed to overcome the failures of fine-tuning like catastrophic forgetting, progressive networks [Rusu 2016] operate by freezing previously learned network, and training a new network on a different task using lateral connections to the previously learned network (see Figure A.2). General idea is that previously learned skills can apply to new tasks. In *Sim-to-Real Robot Learning from Pixels with Progressive Nets* [Rusu 2017], Rusu et al. use progressive networks to move from simulation to real world for vision based robot grasping task, trained with reinforcement learning. They show noteworthy improvement of the performance with the task in real life with progressive networks compared to simple fine-tuning. In this work it is used to bridge the gap between simulation and reality, and we can imagine using it to achieve different level of driving (lane following, obstacle avoidance, then conditional driving, etc...).

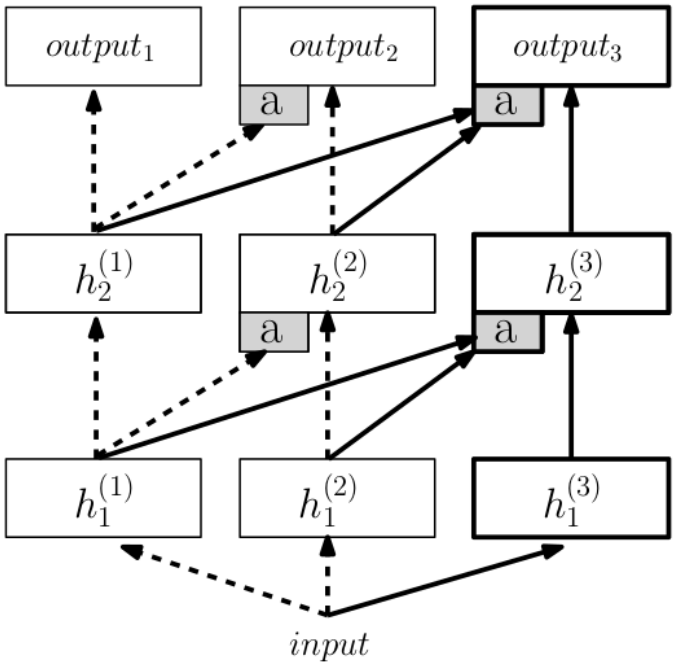


Figure A.2: Progressive network [Rusu 2016]. Each network is represented in column, representing a specific task. When learning a new task, the other column parameters are frozen, and the new column corresponding to the new task is randomly initialized. Transfer can apply from previous columns laterally: hidden parameters h_i^2 depend from both hidden parameters h_{i-1}^1 and h_{i-1}^2, i representing here the i^{th} layer of the network.

A.4 Multi-Task, Auxiliary Task and Intrinsic Rewards

Traditional learning problems learn one task at a time. For complex problems, this means separating them into independent tasks that are learned separately. In the case of driving, one can think of road detection, then trajectory planning, etc... which, in the case of the modular pipeline approaches, are learned separately. However, in the case of complex systems, training several correlated tasks at the same time enables knowledge to be shared between the different tasks, and to learn and generalize better at the same time. We speak of multi-tasks when the tasks that are learned have the same importance, on the other hand we speak of auxiliary tasks when

this is not the main learning objective but helps for performance. A lot of work on multi-task learning focuses on the learning of an agent able to master the different Atari games [Rusu 2015; Parisotto 2016; Hessel 2019].

Both *Policy Distillation* [Rusu 2015] and *Actor-Mimic: Deep Multitask and Transfer Reinforcement Learning* [Parisotto 2016] use several single-game expert DQNs to transfer knowledge to a single DQN using distillation [Hinton 2015]. They show improved performance compared to single-task expert and DQN trained simultaneously, and improved generalization. Distillation is also used in [Teh 2017], with one common policy interacting with task specific policies (see Figure A.3). Task specific policies distill knowledge into the common policy, and in return the common policy guides task-specific policies through Kullback-Leibler divergence for regularization. In [Hessel 2019], PopArt normalization [Van Hasselt 2016] is used to manage the different environments, and in particular the rewards, whose order of magnitude can vary from one environment to another.

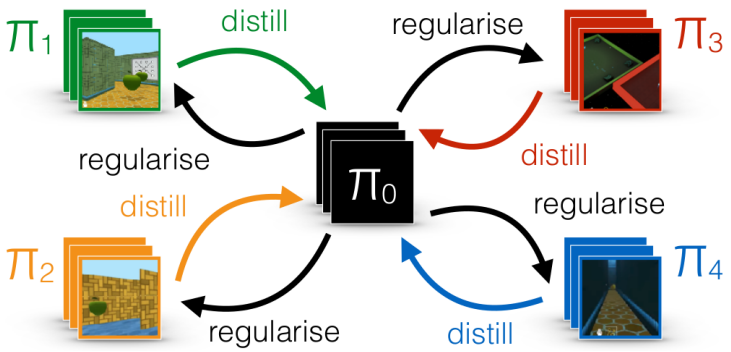


Figure A.3: Distral (distill and transfer) framework [Teh 2017]. Common policy π_0 gets knowledge from task specific policies π_i, and guide them using KL regularization.

We saw earlier that several approaches use auxiliary tasks to help learning the navigation task. These auxiliary tasks are complementary tasks to navigation, whether it is the prediction of affordances [Mehta 2018], depth [Mirowski 2016; Hawke 2019; Z. Li 2018], segmentation [Hawke 2019; Z. Li 2018], loop closure [Mirowski 2016], or internal information that do not require additional information such as image reconstruction with a VAE [Kendall 2018] or self-supervision [Jaderberg 2016; Shelhamer 2019].

Rather than adding an extra loss, it is possible to use intrinsic rewards. Intrinsic rewards are a way to add rewards when the environment produces only very sparse rewards, and can help exploration. In [Pathak 2017] is introduced Intrinsic Curiosity Module (ICM) (see Figure A.4). The Intrinsic Curiosity Module produces a reward according to the agent capabilities to measure the consequence of its own actions. Both [Zhelo 2018; Pathak 2017] use the ICM to encourage exploration. The addition of intrinsic reward allows better data efficiency without adding external information.

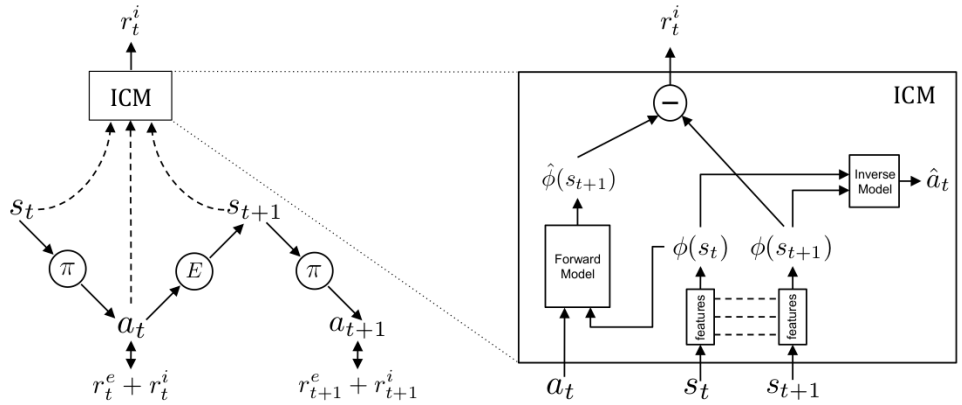


Figure A.4: Intrinsic Curiosity Module (ICM) [Pathak 2017]. The agent interacts with the environment, and consecutive states s_t and s_{t+1} are send to the ICM unit, along with the chosen action a_t. Features $\phi(s_t)$ and $\phi(s_{t+1})$ are computed from the states, and a forward model computes an estimation $\widehat{\phi}(s_{t+1})$ of $\phi(s_{t+1})$ from features $\phi(s_t)$ and action a_t. The difference between $\widehat{\phi}(s_{t+1})$ and $\phi(s_{t+1})$ is the intrinsic reward.

A.5 Meta Learning

In practice very close to multi-tasks learning, meta learning, also called "learning to learn", is a branch of learning that enables more flexibility between different environments. The training set is now part of the learned function f: $f(D_{train}, s) \rightarrow a$, a task to perform now becomes a training data. We present here a few works that we think are relevant, but there are many more [J. X. Wang 2016; Finn and Levine 2017; Mishra 2018; Finn, Yu, 2017; Schwarz 2018; Duan 2016; Finn, Abbeel, 2017]. Based on the observation that learning from scratch takes a really long time, and that animals and humans can learn new task very fast from prior understanding of the world, Duan et al. propose an approach to speed up learning in MDP. In *RL^2: Fast Reinforcement Learning via Slow Reinforcement Learning* [Duan 2016], Duan et al. formulate the principle of learning a RL algorithm as a RL algorithm itself (which leads to the name RL^2). The algorithm will be trained across a set of MDP, and not only one MDP as in classical RL. Figure A.5 shows the principle of the RL^2 algorithm. A trial is defined as a series of episodes with a given MDP. The MDP is fixed inside a trial, but varies from one trial to another. Inside a trial, several episodes (n) are performed (here 2 episodes per trial). The agent consists in a Recurrent Neural Network (with hidden state h) with Gated Recurrent Units, trained with Trust Region Policy Optimization [Schulman, Levine, 2015].

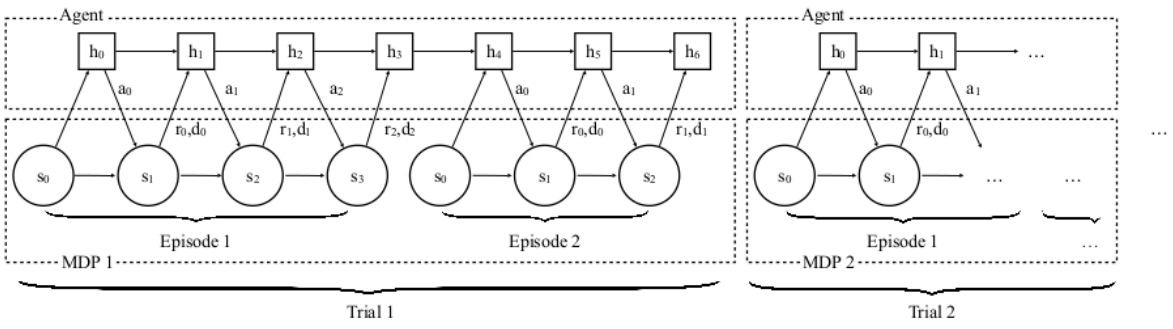


Figure A.5: RL^2 algorithm [Duan 2016]

The RL^2 algorithm is tested on visual navigation on a maze and show a high success rate.

