university of groningen

faculty of mathematics and natural sciences

MASTER THESIS

Modelling human driving behaviour using Generative Adversarial Networks

Author:
Diederik P. GREVELING

Supervisor I:
Dipl.-Ing. Gereon HINZ
Supervisor II:
Frederik DIEHL
Examiner I:
Prof. dr. Nicolai PETKOV
Examiner II:
dr. Nicola STRISCIUGLIO

A Msc thesis submitted in fulfillment of the requirements for the degree of Master of Science

in the

Intelligent Systems Group
Johann Bernoulli Institute for Mathematics and Computer Science

January 25, 2018

Declaration of Authorship

I, Diederik P. GREVELING, declare that this thesis titled, "Modelling human driving behaviour using Generative Adversarial Networks" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed:

Date:

"Never underestimate the smallest of errors."

Diederik Greveling

University of Groningen

Abstract

Faculty of Science and Engineering

Johann Bernoulli Institute for Mathematics and Computer Science

Master of Science

Modelling human driving behaviour using Generative Adversarial Networks

by Diederik P. GREVELING

In this thesis, a novel algorithm is introduced which combines Wasserstein Generative Adversarial Networks with Generative Adversarial Imitation Learning which is then applied to learning human driving behaviour. The focus of this thesis is to solve the problem of mode collapse and vanishing gradients from which Generative Adversarial Imitation Learning suffers and show that our implementation performs equally to the original Generative Adversarial Imitation Learning algorithm. The performance of the novel algorithm is evaluated on OpenAI Gym control problems and the NGSIM traffic dataset. The novel algorithm is shown to solve complex control problems on par with Generative Adversarial Imitation Learning and can learn to navigate vehicle trajectories.

Acknowledgements

This thesis would not have succeeded without the help of Frederik Diehl, his support and insight in our weekly meetings greatly progressed the thesis. Gereon Hinz deserves praise for being very patient in finding a research topic with me.

I would like to thank Professor Petkov for his feedback, patience and giving me the chance to freely choose a topic of my interest and Dr Strisciuglio for reviewing the thesis.

I would like to thank all the people at fortiss GmbH for their insights, interesting conversations and making me feel part of the team for six months.

Last but not least I would like to thank my fellow students for the many projects we did together.

Contents

Declaration of Authorship iii

Abstract vii

Acknowledgements ix

1 Introduction 1

2 Related Work 5
2.1 Human Driver Models . . . 5
2.2 Imitation Learning . . . 6
2.3 Generative Adverserial Networks . . . 7
2.4 Training Data . . . 8
2.4.1 Open AI Gym . . . 8
2.4.2 Vehicle trajectory data . . . 8

3 Methods 9
3.1 Preliminaries . . . 9
3.1.1 Markov Decision Process . . . 9
3.1.2 Partially Observable Problems . . . 9
3.1.3 Policy . . . 10
3.1.4 Policy Gradients . . . 10
3.2 Trust Region Policy Optimization . . . 13
3.2.1 Preliminaries . . . 13
3.2.2 TRPO . . . 14
3.2.3 Practical Algorithm . . . 16
3.3 Generalized Advantage Estimation . . . 18
3.4 Generative Adversarial Networks . . . 19
3.4.1 Preliminaries . . . 19
3.4.2 GAN . . . 20
3.4.3 Wasserstein GAN . . . 22
3.5 Generative Adverserial Imitation Learning . . . 24
3.5.1 Preliminaries . . . 24
3.5.2 GAIL . . . 24
3.5.3 WGAIL . . . 26
3.5.4 WGAIL-GP . . . 26

4 Experiments 29
4.1 OpenAI Gym experiments . . . 29
4.1.1 HalfCheetah-v1 . . . 30
4.1.2 Walker2d-v1 . . . 31
4.1.3 Hopper-v1 . . . 32
4.2 NGSIM experiments . . . 33