In *Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks* [Finn, Abbeel, 2017], the authors prepare a model for quick adaptation to new tasks. The idea of the MAML algorithm is to optimize the reward after one or more gradient step. The idea is not to be perfect on all tasks, but to find a "mean" policy that can quickly adapt to all of them. The idea is represented in Figure A.6.

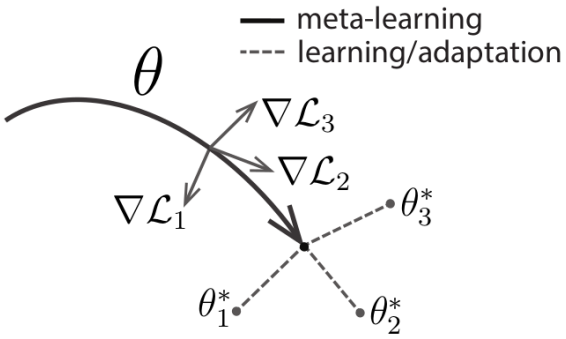


Figure A.6: Model-Agnostic Meta Learning (MAML) framework that optimizes a policy capable of fast adaptation to new tasks [Finn, Abbeel, 2017]

The MAML algorithm is first evaluated on a regression task for sine function, and shows pretty good results compared to finetuning, then on image classification on both Omnilog [Lake 2013] and MiniImageNet [Vinyals 2016] datasets. The evaluation is performed as follow: the MAML network is trained on several classes of images, and then tested on 5 or 20 new classes, with 1 or 5-shot learning, meaning the network is provided 1 or 5 images to adapt to the new class, and then evaluated on test images on the new classes. In all cases, MAML performs better than state-of-the-art classification few-shot of meta-learning algorithms. MAML is also evaluated on a reinforcement learning task: locomotion tasks of half-cheetah and ant (direction and velocity), and the model is shown to adapt very quickly to new task (i.e. new direction or speed), much better than pretraining, and than random initialization. With a single gradient step on the new task, the model is capable of performing very well on that task.

Meta learning works by training an agent on a set of similar tasks, so that it captures the heart of the problem. In our case, we can imagine training an agent to drive in several different environments, with different viewpoints, or even with several different simulators, so that it is able to understand the driving problem, and then to better generalize.

Bibliography

[Adam 2012] Adam, S., L. Buşoniu, and R. Babuška (2012). "Experience replay for real-time reinforcement learning control". In: *IEEE Transactions on Systems, Man and Cybernetics Part C: Applications and Reviews* 42.2, pp. 201–212. ISSN: 10946977. DOI: 10.1109/TSMCC.2011.2106494.

[Agarwal 2019] Agarwal, T., H. Arora, T. Parhar, S. Deshpande, and J. Schneider (2019). "Learning to Drive using Waypoints". In: *Machine Learning for Autonomous Driving Workshop at the 33rd Conference on Neural Information ProcessingSystems (NeurIPS 2019)*.

[Ahmed 2019] Ahmed, Z., N. Le Roux, M. Norouzi, and D. Schuurmans (2019). "Understanding the impact of entropy on policy optimization". In: *Proceedings of the 36th International Conference on Machine Learning*. PMLR, pp. 151–160. URL: http://proceedings.mlr.press/v97/ahmed19a.html.

[Andrychowicz 2021] Andrychowicz, M., A. Raichuk, P. Stańczyk, M. Orsini, S. Girgin, R. Marinier, L. Hussenot, M. Geist, O. Pietquin, M. Michalski, S. Gelly, and O. Bachem (2021). "What Matters In On-Policy Reinforcement Learning? A Large-Scale Empirical Study". In: *9th International Conference on Learning Representations (ICLR 2021)*. arXiv: 2006.05990. URL: http://arxiv.org/abs/2006.05990.

[Andrychowicz 2017] Andrychowicz, M., F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, P. Abbeel, and W. Zaremba (2017). "Hindsight Experience Replay". In: *Advances in Neural Information Processing Systems (NeurIPS)*. 5048–5058. arXiv: 1707.01495. URL: http://arxiv.org/abs/1707.01495.

[Bansal 2018] Bansal, M., A. Krizhevsky, and A. Ogale (2018). "ChauffeurNet: Learning to Drive by Imitating the Best and Synthesizing the Worst". In: pp. 1–20. arXiv: 1812.03079. URL: http://arxiv.org/abs/1812.03079.

[Barth-Maron 2018] Barth-Maron, G., M. W. Hoffman, D. Budden, W. Dabney, D. Horgan, D. TB, A. Muldal, N. Heess, and T. Lillicrap (2018). "Distributed Distributional Deterministic Policy Gradients". In: *arXiv preprint*, pp. 1–16. arXiv: 1804.08617. URL: http://arxiv.org/abs/1804.08617.

[Barto 2003] Barto, A. G. and S. Mahadevan (2003). "Recent Advances in Hierarchical Reinforcement Learning". In: *Discrete Event Dynamic Systems* 13, pp. 41–77. ISSN: 09246703. DOI: 10.1007/s10626-008-0047-2.

[Beattie 2016] Beattie, C., J. Z. Leibo, D. Teplyashin, T. Ward, M. Wainwright, H. Küttler, A. Lefrancq, S. Green, V. Valdés, A. Sadik, J. Schrittwieser, K. Anderson, S. York, M. Cant, A. Cain, A. Bolton, S. Gaffney, H. King, D. Hassabis, S. Legg, and S. Petersen (2016). "DeepMind

Lab". In: *arXiv preprint*, pp. 1–11. arXiv: 1612.03801. URL: http://arxiv.org/abs/1612.03801.

[Behl 2020] Behl, A., K. Chitta, A. Prakash, E. Ohn-Bar, and A. Geiger (2020). "Label efficient visual abstractions for autonomous driving". In: *International Conference on Intelligent Robots and Systems (IROS)*. arXiv: 2005.10091.

[Bellemare 2017] Bellemare, M. G., W. Dabney, and R. Munos (2017). "A Distributional Perspective on Reinforcement Learning". In: arXiv: 1707.06887. URL: http://arxiv.org/abs/1707.06887.

[Bellemare 2013] Bellemare, M. G., Y. Naddaf, J. Veness, and M. Bowling (2013). "The arcade learning environment: An evaluation platform for general agents". In: *Journal of Artificial Intelligence Research* 47, pp. 253–279. ISSN: 10450823. DOI: 10.1613/jair.3912. arXiv: arXiv:1207.4708v1.

[Bewley 2019] Bewley, A., J. Rigley, Y. Liu, J. Hawke, R. Shen, V. D. Lam, and A. Kendall (2019). "Learning to drive from simulation without real world labels". In: *2019 International Conference on Robotics and Automation (ICRA)*, pp. 4818–4824. ISSN: 10504729. DOI: 10.1109/ICRA.2019.8793668. arXiv: 1812.03823.

[Błażej 2020] Błażej, O., P. Miłoś, A. Jakubowski, P. Zięcina, M. Martyniak, C. Galias, A. Breuer, S. Homoceanu, and H. Michalewski (2020). "CARLA Real Traffic Scenarios – novel training ground and benchmark for autonomous driving". In: *arXiv*. arXiv: 2012.11329. URL: http://arxiv.org/abs/2012.11329.

[Bojarski 2016] Bojarski, M., D. Del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, X. Zhang, J. Zhao, and K. Zieba (2016). "End to End Learning for Self-Driving Cars". In: *arXiv preprint*, pp. 1–9. arXiv: 1604.07316. URL: http://arxiv.org/abs/1604.07316.

[Brodeur 2017] Brodeur, S., E. Perez, A. Anand, F. Golemo, L. Celotti, F. Strub, J. Rouat, H. Larochelle, and A. Courville (2017). "HoME: a Household Multimodal Environment". In: *NIPS 2017's Visually-Grounded Interaction and Language Workshop*, pp. 1–6. arXiv: 1711.11017. URL: http://arxiv.org/abs/1711.11017.

[Brostow 2009] Brostow, G. J., J. Fauqueur, and R. Cipolla (2009). "Semantic object classes in video: A high-definition ground truth database". In: *Pattern Recognition Letters* 30.2, pp. 88–97. ISSN: 01678655. DOI: 10.1016/j.patrec.2008.04.005. URL: http://dx.doi.org/10.1016/j.patrec.2008.04.005.

[Cabon 2020] Cabon, Y., N. Murray, and M. Humenberger (2020). "Virtual Kitti 2". In: *arXiv preprint*, pp. 1–11. ISSN: 23318422. arXiv: 2001.10773.

[Caesar 2020] Caesar, H., V. Bankiti, A. H. Lang, S. Vora, V. E. Liong, Q. Xu, A. Krishnan, Y. Pan, G. Baldan, and O. Beijbom (2020). "nuScenes: A multimodal dataset for autonomous driving". In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 11621–11631.

[2019] *CARLA Autonomous Driving Challenge* (2019). https://carlachallenge.org/.

[Carton 2021a] Carton, F., D. Filliat, J. Rabarisoa, and Q. C. Pham (2021a). "Evaluating Robustness over High Level Driving Instruction for Autonomous Driving". In: *accepted to Intelligent Vehicles Symposium (IV21)*.

[Carton 2021b] Carton, F., D. Filliat, J. Rabarisoa, and Q. C. Pham (2021b). "Using Semantic Information to Improve Generalization of Reinforcement Learning Policies for Autonomous Driving". In: *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV) Workshops*, pp. 144–151. URL: https://openaccess.thecvf.com/content/WACV2021W/AVV/html/Carton_Using_Semantic_Information_to_Improve_Generalization_of_Reinforcement_Learning_Policies_WACVW_2021_paper.html.

[Chaplot 2018] Chaplot, D. S., K. M. Sathyendra, R. K. Pasumarthi, D. Rajagopal, and R. Salakhutdinov (2018). "Gated-attention architectures for task-oriented language grounding". In: *32nd AAAI Conference on Artificial Intelligence, AAAI 2018*, pp. 2819–2826. arXiv: 1706.07230.

[Chaurasia 2018] Chaurasia, A. and E. Culurciello (2018). "LinkNet: Exploiting encoder representations for efficient semantic segmentation". In: *2017 IEEE Visual Communications and Image Processing, VCIP 2017* 2018-Janua, pp. 1–4. DOI: 10.1109/VCIP.2017.8305148. arXiv: arXiv:1707.03718v1.

[C. Chen 2015] Chen, C., A. Seff, A. Kornhauser, and J. Xiao (2015). "DeepDriving : Learning Affordance for Direct Perception in Autonomous". In: *ICCV* 15.Figure 1, pp. 137–171. ISSN: 15734285. DOI: 10.1016/S1573-4285(07)00003-8.