4.2.1 Features 34
4.2.2 Results 35

5 Discussion **37**
5.1 Gym experiments 37
5.2 NGSIM experiments 38
5.3 WGAIL performance 39

6 Conclusions **41**

A Methods Appendix **43**
A.1 Lipschitz 43

Bibliography **45**

List of Figures

3.1 GAN Distributions 21
3.2 GAN normal convergence vs Mode Collapse convergence 22
3.3 Jensen Shannon divergence vs Earth Mover distance 23

4.1 OpenAI Gym environments 29
4.2 *HalfCheetah* Experiment results 31
4.3 Walker Experiment results 32
4.4 Hopper Experiment results 33
4.5 Traffic simulator 34
4.6 TrafficSim Experiment results 36

List of Tables

4.1 Traffic simulator Feature Vector 35

Chapter 1

Introduction

Recently, autonomous driving has become an increasingly active research area within the car industry. A surge in publications about autonomous driving as well as technical demonstrations has taken place. And while the industry has already taken large leaps forward there are still major drawbacks to overcome. One of those drawbacks is safety, since developed systems often drive very defensively resulting in frequent waiting times and driver interventions. In order to improve both the overall performance and safety of an autonomous driving system, further research needs to be done in creating driving models which model human driving behaviour.

Early driving models were rule-based using a set of parameters to determine the behaviour of the driver. These models often imply conditions about the road and driving behaviour [1], [2]. Car following models like the Intelligent Driver Model (IDM) [3] are able to capture realistic driving to a certain degree. IDM, for example, captures realistic braking behaviour and acceleration-deceleration asymmetries. IDM does, however, make assumptions about the driver behaviour which introduces a bias in the model.

Due to the technological advances within the car industry and traffic observation systems, it has become easier to obtain data about driving behaviour. For example, roadside cameras monitor traffic flow and onboard computers capture data about the driver. In the field of Machine Learning, this data has been used for a wide array of prediction and control problems. Recently research from Nvidia has shown that a lane following model can be learned using an End-To-End learning technique[4]. This was done by training a Convolutional Neural Network directly on data captured by the onboard computer and front mounted cameras. Whilst this system was shown to work 98% autonomously on a small piece of single-lane road and multi-lane highway, real-life applications are still limited. This process is a form of Imitation Learning (IL).

IL algorithms take data from an *expert* and model a policy which behaves similarly to the expert. IL algorithms use expert data to train a parametric model which represents a policy. A neural network can be used for such a model. There are many different types of IL algorithms from which one of the most basic IL algorithms is Behavioral Cloning (BC). BC algorithms handle expert data as supervised training data resulting in a supervised learning problem. For imitating driving behaviour a BC algorithm can be applied by mapping the input from mounted cameras to the steering angle of the car [4], [5]. However, BC algorithms often fail in practice because the trained policy has difficulty handling states which were not represented in

the expert training set. Thus, small errors tend to compound resulting in major failure [6]. Due to these insufficiencies alternative IL methods were developed which aim to fix these problems.

Reinforcement Learning (RL) aims to train a policy which maximises a reward function. A higher reward indicates a better performing policy, i.e. closer to the expert. Since RL trains a policy based on a reward function instead of training on the target data directly, it can handle unencountered states. Which, in turn, means the RL generalises better than other IL algorithms. However, in the case of prediction and control problems the reward function is often unknown. Meaning that first, a reward function needs to be defined before a policy can be trained.

Inverse Reinforcement Learning (IRL) first finds the reward function which is 'encoded' within the expert data [7]. Once the reward function is found RL, can be applied to train the policy. While IRL algorithms are theoretically sound, in practice they tend to train relatively slow since finding the reward function is computationally expensive.