[D. Chen 2020] Chen, D., Z. Brady, K. Vladlen, and K. Hilipp (2020). "Learning by Cheating". In: *Conference on Robot Learning (CoRL)*. PMLR, pp. 66–75. arXiv: 1912.12294. URL: http://proceedings.mlr.press/v100/chen20a.html.

[H. Chen 2017] Chen, H., S. Lundberg, and S. I. Lee (2017). "Checkpoint ensembles: Ensemble methods from a single training process". In: *arXiv*. ISSN: 23318422. arXiv: 1710.03282.

[J. Chen 2018] Chen, J., Z. Wang, and M. Tomizuka (2018). "Deep Hierarchical Reinforcement Learning for Autonomous Driving with Distinct Behaviors". In: *IEEE Intelligent Vehicles Symposium, Proceedings*. Vol. 2018-June. Iv. IEEE, pp. 1239–1244. ISBN: 9781538644522. DOI: 10.1109/IVS.2018.8500368.

[J. Chen 2019] Chen, J., B. Yuan, and M. Tomizuka (2019). "Model-free Deep Reinforcement Learning for Urban Autonomous Driving". In: *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*, pp. 2765–2771. ISBN: 9781538670248. DOI: 10.1109/ITSC.2019.8917306. arXiv: 1904.09503. URL: http://arxiv.org/abs/1904.09503.

[L. C. Chen 2018] Chen, L. C., G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille (2018). "DeepLab: Semantic Image Segmentation with Deep Convolutional Nets, Atrous Convolution, and Fully Connected CRFs". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 40.4, pp. 834–848. ISSN: 01628828. DOI: 10.1109/TPAMI.2017.2699184. arXiv: 1606.00915.

[Y. Chen 2019] Chen, Y., C. Dong, P. Palanisamy, P. Mudalige, K. Muelling, and J. M. Dolan (2019). "Attention-based Hierarchical Deep Reinforcement Learning for Lane Change Behaviors in Autonomous Driving". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pp. 137–145. URL: http://openaccess.thecvf.com/content_CVPRW_2019/papers/WAD/Chen_Attention-Based_Hierarchical_Deep_Reinforcement_Learning_for_Lane_Change_Behaviors_in_CVPRW_2019_paper.pdf.

[Cobbe 2019] Cobbe, K., O. Klimov, C. Hesse, T. Kim, and J. Schulman (2019). "Quantifying generalization in reinforcement learning". In: *36th International Conference on Machine Learning, ICML 2019* 2019-June, pp. 2280–2293. arXiv: 1812.02341.

[Codevilla 2018] Codevilla, F., A. M. López, V. Koltun, and A. Dosovitskiy (2018). "On offline evaluation of vision-based driving models". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 11219 LNCS, pp. 246–262. ISSN: 16113349. DOI: 10.1007/978-3-030-01267-0_15. arXiv: arXiv:1809.04843v1.

[Codevilla 2017] Codevilla, F., M. Müller, A. López, V. Koltun, and A. Dosovitskiy (2017). "End-to-end Driving via Conditional Imitation Learning". In: *Proceedings ICRA, 2018*. arXiv: 1710.02410.

[Codevilla 2019] Codevilla, F., E. Santana, A. M. López, and A. Gaidon (2019). "Exploring the Limitations of Behavior Cloning for Autonomous Driving". In: *Proceedings of the IEEE International Conference on Computer Vision*, pp. 9329–9338. arXiv: 1904.08980. URL: http://arxiv.org/abs/1904.08980.

[contributors 2019] contributors, T. garage (2019). *Garage: A toolkit for reproducible reinforcement learning research*. https://github.com/rlworkgroup/garage.

[Cordts 2015] Cordts, M., M. Enzweiler, R. Benenson, S. Ramos, T. Scharw, U. Franke, S. Roth, D. a. G. R, T. U. Darmstadt, M. P. I. Informatics, and T. U. Dresden (2015). "The Cityscapes Dataset". In: *CVPR Workshop on The Future of Datasets in Vision*. arXiv: 1604.01685.

[Cordts 2016] Cordts, M., M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele (2016). "The Cityscapes Dataset for Semantic Urban Scene Understanding". In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition* 2016-Decem, pp. 3213–3223. ISSN: 10636919. DOI: 10.1109/CVPR.2016.350. arXiv: arXiv:1604.01685v2.

[Cultrera 2020] Cultrera, L., L. Seidenari, F. Becattini, P. Pala, and A. Del Bimbo (2020). "Explaining autonomous driving by learning end-to-end visual attention". In: *IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops* 2020-June, pp. 1389–1398. ISSN: 21607516. DOI: 10.1109/CVPRW50498.2020.00178. arXiv: 2006.03347.

[Deng 2009] Deng, J., W. Dong, R. Socher, L.-J. Li, Kai Li, and Li Fei-Fei (June 2009). "ImageNet: A large-scale hierarchical image database". In: *2009 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, pp. 248–255. ISBN: 978-1-4244-3992-8. DOI: 10.1109/CVPR.2009.5206848. URL: https://ieeexplore.ieee.org/document/5206848/.

[Dhariwal 2017] Dhariwal, P., C. Hesse, O. Klimov, A. Nichol, M. Plappert, A. Radford, J. Schulman, S. Sidor, Y. Wu, and P. Zhokhov (2017). *OpenAI Baselines*. https://github.com/openai/baselines.

[Dietterich 2000] Dietterich, T. G. (2000). "Ensemble Methods in Machine Learning". In: *MCS: International Workshop on Multiple Classifier Systems*. Springer Berlin Heidelberg, pp. 1–15.

[Dosovitskiy and Koltun 2017] Dosovitskiy, A. and V. Koltun (2017). "Learning to Act by Predicting the Future". In: *ICLR 2017*, pp. 1–14. ISSN: 00234028.

[Dosovitskiy, Ros, 2017] Dosovitskiy, A., G. Ros, F. Codevilla, A. Lopez, and V. Koltun (2017). "CARLA: An Open Urban Driving Simulator". In: *1st Conference on Robot Learning (CoRL 2017)*, pp. 1–16. arXiv: 1711.03938. URL: http://arxiv.org/abs/1711.03938.

[Duan 2016] Duan, Y., X. Chen, R. Houthooft, J. Schulman, and P. Abbeel (2016). "Benchmarking Deep Reinforcement Learning for Continuous Control". In: 48. arXiv: 1604.06778. URL: http://arxiv.org/abs/1604.06778.

[Engstrom 2019] Engstrom, L., A. Ilyas, S. Santurkar, D. Tsipras, F. Janoos, L. Rudolph, and A. Madry (2019). "Implementation Matters in Deep Policy Gradients: a Case Study on Ppo and Trpo". In: *7th International Conference on Learning Representations (ICLR 2019)*, pp. 1–14.

[Espeholt 2018] Espeholt, L., H. Soyer, R. Munos, K. Simonyan, V. Mnih, T. Ward, B. Yotam, F. Vlad, H. Tim, I. Dunning, S. Legg, and K. Kavukcuoglu (2018). "IMPALA: Scalable Distributed Deep-RL with Importance Weighted Actor-Learner Architectures". In: *35th International Conference on Machine Learning, ICML 2018* 4, pp. 2263–2284. arXiv: 1802.01561.

[Fernando 2018] Fernando, T., S. Denman, S. Sridharan, and C. Fookes (2018). "Going deeper: Autonomous steering with neural memory networks". In: *Proceedings - 2017 IEEE International Conference on Computer Vision Workshops, ICCVW 2017* 2018-Janua, pp. 214–221. DOI: 10.1109/ICCVW.2017.34.

[Finn, Abbeel, 2017] Finn, C., P. Abbeel, and S. Levine (2017). "Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks". In: *arXiv preprint*. arXiv: 1703.03400. URL: http://arxiv.org/abs/1703.03400.

[Finn and Levine 2017] Finn, C. and S. Levine (2017). "Meta-Learning and Universality: Deep Representations and Gradient Descent can Approximate any Learning Algorithm". In: pp. 1–18. arXiv: 1710.11622. URL: http://arxiv.org/abs/1710.11622.

[Finn, Yu, 2017] Finn, C., T. Yu, T. Zhang, P. Abbeel, and S. Levine (2017). "One-Shot Visual Imitation Learning via Meta-Learning". In: *arXiv preprint*. arXiv: 1709.04905. URL: http://arxiv.org/abs/1709.04905.

[Fritsch 2013] Fritsch, J., T. Kuhnl, and A. Geiger (2013). "A new performance measure and evaluation benchmark for road detection algorithms". In: *IEEE Conference on Intelligent Transportation Systems, Proceedings, ITSC*, pp. 1693–1700. DOI: 10.1109/ITSC.2013.6728473.

[Gaidon 2016] Gaidon, A., Q. Wang, Y. Cabon, and E. Vig (2016). "VirtualWorlds as Proxy for Multi-object Tracking Analysis". In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition* 2016-Decem, pp. 4340–4349. ISSN: 10636919. DOI: 10.1109/CVPR.2016.470.

[Gandhi 2017] Gandhi, D., L. Pinto, and A. Gupta (2017). "Learning to fly by crashing". In: *IEEE International Conference on Intelligent Robots and Systems* 2017-Septe, pp. 3948–3955. ISSN: 21530866. DOI: 10.1109/IROS.2017.8206247. arXiv: arXiv:1704.05588v2.

[Gao 2018] Gao, Y., Huazhe, Xu, J. Lin, F. Yu, S. Levine, and T. Darrell (2018). "Reinforcement Learning from Imperfect Demonstrations". In: arXiv: 1802.05313. URL: http://arxiv.org/abs/1802.05313.

[Geiger 2013] Geiger, A., P. Lenz, C. Stiller, and R. Urtasun (2013). "Vision meets robotics: The KITTI dataset". In: *International Journal of Robotics Research* 32.11, pp. 1231–1237. ISSN: 02783649. DOI: 10.1177/0278364913491297.

[Geiger 2012] Geiger, A., P. Lenz, and R. Urtasun (2012). "Are we ready for autonomous driving? the KITTI vision benchmark suite". In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 3354–3361. ISSN: 10636919. DOI: 10.1109/CVPR.2012.6248074.

[Geyer 2020] Geyer, J., Y. Kassahun, M. Mahmudi, X. Ricou, R. Durgesh, A. S. Chung, L. Hauswald, V. H. Pham, M. Mühlegg, S. Dorn, T. Fernandez, M. Jänicke, S. Mirashi, C. Savani, M. Sturm, O. Vorobiov, M. Oelker, S. Garreis, and P. Schuberth (2020). "A2d2: Audi autonomous driving dataset". In: *arXiv*. ISSN: 23318422. arXiv: 2004.06320.