Generative Adversarial Imitation Learning (GAIL) is a direct policy optimisation algorithm meaning that no expert cost function needs to be learned [8]. GAIL is derived from the basic Generative Adversarial Network (GAN) method [9]. In a GAN a discriminator function learns to differentiate between states and actions from the expert and states and actions produced by a generator. The generator produces samples which should pass for expert states and actions. Trust Region Policy Optimisation (TRPO), a direct optimisation method [10], functions as the generator algorithm in GAIL. Using TRPO removes the need to optimise an expert cost function. GAIL performs well on some benchmark tasks and has shown promising results in modelling human driving behaviour [8], [11].

GANs are suitable for a wide array of learning tasks. However, they tend to suffer from mode collapse and vanishing gradients [12]. Mode collapse happens when the generator produces the same output for a large number of different inputs. Vanishing gradients result in weights not being updated and hinder learning. Since GAIL also uses adversarial training, it is also prone to mode collapse and vanishing gradients. A Wasserstein generative adversarial network (WGAN) aims to solve these problems by optimising the Earth Mover distance instead of the Jenson-Shannon divergence optimised in traditional GANs [13].

In this thesis, we aim to combine GAIL with a WGAN and in doing so removing mode collapse and vanishing gradients from GAIL. This new algorithm is dubbed Wasserstein Generative Adversarial Imitation Learning or WGAIL for short. Fixing these problems should improve the training performance of GAIL. Both GAIL and WGAIL are trained on the OpenAI Gym environments set [14] so a comparison can be made between the two algorithms. This should show whether changing to a Wasserstein GAN improves GAIL. WGAIL will also be used to model human driving behaviour by applying it to the NGSIM traffic dataset [15].

The main contributions of this thesis are:

- A novel algorithm is proposed which solves the problem of mode collapse for GAIL.
- A comparison is made between GAIL and WGAIL for OpenAI gym environments using techniques that improve and stabilize training.

- WGAIL is shown to learn driving trajectories using the NGSIM traffic dataset.

Chapter 2

Related Work

Understanding human driver behaviour is one of the key concepts for creating realistic driving simulations and improving autonomous driving regarding reliability and safety [11], [16]. Hence, modelling human driving behaviour is necessary and has thus been a topic with a lot of research interest.

In the next sections, the related work to the research topic is discussed. First, a general introduction about early attempts for modelling human driving behaviour is given, then Imitation Learning is described and how it can be used for modelling human driving behaviour. Lastly, we discuss Generative Adversarial Networks and its variants.

2.1 Human Driver Models

In the past, a lot of research has been done into modelling driving behaviour, in early attempts these models were based on rules and modelled using simple equations. Hyperparameters needed to be set for these models to determine a specific type of driving behaviour. However, while these early models can simulate traffic flow, it is hard to determine the hyperparameters for actual human driving behaviour [1], [2]. Since these single lane car-following models are useful, they were extended to simulate real traffic more closely. The Intelligent Driver Model (IDM) simulates realistic braking behaviour and asymmetries between acceleration and deceleration [3]. Additions of other algorithms include adding multiple lanes, lane changes and adding parameters which directly relate to the driving behaviour of a human. The "politeness" parameter in the MOBIL model captures intelligent driving behaviour in terms of steering and acceleration [17]. When a driver performs a lane-changing manoeuvre strategic planning is often involved. These models, however, do not capture this strategic planning behaviour which is necessary to model human driving behaviour.

Until recently, many attempts at modelling human driving behaviour have focused on one task, such as lane changing [18], [19] and stop behaviour [20]. These methods were, for example, able to predict when a human would change lanes with high confidence.

Current work focuses on trying to model human driving behaviour directly using data recorded from actual human drivers. Bayesian Networks can be used for learning overall driving behaviour, where the parameters and structure of the system are directly inferred from the data [21]. The same idea can be applied to Generative Algorithms, where a model is learned directly from the data without setting

hyperparameters which influence the resulting agents driving behaviour. Imitation Learning (IL) methods also learn from the data by training a policy directly and will be the focus of the next section.

2.2 Imitation Learning

A policy, maps states to actions. States might be the sensor output in a vehicle and actions are the acceleration and steering angle. Using Imitation Learning (IL) methods, a policy can be learned which is directly derived from the data. This data can be provided through human demonstration, in our case human driving data. From this data, a policy can be learned which acts in the same way as an expert. IL methods have been successful for a variety of applications including outdoor mobile robot navigation [22] and autonomous driving [4].

In early attempts to model human driving behaviour, Behavioral Cloning (BC) was applied. Behavioral Cloning is a supervised learning problem where a model is fitted to a dataset. For example, a neural network can be trained on images of the road and learns to follow it by mapping the angle of the steering wheel to certain input images [5]. This method has been extended to work in real driving scenarios, where a car was able to safely navigate parking lots, highways and markerless roads [4]. An advantage of these systems is that no assumptions have to be made about, road markings or signs and supporters of the methods claim that this will eventually lead to better performance [4]. While these BC methods are conceptually sound [23], a problem arises when there are states within the dataset which are underrepresented. Meaning that when the system is trained on these states, small inaccuracies will compound during simulation resulting in cascading errors [6]. In the case of driving behaviour, when the system would drift from the centre of the lane, it should correct itself and move back to the centre. Since this does not happen very often for human drivers, data on the recovery manoeuvre is scarce which results in the cascading error problem. Thus, research is done in alternative IL methods.

Inverse Reinforcement Learning (IRL) learns a policy without knowing the reward function. The expert is assumed to have followed an optimal policy, which can be learned after the reward function has been recovered [7]. IRL has been used for modelling human driving behaviour [24]. Since IRL tries to find a policy which behaves the same as the expert, it will also react the same in unseen states. For example, when driving on the highway, the agent knows to return to the centre of the lane when it is close to the side. In behavioural cloning, this situation would have been a problem since states, where the driver is driving at the side of the road, are rather rare. A downside is that it is very computationally expensive to retrieve the expert cost function.

Instead of learning the expert cost function and learning the policy based on this cost function, the policy can be learned directly using direct policy optimisation, thus eliminating the computationally expensive step of retrieving the expert cost function. These methods have been applied successfully to modelling human driving behaviour [25]. With the introduction of the Generative Adversarial Network (GAN) [9] and Generative Adversarial Imitation Learning (GAIL) [8] new methods have become available which can perform well on certain benchmarking tests. GAIL has been used for modelling human driving behaviour by using the recent advances in GANs and the results are promising [11].

2.3 Generative Adverserial Networks

Generative Adversarial Networks are very useful for generating complex outputs. They have been successfully applied to a number of different tasks like generating new images from existing ones with the same structure [9], image super-resolution [26] and probabilistic inference [27].

GANs are based on a two-player minimax game where one network acts as a discriminator which has to learn the difference between real and fake samples. The fake samples are generated by a second network dubbed the generator, whose goal is to generate samples which mimic the expert policy [9]. The overall objective is to find a Nash-equilibrium of a minimax game between the generator and the discriminator. GANs are competitive with other state of the art generative models. Advantages of GANs are that they can represent very sharp, even degenerate distributions and the trained models may gain some statistical advantage because the generator network is not updated directly based on samples.

It is known however that GANs are notoriously hard to train. They require the precise design of the training model, by choosing an objective function which is easy to train or adapting the architecture. One of the reasons for this behaviour is that it is often very hard to find the Nash-equilibrium using standard gradient descent algorithms or that no Nash-equilibrium exists at all [28]. Another disadvantage of GANs is that the generator network is prone to mode collapse, where the generator maps large portions of the input space to the same output, thereby greatly decreasing the variability of produced output. Complete mode collapse is rare. However, partial mode collapse may happen [12].

The Wasserstein GAN aims to solve the problem of mode collapse. While a standard GAN uses the KL divergence to measure how different distributions are, the Wasserstein GAN uses the Wasserstein distance to measure the similarities between two distributions [13]. The Wasserstein model improves stability, solves the problem of mode collapse and supplies useful learning metrics for debugging.