[Goodfellow 2016] Goodfellow, I., Y. Bengio, and A. Courville (2016). *Deep Learning*. http://www.deeplearningbook.org. MIT Press.

[Gordon 2012] Gordon, J. (2012). *Javascript Racer*. https://codeincomplete.com/posts/javascript-racer/.

[Gu 2016] Gu, S., T. Lillicrap, Z. Ghahramani, R. E. Turner, and S. Levine (2016). "Q-Prop: Sample-Efficient Policy Gradient with An Off-Policy Critic". In: pp. 1–13. arXiv: 1611.02247. URL: http://arxiv.org/abs/1611.02247.

[Guadarrama 2018] Guadarrama, S., A. Korattikara, O. Ramirez, P. Castro, E. Holly, S. Fishman, K. Wang, E. Gonina, N. Wu, E. Kokiopoulou, L. Sbaiz, J. Smith, G. Bartók, J. Berent, C. Harris, V. Vanhoucke, and E. Brevdo (2018). *TF-Agents: A library for Reinforcement Learning in TensorFlow*. https://github.com/tensorflow/agents. [Online; accessed 25-June-2019]. URL: https://github.com/tensorflow/agents.

[Gulcehre 2020] Gulcehre, C., T. Le Paine, B. Shahriari, M. Denil, M. Hoffman, H. Soyer, R. Tanburn, S. Kapturowski, N. Rabinowitz, D. Williams, G. Barth-Maron, Z. Wang, N. de Freitas, and W. Team (2020). "Making efficient use of demonstrations to solve hard exploration problems". In: *8th International Conference on Learning Representations (ICLR 2020)*, pp. 1–22. arXiv: 1909.01387.

[Haarnoja 2017] Haarnoja, T., H. Tang, P. Abbeel, and S. Levine (2017). "Reinforcement Learning with Deep Energy-Based Policies". In: arXiv: 1702.08165. URL: http://arxiv.org/abs/1702.08165.

[Haarnoja, Zhou, Abbeel, 2018] Haarnoja, T., A. Zhou, P. Abbeel, and S. Levine (2018). "Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor". In: *Proceedings of the 35th International Conference on Machine Learning*, pp. 1861–1870. arXiv: 1801.01290. URL: http://arxiv.org/abs/1801.01290.

[Haarnoja, Zhou, Hartikainen, 2018] Haarnoja, T., A. Zhou, K. Hartikainen, G. Tucker, S. Ha, J. Tan, V. Kumar, H. Zhu, A. Gupta, P. Abbeel, and S. Levine (2018). "Soft Actor-Critic Algorithms and Applications". In: arXiv: 1812.05905. URL: http://arxiv.org/abs/1812.05905.

[Hasselt 2010] Hasselt, H. van (2010). "Double Q-Learning". In: *Advances in Neural Information Processing Systems*, pp. 2613–2621.

[Hasselt 2015] Hasselt, H. van, A. Guez, and D. Silver (2015). "Deep Reinforcement Learning with Double Q-learning". In: arXiv: 1509.06461. URL: http://arxiv.org/abs/1509.06461.

[Hawke 2019] Hawke, J., R. Shen, C. Gurau, S. Sharma, D. Reda, N. Nikolov, P. Mazur, S. Micklethwaite, N. Griffiths, A. Shah, and A. Kendall (2019). "Urban Driving with Conditional Imitation Learning". In: arXiv: 1912.00177. URL: http://arxiv.org/abs/1912.00177.

[He 2016] He, K., X. Zhang, S. Ren, and J. Sun (2016). "Deep residual learning for image recognition". In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition* 2016-Decem, pp. 770–778. ISSN: 10636919. DOI: 10.1109/CVPR.2016.90. arXiv: arXiv:1512.03385v1.

[Hecker 2019] Hecker, S., D. Dai, and L. Van Gool (2019). "Learning Accurate, Comfortable and Human-like Driving". In: *arXiv preprint*. arXiv: 1903.10995. URL: http://arxiv.org/abs/1903.10995.

[Heess 2015] Heess, N., J. J. Hunt, T. P. Lillicrap, and D. Silver (2015). "Memory-based control with recurrent neural networks". In: pp. 1–11. arXiv: 1512.04455. URL: http://arxiv.org/abs/1512.04455.

[Henderson 2018] Henderson, P., R. Islam, P. Bachman, J. Pineau, D. Precup, and D. Meger (2018). "Deep Reinforcement Learning that Matters". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. arXiv: 1709.06560. URL: http://arxiv.org/abs/1709.06560.

[Hessel 2018] Hessel, M., J. Modayil, H. van Hasselt, T. Schaul, G. Ostrovski, W. Dabney, D. Horgan, B. Piot, M. Azar, and D. Silver (2018). "Rainbow: Combining Improvements in Deep Reinforcement Learning". In: arXiv: 1710.02298. URL: http://arxiv.org/abs/1710.02298.

[Hessel 2019] Hessel, M., H. Soyer, L. Espeholt, W. Czarnecki, S. Schmitt, and H. Van Hasselt (2019). "Multi-Task Deep Reinforcement Learning with PopArt". In: *Proceedings of the AAAI Conference on Artificial Intelligence* 33, pp. 3796–3803. ISSN: 2159-5399. DOI: 10.1609/aaai.v33i01.33013796. arXiv: arXiv:1809.04474v1.

[Hester 2017] Hester, T., M. Vecerik, O. Pietquin, M. Lanctot, T. Schaul, B. Piot, D. Horgan, J. Quan, A. Sendonaris, G. Dulac-Arnold, I. Osband, J. Agapiou, J. Z. Leibo, and A. Gruslys (2017). "Deep Q-learning from Demonstrations". In: *Proceedings of the AAAI Conference on Artificial Intelligence* 32. arXiv: 1704.03732. URL: http://arxiv.org/abs/1704.03732.

[Hill 2018] Hill, A., A. Raffin, M. Ernestus, A. Gleave, A. Kanervisto, R. Traore, P. Dhariwal, C. Hesse, O. Klimov, A. Nichol, M. Plappert, A. Radford, J. Schulman, S. Sidor, and Y. Wu (2018). *Stable Baselines*. https://github.com/hill-a/stable-baselines.

[Hinton 2015] Hinton, G., O. Vinyals, and J. Dean (2015). "Distilling the Knowledge in a Neural Network". In: *NIPS Deep Learning and Representation Learning Workshop*, pp. 1–9. arXiv: 1503.02531. URL: http://arxiv.org/abs/1503.02531.

[Huang 2020] Huang, X., P. Wang, X. Cheng, D. Zhou, Q. Geng, and R. Yang (2020). "The ApolloScape Open Dataset for Autonomous Driving and Its Application". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 42.10, pp. 2702–2719. ISSN: 19393539. DOI: 10.1109/TPAMI.2019.2926463. arXiv: 1803.06184.

[Hubschneider 2018] Hubschneider, C., A. Bauer, M. Weber, and J. M. Zollner (2018). "Adding navigation to the equation: Turning decisions for end-to-end vehicle control". In: *IEEE Conference on Intelligent Transportation Systems, Proceedings, ITSC* 2018-March, pp. 1–8. DOI: 10.1109/ITSC.2017.8317923.

[Islam 2017] Islam, R., P. Henderson, M. Gomrokchi, and D. Precup (2017). "Reproducibility of Benchmarked Deep Reinforcement Learning Tasks for Continuous Control". In: arXiv: 1708.04133. URL: http://arxiv.org/abs/1708.04133.

[Izadinia 2016] Izadinia, H. and S. M. Seitz (2016). "IM2CAD". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 5134–5143. URL: https://openaccess.thecvf.com/content_cvpr_2017/papers/Izadinia_IM2CAD_CVPR_2017_paper.pdf.

[Jaderberg 2016] Jaderberg, M., V. Mnih, W. M. Czarnecki, T. Schaul, J. Z. Leibo, D. Silver, and K. Kavukcuoglu (2016). "Reinforcement Learning with Unsupervised Auxiliary Tasks". In: pp. 1–14. arXiv: 1611.05397. URL: http://arxiv.org/abs/1611.05397.

[Jaritz 2018] Jaritz, M., R. De Charette, M. Toromanoff, E. Perot, and F. Nashashibi (2018). "End-to-End Race Driving with Deep Reinforcement Learning". In: *Proceedings - IEEE International Conference on Robotics and Automation*, pp. 2070–2075. ISSN: 10504729. DOI: 10.1109/ICRA.2018.8460934.

[Joseph 2016] Joseph, S. (Aug. 2016). "Australian Literary Journalism and "Missing Voices"". In: *Journalism Practice* 10.6, pp. 730–743. URL: http://proceedings.mlr.press/v37/ioffe15.pdf%20http://www.tandfonline.com/doi/full/10.1080/17512786.2015.1058180.

[Jung 2018] Jung, A. (2018). *imgaug*. https://imgaug.readthedocs.io/en/latest//.

[Kempka 2017] Kempka, M., M. Wydmuch, G. Runc, J. Toczek, and W. Jaskowski (2017). "ViZDoom: A Doom-based AI research platform for visual reinforcement learning". In: *IEEE Conference on Computatonal Intelligence and Games, CIG*, pp. 3–4. ISSN: 23254289. DOI: 10.1109/CIG.2016.7860433. arXiv: arXiv:1605.02097.

[Kendall 2017] Kendall, A. and Y. Gal (2017). "What uncertainties do we need in Bayesian deep learning for computer vision?" In: *Advances in Neural Information Processing Systems* 2017-Decem.Nips, pp. 5575–5585. ISSN: 10495258. arXiv: arXiv:1703.04977v2.

[Kendall 2018] Kendall, A., J. Hawke, D. Janz, P. Mazur, D. Reda, J.-M. Allen, V.-D. Lam, A. Bewley, and A. Shah (2018). "Learning to Drive in a Day". In: arXiv: 1807.00412. URL: http://arxiv.org/abs/1807.00412.

[Kingma 2015] Kingma, D. P. and J. L. Ba (2015). "Adam: A method for stochastic optimization". In: *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*, pp. 1–15. arXiv: 1412.6980.

[Kolve 2017a] Kolve, E., R. Mottaghi, W. Han, E. VanderBilt, L. Weihs, A. Herrasti, D. Gordon, Y. Zhu, A. Gupta, and A. Farhadi (2017a). "AI2-THOR: An Interactive 3D Environment for Visual AI". In: pp. 3–6. arXiv: 1712.05474. URL: http://arxiv.org/abs/1712.05474.