GANs can be extended to the Imitation Learning domain by replacing the generator with an algorithm which optimises the policy directly. The generator produces samples based on a learned policy, which is derived from the performance of the discriminator. Generative Adversarial Imitation Learning (GAIL) uses this technique in combination with Trust Region Policy Optimization (TRPO). TRPO is used because it can iteratively optimise policies with guaranteed monotonic improvement [10]. A surrogate loss function is defined which is subject to a KL divergence. Using Conjugate gradient optimisation TRPO results in monotonically improved steps.

For TRPO to train more stable, Generalized Advantage Estimation (GAE) is used. This estimator discounts the advantage function similar to the TD(λ) algorithm. However, in this case, the advantage instead of the value function is estimated. The goal of GAE is to adjust the variance-bias tradeoff and effectively reduce the variance [29]. TRPO in combination with GAE is able to learn difficult high-dimensional control tasks which were previous out of reach for standard reinforcement learning methods[29].

In GAIL, the TRPO algorithm acts as the generator which results in a model-free Imitation Learning algorithm. The result is a policy which should approximate the

underlying expert policy from the data. The authors note that the algorithm is efficient in terms of expert data. However, it is not very efficient regarding environment interaction [8].

GAIL was used for modelling driving behaviour based on the NGSIM highway dataset [11]. From the NGSIM dataset features like speed, vehicle length, and lane offset were extracted. Lidar beams were also generated from the data to simulate real-life applications. One of the encountered problems was that the algorithm would oscillate between actions, outputting small positive and negative values which do not result in human-like driving behaviour. However, the algorithm performed very well regarding keeping the car on the road. Improvements can be made by engineering a reward function based on hand-picked features.

2.4 Training Data

2.4.1 Open AI Gym

OpenAI Gym environments are a widely used set of tasks which are used for benchmarking Machine Learning Implementations [14]. The environments range from classical control problems like balancing a pole on a cart and driving a car up unto a mountain to Atari games and learning a humanoid to walk. These environments encapsulate difficult learning tasks which have not been solved yet by Machine Learning systems. Since OpenAI Gym environments are easy to integrate and offer a platform for sharing results, it has become the standard for many research groups in benchmarking their systems performance [8], [10], [13].

2.4.2 Vehicle trajectory data

The Next Generation Simulation (NGSIM) dataset is a dataset published by the U.S. Federal Highway Administration (FHWA) which aims to collect traffic data for microscopic modelling and develop behavioural algorithms for traffic simulation [15]. The I-80 freeway dataset contains 45 minutes of the car movement data on the I-80 freeway in three 15 minute periods [30]. These data are used for a wide range of research topics like fuel consumption analysis [31], shockwave analysis [32] and human driving behaviour prediction [11], [16].

While the NGSIM data is very useful for modelling human driving behaviour, the data often contain erroneous values. For example, the velocity of a vehicle not lining up with the position values between two data points. Thus pre-processing is needed to improve the quality of the data [33].

One of the goals of the Providentia project is to collect high-quality traffic data [34]. This is done by building a permanent setup at the A9 highway in Munich using radar, infrared and HD cameras which should, in theory, result in high-quality trajectory data. As of the time of writing, the Providentia data is not yet available. However, implementations using the NGSIM dataset should work on the Providentia data with minimal adjustments.

Chapter 3

Methods

3.1 Preliminaries

3.1.1 Markov Decision Process

A Markov Decision Process (MDP) is a mathematical system used for modelling decision making. It can be described with five elements:

- $\mathcal{S}$ denotes the state space, i.e a finitie set of states.
- $\mathcal{A}$ denotes a set of actions the actor can take at each timestep t.
- $P_a(s, s') = Pr(s_{t+1} = s' | s_t = s, a_t = a)$ denotes the probability that taking action a at timestep t in state s_t will result in state s_{t+1}.
- $R_a(s, s')$ is the expected reward from taking action a and transitioning to s'.
- $y \in [1, 0]$ is the discount factor, which discounts the future reward.