[Kolve 2017b] Kolve, E., R. Mottaghi, W. Han, E. VanderBilt, L. Weihs, A. Herrasti, D. Gordon, Y. Zhu, A. Gupta, and A. Farhadi (2017b). "AI2-THOR: An interactive 3D environment for visual AI". In: *arXiv*, pp. 2–5. ISSN: 23318422. arXiv: 1712.05474.

[Krogh 1992] Krogh, A. and J. Hertz (1992). "A Simple Weight Decay Can Improve Generalization". In: *Advances in Neural Information Processing Systems*. Vol. 4, pp. 950–957. URL: https://proceedings.neurips.cc/paper/1991/file/8eefcfdf5990e441f0fb6f3fad709e21-Paper.pdf.

[Kulhánek 2019] Kulhánek, J., E. Derner, T. de Bruin, and R. Babuška (2019). "Vision-based Navigation Using Deep Reinforcement Learning". In: *2019 European Conference on Mobile Robots (ECMR)*, pp. 2–9. ISSN: 23318422. DOI: 10.1109/ECMR.2019.8870964. arXiv: 1908.03627.

[Lake 2013] Lake, B. M. (2013). "Supplementary material: One-shot learning by inverting a compositional causal process". In: pp. 1–6.

[Lakshminarayanan 2016] Lakshminarayanan, A. S., S. Ozair, and Y. Bengio (2016). "Reinforcement Learning with Few Expert Demonstrations". In: *Neural Information Processing Systems - Workshop on Deep Learning for Action and Interaction*.

[Lample 2017] Lample, G. and D. S. Chaplot (2017). "Playing FPS Games with Deep Reinforcement Learning". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. arXiv: 1609.05521. URL: http://arxiv.org/abs/1609.05521.

[Laskin 2020] Laskin, M., K. Lee, A. Stooke, L. Pinto, P. Abbeel, and A. Srinivas (2020). "Reinforcement Learning with Augmented Data". In: *arXiv preprint*. arXiv: 2004.14990. URL: http://arxiv.org/abs/2004.14990.

[Lecun 1998] Lecun, Y., L. Bottou, Y. Bengio, and P. Haffner (1998). "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11, pp. 2278–2324. DOI: 10.1109/5.726791. URL: http://ieeexplore.ieee.org/document/726791/%7B%5C#%7Dfull-text-section.

[LeCun 2005] LeCun, Y., U. Muller, J. Ben, E. Cosatto, and B. Flepp (2005). "Off-road obstacle avoidance through end-to-end learning". In: *Advances in Neural Information Processing Systems*, pp. 739–746. ISSN: 10495258.

[Lee 2020] Lee, K., K. Lee, J. Shin, and H. Lee (2020). "Network randomization: A simple technique for generalization in deep reinforcement learning". In: *8th International Conference on Learning Representations (ICLR 2020)*, pp. 1–22. ISSN: 23318422. arXiv: 1910.05396.

[Levine 2017] Levine, S. (2017). *CS294 - Deep Reinforcement Learning lecture - UC Berkeley*. http://rll.berkeley.edu/deeprlcourse/.

[D. Li 2019] Li, D., D. Zhao, Q. Zhang, and Y. Chen (2019). "Reinforcement Learning and Deep Learning Based Lateral Control for Autonomous Driving [Application Notes]". In: *IEEE Computational Intelligence Magazine* 14.2, pp. 83–98. ISSN: 15566048. DOI: 10.1109/MCI.2019.2901089. arXiv: arXiv:1810.12778v1.

[Z. Li 2018] Li, Z., T. Motoyoshi, K. Sasaki, T. Ogata, and S. Sugano (2018). "Rethinking Self-Driving: Multi-Task Knowledge for Better Generalization and Accident Explanation Ability". In: *arXiv*, pp. 1–11. ISSN: 23318422. arXiv: 1809.11100.

[Liang 2018] Liang, X., T. Wang, L. Yang, and E. Xing (2018). "CIRL: Controllable Imitative Reinforcement Learning for Vision-based Self-driving". In: *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 584–599. URL: http://arxiv.org/abs/1807.03776.

[Lillicrap 2016] Lillicrap, T. P., J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra (2016). "Continuous control with deep reinforcement learning". In: *4th International Conference on Learning Representations, ICLR 2016*. arXiv: 1509.02971. URL: http://arxiv.org/abs/1509.02971.

[Lin 1993] Lin, L.-J. (1993). "Reinforcement learning for robots using neural networks". In: *ProQuest Dissertations and Theses*, p. 160. URL: https://search.proquest.com/docview/303995826?accountid=12063%7B%5C%%7D0Ahttp://fg2fy8yh7d.search.serialssolutions.com/directLink?%7B%5C&%7Datitle=Reinforcement+learning+for+robots+using+neural+networks%7B%5C&%7Dauthor=Lin%7B%5C%%7D2C+Long-Ji%7B%5C&%7Dissn=%7B%5C&%7Dtitle=Reinforcement+learning+for+robots+us.

[Liu 2017] Liu, M. Y., T. Breuel, and J. Kautz (2017). "Unsupervised image-to-image translation networks". In: *31st Conference on Neural Information Processing Systems (NIPS 2017*, pp. 701–709. ISSN: 10495258. arXiv: 1703.00848.

[Ma 2019] Ma, X., K. Driggs-Campbell, and M. J. Kochenderfer (2019). "Improved Robustness and Safety for Autonomous Vehicle Control with Adversarial Reinforcement Learning". In: *arXiv* Iv, pp. 1665–1671.

[Maddern 2017] Maddern, W., G. Pascoe, C. Linegar, and P. Newman (2017). "1 year, 1000 km: The Oxford RobotCar dataset". In: *The International Journal of Robotics Research* 36, pp. 3–15.

[Mahjourian 2017] Mahjourian, R., M. Wicke, and A. Angelova (2017). "Geometry-based next frame prediction from monocular video". In: *IEEE Intelligent Vehicles Symposium, Proceedings*, pp. 1700–1707. DOI: 10.1109/IVS.2017.7995953. arXiv: arXiv:1609.06377v2.

[McCormac 2016] McCormac, J., A. Handa, S. Leutenegger, and A. J. Davison (2016). "SceneNet RGB-D: 5M Photorealistic Images of Synthetic Indoor Trajectories with Ground Truth". In: arXiv: 1612.05079. URL: http://arxiv.org/abs/1612.05079.

[McCulloch 1943] McCulloch, W. S. and W. Pitts (Dec. 1943). "A logical calculus of the ideas immanent in nervous activity". In: *The Bulletin of Mathematical Biophysics* 5.4, pp. 115–133. ISSN: 0007-4985. DOI: 10.1007/BF02478259. URL: http://link.springer.com/10.1007/BF02478259.

[Mehta 2018] Mehta, A., A. Subramanian, and A. Subramanian (2018). "Learning End-to-end Autonomous Driving using Guided Auxiliary Supervision". In: arXiv: 1808.10393. URL: http://arxiv.org/abs/1808.10393.

[Menze 2015] Menze, M. and A. Geiger (2015). "Object scene flow for autonomous vehicles". In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 3061–3070. ISSN: 10636919. DOI: 10.1109/CVPR.2015.7298925.

[Michaelis 2019] Michaelis, C., B. Mitzkus, R. Geirhos, E. Rusak, O. Bringmanny, A. S. Eckery, M. Bethgey, and W. Brendely (2019). "Benchmarking Robustness in Object Detection: Autonomous driving when winter is coming". In: *arXiv preprint*. arXiv: 1907.07484.

[Michels 2005] Michels, J., A. Saxena, and A. Y. Ng (2005). "High speed obstacle avoidance using monocular vision and reinforcement learning". In: *ICML 2005 - Proceedings of the 22nd International Conference on Machine Learning*, pp. 593–600. DOI: 10.1145/1102351.1102426.

[Mirchevska 2018] Mirchevska, B., C. Pek, M. Werling, M. Althoff, and J. Boedecker (2018). "High-level Decision Making for Safe and Reasonable Autonomous Lane Changing using Reinforcement Learning". In: *IEEE Conference on Intelligent Transportation Systems, Proceedings, ITSC* 2018-Novem, pp. 2156–2162. DOI: 10.1109/ITSC.2018.8569448.

[Mirowski 2018] Mirowski, P., M. K. Grimes, M. Malinowski, K. M. Hermann, K. Anderson, D. Teplyashin, K. Simonyan, K. Kavukcuoglu, A. Zisserman, and R. Hadsell (2018). "Learning to Navigate in Cities Without a Map". In: *Advances in Neural Information Processing Systems* 31, pp. 2419–2430. arXiv: 1804.00168. URL: http://arxiv.org/abs/1804.00168.

[Mirowski 2016] Mirowski, P., R. Pascanu, F. Viola, H. Soyer, A. J. Ballard, A. Banino, M. Denil, R. Goroshin, L. Sifre, K. Kavukcuoglu, D. Kumaran, and R. Hadsell (2016). "Learning to Navigate in Complex Environments". In: *arXiv preprint*. arXiv: 1611.03673. URL: http://arxiv.org/abs/1611.03673.

[Mishra 2018] Mishra, N., M. Rohaninejad, X. Chen, and P. Abbeel (2018). "A Simple Neural Attentive Meta-Learner". In: *arXiv preprint*. arXiv: 1707.03141. URL: http://arxiv.org/abs/1707.03141.

[Mnih 2016] Mnih, V., A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu (2016). "Asynchronous Methods for Deep Reinforcement Learning". In: *International conference on machine learning*, pp. 1928–1937. arXiv: 1602.01783. URL: http://arxiv.org/abs/1602.01783.

[Mnih 2015] Mnih, V., K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis (2015). "Human-level control through deep reinforcement learning." In: *Nature* 518.7540, pp. 529–33. ISSN: 1476-4687. DOI: 10.1038/nature14236. URL: http://www.ncbi.nlm.nih.gov/pubmed/25719670.

[Mnih 2013] Mnih, V., D. Silver, and M. Riedmiller (2013). "Playing Atari with Deep Reinforcement Learning". In: *Nips*, pp. 1–9. ISSN: 0028-0836. DOI: 10.1038/nature14236. arXiv: 1312.5602.

[Müller 2018] Müller, M., A. Dosovitskiy, B. Ghanem, and V. Koltun (2018). "Driving Policy Transfer via Modularity and Abstraction". In: arXiv: 1804.09364. URL: http://arxiv.org/abs/1804.09364.

[Neuhold 2017] Neuhold, G., T. Ollmann, S. R. Bulo, and P. Kontschieder (2017). "The Mapillary Vistas Dataset for Semantic Understanding of Street Scenes". In: *Proceedings of the IEEE International Conference on Computer Vision* 2017-Octob, pp. 5000–5009. ISSN: 15505499. DOI: 10.1109/ICCV.2017.534.

[Nielsen 2015] Nielsen, M. A. (2015). *Neural Networks and Deep Learning*. Determination Press.

[Oh 2015] Oh, J., X. Guo, H. Lee, R. Lewis, and S. Singh (2015). "Action-Conditional Video Prediction using Deep Networks in Atari Games". In: pp. 1–26. arXiv: 1507.08750. URL: http://arxiv.org/abs/1507.08750.

[Ohn-bar 2020] Ohn-bar, E., A. Prakash, A. Behl, K. Chitta, and A. Geiger (2020). "Learning Situational Driving". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.

[Osband 2016] Osband, I., C. Blundell, A. Pritzel, and B. V. Roy (2016). "Deep Exploration via Bootstrapped DQN". In: *30th Conference on Neural Information Processing Systems (NIPS 2016)*, pp. 1–18. arXiv: arXiv:1602.04621v3.

[X. Pan 2017] Pan, X., Y. You, Z. Wang, and C. Lu (2017). "Virtual to Real Reinforcement Learning for Autonomous Driving". In: *British Machine Vision Conference (BMVC)*. arXiv: 1704.03952. URL: http://arxiv.org/abs/1704.03952.

[Y. Pan 2017] Pan, Y., C. A. Cheng, K. Saigol, K. Lee, X. Yan, E. A. Theodorou, and B. Boots (2017). "Agile Autonomous Driving using End-to-End Deep Imitation Learning". In: *arXiv*. DOI: 10.15607/rss.2018.xiv.056. arXiv: 1709.07174.

[Pardo 2018] Pardo, F., A. Tavakoli, V. Levdik, and P. Kormushev (2018). "Time Limits in Reinforcement Learning". In: *International Conference on Machine Learning*, pp. 4045–4054. arXiv: 1712.00378. URL: http://arxiv.org/abs/1712.00378.

[Parisotto 2016] Parisotto, E., J. L. Ba, and R. Salakhutdinov (2016). "Actor-Mimic: Deep Multitask and Transfer Reinforcement Learning". In: *4th International Conference on Learning Representations (ICLR 2016)*, pp. 1–16. arXiv: 1511.06342. URL: http://arxiv.org/abs/1511.06342.

[Pathak 2017] Pathak, D., P. Agrawal, A. A. Efros, and T. Darrell (2017). "Curiosity-driven Exploration by Self-supervised Prediction". In: *Proceedings of the 34th International Conference on Machine Learning*, pp. 2778–2787. URL: http://proceedings.mlr.press/v70/pathak17a.html.

[Peng, Abbeel, 2018] Peng, X. B., P. Abbeel, S. Levine, and M. van de Panne (2018). "DeepMimic: Example-Guided Deep Reinforcement Learning of Physics-Based Character Skills". In: 37.4. DOI: 10.1145/3197517.3201311. arXiv: 1804.02717. URL: http://arxiv.org/abs/1804.02717%7B%5C%%7D0Ahttp://dx.doi.org/10.1145/3197517.3201311.

[Peng, Andrychowicz, 2018] Peng, X. B., M. Andrychowicz, W. Zaremba, and P. Abbeel (2018). "Sim-to-Real Transfer of Robotic Control with Dynamics Randomization". In: *Proceedings - IEEE International Conference on Robotics and Automation*, pp. 3803–3810. ISSN: 10504729. DOI: 10.1109/ICRA.2018.8460528. arXiv: arXiv:1710.06537v1.

[Perot 2017] Perot, E., M. Jaritz, M. Toromanoff, and R. D. Charette (2017). "End-to-End Driving in a Realistic Racing Game with Deep Reinforcement Learning". In: *IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops* 2017-July, pp. 474–475. ISSN: 21607516. DOI: 10.1109/CVPRW.2017.64. arXiv: arXiv:1605.02097.

[Pfeiffer 2018] Pfeiffer, M., S. Shukla, M. Turchetta, C. Cadena, A. Krause, R. Siegwart, and J. Nieto (2018). "Reinforced Imitation: Sample Efficient Deep Reinforcement Learning for Mapless Navigation by Leveraging Prior Demonstrations". In: *IEEE Robotics and Automation Letters* 3.4, pp. 4423–4430. ISSN: 23773766. DOI: 10.1109/LRA.2018.2869644. arXiv: arXiv:1805.07095v1.

[Pomerleau 1989] Pomerleau, D. A. (1989). "ALVINN: an autonomous land vehicle in a neural network". In: pp. 305–313. URL: http://dl.acm.org/citation.cfm?id=89851.89891.

[Prakash 2020] Prakash, A., A. Behl, E. Ohn-Bar, K. Chitta, and A. Geiger (2020). "Exploring data aggregation in policy learning for vision-based Urban autonomous driving". In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 11760–11770. ISSN: 10636919. DOI: 10.1109/CVPR42600.2020.01178.

[Qian 1999] Qian, N. (1999). "On the momentum term in gradient descent learning algorithms". In: *Neural Networks* 12.1, pp. 145–151. ISSN: 08936080. DOI: 10.1016/S0893-6080(98)00116-6.

[Al-Qizwini 2017] Al-Qizwini, M., I. Barjasteh, H. Al-Qassab, and H. Radha (2017). "Deep learning algorithm for autonomous driving using GoogLeNet". In: *IEEE Intelligent Vehicles Symposium (IV)*, pp. 89–96.

[Raileanu 2021] Raileanu, R. and R. Fergus (2021). "Decoupling Value and Policy for Generalization in Reinforcement Learning". In: arXiv: 2102.10330. URL: http://arxiv.org/abs/2102.10330.

[Rajeswaran 2016] Rajeswaran, A., S. Ghotra, B. Ravindran, and S. Levine (2016). "EPOpt: Learning Robust Neural Network Policies Using Model Ensembles". In: 2, pp. 1–15. arXiv: 1610.01283. URL: http://arxiv.org/abs/1610.01283.

[Rajeswaran 2017] Rajeswaran, A., V. Kumar, A. Gupta, G. Vezzani, J. Schulman, E. Todorov, and S. Levine (2017). "Learning Complex Dexterous Manipulation with Deep Reinforcement Learning and Demonstrations". In: arXiv: 1709.10087. URL: http://arxiv.org/abs/1709.10087.

[Rausch 2017] Rausch, V., A. Hansen, E. Solowjow, C. Liu, E. Kreuzer, and J. K. Hedrick (2017). "Learning a deep neural net policy for end-to-end control of autonomous vehicles". In: *Proceedings of the American Control Conference*, pp. 4914–4919. ISSN: 07431619. DOI: 10.23919/ACC.2017.7963716.

[Rhinehart 2020] Rhinehart, N., R. McAllister, and S. Levine (2020). "Deep Imitative Models for Flexible Inference, Planning, and Control". In: *International Conference on Learning Representations* 1, pp. 1–20. arXiv: 1810.06544. URL: http://arxiv.org/abs/1810.06544.

[Richter 2017] Richter, S. R., Z. Hayder, and V. Koltun (2017). "Playing for Benchmarks". In: *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pp. 2213–2222.

[Riedmiller 2007] Riedmiller, M., M. Montemerlo, and H. Dahlkamp (2007). "Learning to drive a real car in 20 minutes". In: *Proceedings of the Frontiers in the Convergence of Bioscience and Information Technologies, FBIT 2007*, pp. 645–650. DOI: 10.1109/FBIT.2007.37.

[Romera 2018] Romera, E., J. M. Alvarez, L. M. Bergasa, and R. Arroyo (2018). "ERFNet: Efficient Residual Factorized ConvNet for Real-Time Semantic Segmentation". In: *IEEE Transactions on Intelligent Transportation Systems* 19.1, pp. 263–272. ISSN: 15249050. DOI: 10.1109/TITS.2017.2750080.

[Ros 2016] Ros, G., L. Sellart, J. Materzynska, D. Vazquez, and A. M. Lopez (2016). "The SYNTHIA Dataset: A Large Collection of Synthetic Images for Semantic Segmentation of Urban Scenes". In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition* 2016-Decem.600388, pp. 3234–3243. ISSN: 10636919. DOI: 10.1109/CVPR.2016.352.

[Ross 2011] Ross, S., G. J. Gordon, and J. A. Bagnell (2011). "A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning". In: *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*. Vol. 15. JMLR Workshop and Conference Proceedings, pp. 627–635. URL: http://proceedings.mlr.press/v15/ross11a/ross11a.pdf.

[Rusu 2015] Rusu, A. A., S. G. Colmenarejo, C. Gulcehre, G. Desjardins, J. Kirkpatrick, R. Pascanu, V. Mnih, K. Kavukcuoglu, and R. Hadsell (2015). "Policy Distillation". In: pp. 1–13. arXiv: 1511.06295. URL: http://arxiv.org/abs/1511.06295.

[Rusu 2016] Rusu, A. A., N. C. Rabinowitz, G. Desjardins, H. Soyer, J. Kirkpatrick, K. Kavukcuoglu, R. Pascanu, and R. Hadsell (2016). "Progressive Neural Networks". In: arXiv: 1606.04671. URL: http://arxiv.org/abs/1606.04671.

[Rusu 2017] Rusu, A. A., M. Vecerik, T. Rothörl, N. Heess, R. Pascanu, and R. Hadsell (2017). "Sim-to-Real Robot Learning from Pixels with Progressive Nets". In: *Proceedings of the 1st Annual Conference on Robot Learning (CoRL 2017)*, pp. 1–9. arXiv: 1610.04286. URL: http://arxiv.org/abs/1610.04286.

[Sadeghi 2016] Sadeghi, F. and S. Levine (2016). "CAD2RL: Real Single-Image Flight without a Single Real Image". In: arXiv: 1611.04201. URL: http://arxiv.org/abs/1611.04201.

[Sallab 2016] Sallab, A. E., M. Abdou, E. Perot, and S. Yogamani (2016). "End-to-End Deep Reinforcement Learning for Lane Keeping Assist". In: Nips, pp. 1–9. arXiv: 1612.04340. URL: http://arxiv.org/abs/1612.04340.

[Sallab 2017] Sallab, A. E., M. Abdou, E. Perot, and S. Yogamani (2017). "Deep Reinforcement Learning framework for Autonomous Driving". In: DOI: 10.2352/ISSN.2470-1173.2017.19.AVM-023. arXiv: 1704.02532. URL: http://arxiv.org/abs/1704.02532%7B%5C%%7D0Ahttp://dx.doi.org/10.2352/ISSN.2470-1173.2017.19.AVM-023.