An MDP can be represented as tuple $(\mathcal{S}, \mathcal{A}, P, R, y)$. Using an MDP, the goal is to determine which actions to take given a certain state. For this, a policy π needs to be determined which maps states s to actions a. A policy can either be stochastic $\pi(a|s)$ or deterministic $\pi(s)$.

For modelling human driving behaviour, states can represent the environment around the vehicle, for example, the position of the vehicle, the number of lanes or how many other vehicles are within a specific area of interest and actions can represent the vehicles acceleration or steering angle.

3.1.2 Partially Observable Problems

To model human driving behaviour, scenarios are broken up into a series of *episodes*. These episodes contain a set of rewards, states, and actions. Every episode contains n states corresponding to time t. These states are samples for the initial state s_0. For each time step t the actor chooses an action a_t based on a certain policy π. This policy is a probability distribution where the action is sampled given the state at time t, i.e. $\pi(a_t|s_t)$. Based on the action a_t, the environment will sample the reward R_t and the next state s_t according to some distribution $P(s_{(t+1)}, r_t|s_t, a_t)$. An episode runs for a predetermined number of time steps or until it reaches a terminal state.

To take the best actions according to the environment distribution, the policy π is chosen such that the expected reward is maximised. The expectation is taken over

episodes τ containing a sequence of rewards, actions, and states ending at time $t = n - 1$, i.e. the terminal state.

In a *partially-observable setting* the actor only has access to an observation for the current time step t. Choosing an action based on only the state for the current time step can lead to very noisy decision making. Hence the actor should combine the information from the previous observations creating a history. This is called a partially observable markov decision process (POMDP). A POMDP can be rewritten as an MDP where a state in the MDP would represent the current state and all the states which occurred before the current state.

3.1.3 Policy

A policy maps states s to actions a. A policy can be stochastic or deterministic. Since humans may take different actions while encountering the same state, the focus of this thesis will be on stochastic policies.

A parameterised stochastic policy is a policy with an underlying model which is parameterised by $\theta \in \mathbb{R}^d$. When using a neural network the weights and biases are real-valued. A stochastic policy can be generated from a deterministic policy by generating parameters to a probability distribution and drawing actions from it. For example, the μ and σ of a Gaussian distribution could be generated.

3.1.4 Policy Gradients

Policy Gradient algorithms are a type of Reinforcement algorithms which try to optimise a policy directly by adjusting the parameters θ. I.e. the parameters θ of policy π_θ are updated such that the performance of the policy is increased. The performance of a policy is measured by a value (reward).

This value can be described as a scalar-valued function $R_{s,a}$ which gives a reward for being in state s and performing action a. We want to calculate the gradient for the expected reward $\nabla_\theta E_{\pi_\theta}[R_{s,a}]$. Using likelihood ratios, this can be rewritten as:

$$\nabla_\theta E_{\pi_\theta}[R_{s,a}] = E_{\pi_\theta}[\nabla_\theta \log \pi_\theta(a|s) R_{s,a}] \tag{3.1}$$

where $\nabla_\theta \log \pi_\theta(a|s)$ is the expected score function and $\pi_\theta(a|s)$ is the policy which determines action a given state s.

Equation 3.1 can be estimated by taking an episode τ or a set of episodes. τ is defined as a sequence of states $\tau \equiv (s_0, a_0, s_1, a_1, \ldots, s_n, a_n)$ where n determines the length of the episode. Using these episodes function 3.1 can be estimated as:

$$\nabla_\theta E_{\pi_\theta}[R_\tau] \approx E_\tau\Big[\sum_{t=0}^{n-1} \nabla_\theta \log \pi_\theta(a_t|s_t) R_{s_t,a_t}\Big] \tag{3.2}$$