[Santana 2016] Santana, E. and G. Hotz (2016). "Learning a Driving Simulator". In: *arXiv preprint*, pp. 1–8. arXiv: 1608.01230. URL: http://arxiv.org/abs/1608.01230.

[Sauer 2018] Sauer, A., N. Savinov, and A. Geiger (2018). "Conditional Affordance Learning for Driving in Urban Environments". In: CoRL, pp. 1–16. arXiv: 1806.06498. URL: http://arxiv.org/abs/1806.06498.

[Savva 2017] Savva, M., A. X. Chang, A. Dosovitskiy, T. Funkhouser, and V. Koltun (2017). "MINOS: Multimodal Indoor Simulator for Navigation in Complex Environments". In: pp. 1–14. arXiv: 1712.03931. URL: http://arxiv.org/abs/1712.03931.

[Schaul 2016] Schaul, T., J. Quan, I. Antonoglou, and D. Silver (2016). "Prioritized Experience Replay". In: *4th International Conference on Learning Representations, ICLR 2016*, pp. 1–21. arXiv: 1511.05952. URL: http://arxiv.org/abs/1511.05952.

[Schulman 2017] Schulman, J. (2017). *Policy Gradient Methods : Lecture from UC Berkeley*. http://rll.berkeley.edu/deeprlcourse/docs/lec2.pdf. Accessed: 2017.

[Schulman, Levine, 2015] Schulman, J., S. Levine, P. Moritz, M. I. Jordan, and P. Abbeel (2015). "Trust Region Policy Optimization". In: *International conference on machine learning*, pp. 1889–1897. arXiv: 1502.05477. URL: http://arxiv.org/abs/1502.05477.

[Schulman, Moritz, 2015] Schulman, J., P. Moritz, S. Levine, M. Jordan, and P. Abbeel (2015). "High-Dimensional Continuous Control Using Generalized Advantage Estimation". In: pp. 1–14. arXiv: 1506.02438. URL: http://arxiv.org/abs/1506.02438.

[Schulman 2017] Schulman, J., F. Wolski, P. Dhariwal, A. Radford, and O. Klimov (2017). "Proximal Policy Optimization Algorithms". In: *arXiv preprint*, pp. 1–12. arXiv: 1707.06347. URL: http://arxiv.org/abs/1707.06347.

[Schwarz 2018] Schwarz, J., J. Luketina, W. M. Czarnecki, A. Grabska-Barwinska, Y. W. Teh, R. Pascanu, and R. Hadsell (2018). "Progress & Compress: A scalable framework for continual learning". In: *Proceedings of the 35th International Conference on Machine Learning*. PMLR,

pp. 4528–4537. arXiv: 1805.06370. URL: http://proceedings.mlr.press/v80/schwarz18a.html.

[Shah 2017] Shah, S., D. Dey, C. Lovett, and A. Kapoor (2017). "AirSim: High-Fidelity Visual and Physical Simulation for Autonomous Vehicles". In: pp. 1–14. arXiv: 1705.05065. URL: http://arxiv.org/abs/1705.05065.

[Shelhamer 2019] Shelhamer, E., P. Mahmoudieh, M. Argus, and T. Darrell (2019). "Loss is its own reward: Self-supervision for reinforcement learning". In: *5th International Conference on Learning Representations, ICLR 2017 - Workshop Track Proceedings*. arXiv: 1612.07307.

[Shkurti 2021] Shkurti, F. (2021). *CSC2626: Imitation Learning for Robotics, Winter 2021*. http://www.cs.toronto.edu/~florian/courses/csc2626w21/.

[Shorten 2019] Shorten, C. and T. M. Khoshgoftaar (2019). "A survey on Image Data Augmentation for Deep Learning". In: *Journal of Big Data* 6.1. ISSN: 21961115. DOI: 10.1186/s40537-019-0197-0. URL: https://doi.org/10.1186/s40537-019-0197-0.

[Silver 2015] Silver, D. (2015). *Lectures on Reinforcement Learning*. URL: https://www.davidsilver.uk/teaching/.

[Silver 2016] Silver, D., A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis (2016). "Mastering the game of Go with deep neural networks and tree search." In: *Nature* 529.7587, pp. 484–9. ISSN: 1476-4687. DOI: 10.1038/nature16961. URL: http://www.ncbi.nlm.nih.gov/pubmed/26819042.

[Silver 2014] Silver, D., G. Lever, D. Technologies, G. U. Y. Lever, and U. C. L. Ac (2014). "Deterministic Policy Gradient (DPG)". In: *Proceedings of the 31st International Conference on Machine Learning* 32.1, pp. 387–395. ISSN: 1938-7228. URL: http://proceedings.mlr.press/v32/silver14.html.

[Silver 2017] Silver, D., J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, T. Lillicrap, F. Hui, L. Sifre, G. van den Driessche, T. Graepel, and D. Hassabis (2017). "Mastering the game of Go without human knowledge." In: *Nature* 550.7676, pp. 354–359. ISSN: 1476-4687. DOI: 10.1038/nature24270. URL: http://www.ncbi.nlm.nih.gov/pubmed/29052630.

[Simonyan 2015] Simonyan, K. and A. Zisserman (2015). "Very deep convolutional networks for large-scale image recognition". In: *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*, pp. 1–14. arXiv: 1409.1556.

[Singh 2005] Singh, S., A. G. Barto, and N. Chentanez (2005). "Intrinsically motivated reinforcement learning". In: *Advances in Neural Information Processing Systems*. ISSN: 10495258.

[Song 2017] Song, S., F. Yu, A. Zeng, A. X. Chang, M. Savva, and T. Funkhouser (2017). "Semantic Scene Completion from a Single Depth Image". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1746–1754. arXiv: 1611.08974. URL: http://arxiv.org/abs/1611.08974.

[Srivastava 2014] Srivastava, N., G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov (2014). "Dropout: A Simple Way to Prevent Neural Networks from Overfitting". In: *Journal*

of Machine Learning Research 15, pp. 1929–1958. URL: http://jmlr.org/papers/v15/srivastava14a.html.

[Sun 2020] Sun, P., H. Kretzschmar, X. Dotiwalla, A. Chouard, V. Patnaik, P. Tsui, J. Guo, Y. Zhou, Y. Chai, B. Caine, V. Vasudevan, W. Han, J. Ngiam, H. Zhao, A. Timofeev, S. Ettinger, M. Krivokon, A. Gao, A. Joshi, Y. Zhang, J. Shlens, Z. Chen, and D. Anguelov (2020). "Scalability in perception for autonomous driving: Waymo open dataset". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 2446–2454.

[Sutton 2017] Sutton, R. S. and A. G. Barto (2017). *Reinforcement Learning: An Introduction*. MIT press.

[Szegedy 2015] Szegedy, C., W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich (Aug. 2015). "Going Deeper with Convolutions". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1–9.

[Talpaert 2019] Talpaert, V., I. Sobh, B. R. Kiran, P. Mannion, S. Yogamani, A. El-Sallab, and P. Perez (2019). "Exploring applications of deep reinforcement learning for real-world autonomous driving systems". In: arXiv: 1901.01536. URL: http://arxiv.org/abs/1901.01536.

[Tang 2017] Tang, H. and T. Haarnoja (2017). *Soft Q-learning - BAIR blog; Berkeley*. http://bair.berkeley.edu/blog/2017/10/06/soft-q-learning/.

[Teh 2017] Teh, Y., V. Bapst, W. M. Czarnecki, J. Quan, J. Kirkpatrick, R. Hadsell, N. Heess, and R. Pascanu (2017). "Distral: Robust Multitask Reinforcement Learning". In: *Advances in Neural Information Processing Systems* 30, pp. 4496–4506. ISSN: 0921898X. arXiv: 1511.06342. URL: https://arxiv.org/abs/1707.04175.

[Tobin, Biewald, 2017] Tobin, J., L. Biewald, R. Duan, M. Andrychowicz, A. Handa, V. Kumar, B. McGrew, A. Ray, J. Schneider, P. Welinder, W. Zaremba, and P. Abbeel (2017). "Domain Randomization and Generative Models for Robotic Grasping". In: *IEEE International Conference on Intelligent Robots and Systems*, pp. 3482–3489. ISSN: 21530866. DOI: 10.1109/IROS.2018.8593933. arXiv: arXiv:1710.06425v1.

[Tobin, Fong, 2017] Tobin, J., R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel (2017). "Domain randomization for transferring deep neural networks from simulation to the real world". In: *IEEE International Conference on Intelligent Robots and Systems* 2017-Septe, pp. 23–30. ISSN: 21530866. DOI: 10.1109/IROS.2017.8202133. arXiv: arXiv:1703.06907v1.

[Todorov 2012] Todorov, E., T. Erez, and Y. Tassa (2012). "MuJoCo : A physics engine for model-based control". In: *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 5026–5033. ISBN: 9781467317368. DOI: 10.1109/IROS.2012.6386109.

[Toromanoff 2019] Toromanoff, M., E. Wirbel, and F. Moutarde (2019). "End-to-End Model-Free Reinforcement Learning for Urban Driving using Implicit Affordances". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 7153–7162. arXiv: 1911.10868. URL: http://arxiv.org/abs/1911.10868.

[Toromanoff 2018] Toromanoff, M., E. Wirbel, C. Vejarano, X. Perrotton, and F. Moutarde (2018). "End to End Vehicle Lateral Control Using a Single Fisheye Camera". In: pp. 3613–3619. arXiv: 1808.06940. URL: http://arxiv.org/abs/1808.06940.

[Tran 2018] Tran, D., H. Wang, L. Torresani, and J. Ray (2018). "A Closer Look at Spatiotemporal Convolutions for Action Recognition". In: *Cvpr*, pp. 6450–6459. arXiv: arXiv:1711.11248v3.

URL: http://openaccess.thecvf.com/content_cvpr_2018/html/Tran_A_Closer_Look_CVPR_2018_paper.html.

[Van Hasselt 2016] Van Hasselt, H., A. Guez, M. Hessel, V. Mnih, and D. Silver (2016). "Learning values across many orders of magnitude". In: *Advances in Neural Information Processing Systems*, pp. 4294–4302. ISSN: 10495258. arXiv: 1602.07714.

[Vecerik 2017] Vecerik, M., T. Hester, J. Scholz, F. Wang, O. Pietquin, B. Piot, N. Heess, T. Rothörl, T. Lampe, and M. Riedmiller (2017). "Leveraging Demonstrations for Deep Reinforcement Learning on Robotics Problems with Sparse Rewards". In: *arXiv preprint*, pp. 1–10. arXiv: 1707.08817. URL: http://arxiv.org/abs/1707.08817.

[Vinyals 2016] Vinyals, O., C. Blundell, T. Lillicrap, K. Kavukcuoglu, and D. Wierstra (2016). "Matching Networks for One Shot Learning". In: *Advances in neural information processing systems* 29, pp. 3630–3638. arXiv: 1606.04080. URL: http://arxiv.org/abs/1606.04080.

[Vukotić 2017] Vukotić, V., S. L. Pintea, C. Raymond, G. Gravier, and J. C. van Gemert (2017). "One-step time-dependent future video frame prediction with a convolutional encoder-decoder neural network". In: *International Conference on Image Analysis and Processing*, pp. 140–151. ISSN: 16113349. DOI: 10.1007/978-3-319-68560-1_13. arXiv: arXiv:1702.04125v2.

[J. X. Wang 2016] Wang, J. X., Z. Kurth-Nelson, D. Tirumala, H. Soyer, J. Z. Leibo, R. Munos, C. Blundell, D. Kumaran, and M. Botvinick (2016). "Learning to reinforcement learn". In: *arXiv preprint*. arXiv: 1611.05763. URL: http://arxiv.org/abs/1611.05763.

[S. Wang 2017] Wang, S., M. Bai, G. Mattyus, H. Chu, W. Luo, B. Yang, J. Liang, J. Cheverie, S. Fidler, and R. Urtasun (2017). "TorontoCity: Seeing the World with a Million Eyes". In: *Proceedings of the IEEE International Conference on Computer Vision* 2017-Octob, pp. 3028–3036. ISSN: 15505499. DOI: 10.1109/ICCV.2017.327. arXiv: 1612.00423.

[Wang, Bapst, 2016] Wang, Z., V. Bapst, N. Heess, V. Mnih, R. Munos, K. Kavukcuoglu, and N. de Freitas (2016). "Sample Efficient Actor-Critic with Experience Replay". In: 2016. arXiv: 1611.01224. URL: http://arxiv.org/abs/1611.01224.

[Wang, Schaul, 2016] Wang, Z., T. Schaul, M. Hessel, H. van Hasselt, M. Lanctot, and N. de Freitas (2016). "Dueling Network Architectures for Deep Reinforcement Learning". In: *International conference on machine learning*, pp. 1995–2003. arXiv: 1511.06581. URL: http://arxiv.org/abs/1511.06581.

[2019] *wayve blog* (2019). https://wayve.ai/blog/learned-urban-driving/.

[Williams 2012] Williams, R. (2012). "Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning". In: *Journal of Pain and Symptom Management* 44.5, p. 794. ISSN: 18736513. DOI: 10.1016/S0885-3924(12)00520-9.

[Wolf 2017] Wolf, P., C. Hubschneider, M. Weber, A. Bauer, J. Hartl, F. Durr, and J. M. Zollner (2017). "Learning how to drive in a real world simulation with deep Q-Networks". In: *IEEE Intelligent Vehicles Symposium, Proceedings*. Iv, pp. 244–250. ISBN: 9781509048045. DOI: 10.1109/IVS.2017.7995727.

[2016] *World Rally Championship* (2016). https://www.wrcthegame.com/.

[Wu 2018] Wu, Y., Y. Wu, G. Gkioxari, and Y. Tian (2018). "Building Generalizable Agents with a Realistic and Rich 3D Environment". In: *6th International Conference on Learning*

Representations (ICLR 2018) Workshop track, pp. 1–15. arXiv: 1801.02209. URL: http://arxiv.org/abs/1801.02209.

[Wu 2017] Wu, Y., E. Mansimov, S. Liao, R. Grosse, and J. Ba (2017). "Scalable trust-region method for deep reinforcement learning using Kronecker-factored approximation". In: *31st Conference on Neural Information Processing Systems (NIPS 2017)*, pp. 1–14. arXiv: 1708.05144. URL: http://arxiv.org/abs/1708.05144.

[Wydmuch 2019] Wydmuch, M., M. Kempka, and W. Jaśkowski (2019). "Vizdoom competitions: Playing doom from pixels". In: *IEEE Transactions on Games* 11.3, pp. 248–259. ISSN: 24751510. DOI: 10.1109/TG.2018.2877047. arXiv: 1809.03470.

[Wymann 2014] Wymann, B., Eric Espié, Christophe Guionneau, D. Christos, C. Rémi, and S. Andrew (2014). "TORCS: The Open Racing Car Simulator". In: pp. 1–5. URL: https://pdfs.semanticscholar.org/b9c4/d931665ec87c16fcd44cae8fdaec1215e81e.pdf.

[Xu 2017] Xu, H., Y. Gao, F. Yu, and T. Darrell (2017). "End-to-end learning of driving models from large-scale video datasets". In: *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017* 2017-Janua, pp. 3530–3538. DOI: 10.1109/CVPR.2017.376. arXiv: arXiv:1612.01079v2.

[Yu 2020] Yu, F., H. Chen, X. Wang, W. Xian, Y. Chen, F. Liu, V. Madhavan, and T. Darrell (2020). "BDD100K: A Diverse Driving Dataset for Heterogeneous Multitask Learning". In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 2633–2642. ISSN: 10636919. DOI: 10.1109/CVPR42600.2020.00271. arXiv: 1805.04687.

[Zeiler 2014] Zeiler, M. D. and R. Fergus (2014). "Visualizing and understanding convolutional networks". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 8689 LNCS.PART 1, pp. 818–833. ISSN: 16113349. DOI: 10.1007/978-3-319-10590-1_53. arXiv: arXiv:1311.2901v3.

[Zhelo 2018] Zhelo, O., J. Zhang, L. Tai, M. Liu, and W. Burgard (2018). "Curiosity-driven Exploration for Mapless Navigation with Deep Reinforcement Learning". In: *ICRA 2018 Workshop in Machine Learning in the Planning and Control of Robot Motion*. arXiv: 1804.00456. URL: http://arxiv.org/abs/1804.00456.

[B. Zhou 2019] Zhou, B., P. Krähenbühl, and V. Koltun (2019). "Does computer vision matter for action?" In: *Science Robotics* 4.30, pp. 1–12. ISSN: 24709476. DOI: 10.1126/scirobotics.aaw6661. arXiv: 1905.12887.

[W. Zhou 2019] Zhou, W., J. S. Berrio, S. Worrall, and E. Nebot (2019). "Automated evaluation of semantic segmentation robustness for autonomous driving". In: *IEEE Transactions on Intelligent Transportation Systems* 21.5, pp. 1951–1963.

[Zhu 2017] Zhu, Y., R. Mottaghi, E. Kolve, J. J. Lim, A. Gupta, L. Fei-Fei, and A. Farhadi (2017). "Target-driven visual navigation in indoor scenes using deep reinforcement learning". In: *Proceedings - IEEE International Conference on Robotics and Automation*, pp. 3357–3364. ISSN: 10504729. DOI: 10.1109/ICRA.2017.7989381. arXiv: arXiv:1609.05143v1.

ECOLE DOCTORALE

Titre: Exploration des algorithmes d'apprentissage par renforcement pour la perception et le controle d'un véhicule autonome

Mots clés: Apprentissage par renforcement, Véhicule autonome, Apprentissage de bout-en-bout, Vision par ordinateur, Tâche auxiliaire

Résumé: L'apprentissage par renforcement est une approche permettant de résoudre un problème de prise de décision séquentielle. Dans ce formalisme, un agent autonome interagit avec un environnement et reçoit des récompenses en fonction des décisions qu'il prend. L'objectif de l'agent est de maximiser le montant total des récompenses qu'il obtient. Dans le paradigme de l'apprentissage par renforcement, l'agent apprend par essais-erreurs la politique (séquence d'actions) qui donne les meilleures récompenses.
Dans cette thèse, nous nous concentrons sur son application à la perception et au contrôle d'un véhicule autonome. Pour rester proche des conditions d'un conducteur humain, seule la caméra embarquée est utilisée comme capteur d'entrée. Nous nous focalisons en particulier sur l'apprentissage de bout-en-bout de la conduite, c'est-à-dire une correspondance directe entre les informations provenant de l'environnement et l'action choisie par l'agent. Ce type d'apprentissage pose cependant certains défis : les grandes dimensions des espaces d'états et d'actions ainsi que l'instabilité et la faiblesse du signal de l'apprentissage par renforcement pour entraîner des réseaux de neurones profonds. Les approches que nous avons mises en œuvre pour faire face à ces défis reposent sur l'utilisation de l'information sémantique (segmentation d'images). En particulier, nous explorons l'apprentissage conjoint de l'information sémantique et de la navigation. Nous montrons que ces méthodes sont prometteuses et permettent de lever certains verrous. D'une part combiner l'apprentissage supervisé de la segmentation à l'apprentissage par renforcement de la navigation améliore les performances de l'agent, ainsi que sa capacité à généraliser à un environnement inconnu. D'autre part, cela permet d'entraîner un agent qui sera plus robuste aux événements inattendus et capable de prendre des décisions en limitant les risques. Les expériences sont menées en simulation, et de nombreuses comparaisons avec les méthodes de l'état de l'art sont effectuées.

Title: Exploration of reinforcement learning algorithms for autonomous vehicle visual perception and control

Keywords: Reinforcement Learning, Autonomous Vehicle, End-to-end training, Auxiliary task, Computer vision

Abstract: Reinforcement learning is an approach to solve a sequential decision making problem. In this formalism, an autonomous agent interacts with an environment and receives rewards based on the decisions it makes. The goal of the agent is to maximize the total amount of rewards it receives. In the reinforcement learning paradigm, the agent learns by trial and error the policy (sequence of actions) that yields the best rewards.
In this thesis, we focus on its application to the perception and control of an autonomous vehicle. To stay close to human driving, only the onboard camera is used as input sensor. We focus in particular on end-to-end training, i.e. a direct mapping between information from the environment and the action chosen by the agent. However, training end-to-end reinforcement learning for autonomous driving poses some challenges: the large dimensions of the state and action spaces as well as the instability and weakness of the reinforcement learning signal to train deep neural networks.
The approaches we implemented are based on the use of semantic information (image segmentation). In particular, this work explores the joint training of semantic information and navigation. We show that these methods are promising and allow to overcome some limitations. On the one hand, combining segmentation supervised learning with navigation reinforcement learning improves the performance of the agent and its ability to generalize to an unknown environment. On the other hand, it enables to train an agent that will be more robust to unexpected events and able to make decisions limiting the risks. Experiments are conducted in simulation, and numerous comparisons with state of the art methods are made.

Institut Polytechnique de Paris
91120 Palaiseau, France