Equation 3.2 describes using a trajectory as an estimate for the policy gradient function. As Equation 3.2 shows, the policy gradient can be computed without knowledge of the system dynamics. Given a trajectory, the reward R_{s_t,a_t} is calculated

and parameters θ are adjusted such that the log probability $\log p(\tau|\theta)$ increases. We rewrite Equation 3.2 as:

$$\nabla_\theta E_{\pi_\theta}[R_\tau] \approx E_\tau\Big[\sum_{t=0}^{n-1} \nabla_\theta \log \pi_\theta(a_t|s_t) \sum_{t'=t}^{n-1} R_{s_{t'},a_{t'}}\Big] \tag{3.3}$$

where we sum over the rewards $R_{s,a}$ from t to $n-1$ for t'. Variance can be reduced by introducing a baseline function $b(s_t)$. The baseline function is subtracted from the immediate reward function $R_{s,a}$ resulting in the following function:

$$\nabla_\theta E_{\pi_\theta}[R_\tau] \approx E_\tau\Big[\sum_{t=0}^{n-1} \nabla_\theta \log \pi_\theta(a_t|s_t)\Big(\sum_{t=t'}^{n-1} R_{s_{t'},a_{t'}} - b(s_t)\Big)\Big] \tag{3.4}$$

The added term $b(s_t)$ in Equation 3.4 does not influence the minimum of the expectation, i.e. $E_\tau[\nabla_\theta \log \pi_\theta(a_{t'}|s_{t'})b(s_t)] = 0$. A good choice for the baseline function is the state-value function [35]:

$$b(s) \approx V^\pi(s) = E[r_t + r_{t+1} + \ldots + r_{n-1}|s_t = s, a_t \sim \pi] \tag{3.5}$$

Intuitively, choosing the state-value function as the baseline makes sense. For every gradient step a certain reward $R_{s,a}$ is determined. If the reward is high, the policy is updated in such a way that the log probability of taking action a_t is increased because the action was good. Using a baseline we can see if an action was better than expected. Before taking an action the value $V_\pi(s)$ is calculated giving a value for being in a certain state, which is then subtracted from the reward after taking action a_t. This determines how much action a_t improves the reward compared to the expected reward s_t. This is also called the *advantage function*.

In order to reduce the variance further, we can introduce a discount factor for the reward $R_{s,a}$:

$$\nabla_\theta E_{\pi_\theta}[R_\tau] \approx E_\tau\Big[\sum_{t=0}^{n-1} \nabla_\theta \log \pi_\theta(a_t|s_t)\Big(\sum_{t=t'}^{n-1} \gamma^{t'-t} R_{s_{t'},a_{t'}} - b(s_t)\Big)\Big] \tag{3.6}$$

Intuitively, the discount factor decreases the reward of an action taken far into the past. The higher the discount function, the less we value actions taken in the past. The discounted value estimator is described in the following equation:

$$b(s) \approx V^\pi(s) = E\Big[\sum_{t=t'}^{n-1} \gamma^{t'-t} r'_t|s_t = s, a_t \sim \pi\Big] \tag{3.7}$$

A basic policy gradient algorithm is described in algorithm 1 [36]. The discount factor and the baseline function were added in order to decrease the variance.

While REINFORCE is a simple and straightforward algorithm it has some practical difficulties. Choosing a correct step size is often difficult because the statistics of the states and rewards change and the algorithm tends to prematurely converge with

Algorithm 1 REINFORCE algorithm

procedure TRAIN REINFORCE
- Initialise θ at random
- **for** number of training iterations **do**
  - Generate Trajectory
  - **for** each step s_t, a_t in trajectory τ **do**
    - $A_t = \sum_{t=t'}^{n-1} \gamma^{t'-t} R_{s_{t'},a_{t'}} - V(s_t)$ ▷ Advantage function
    - $\theta = \theta + \alpha \sum_{t=0}^{n-1} \nabla_\theta \log \pi_\theta(a_t|s_t) A_t$ ▷ Stochastich Gradient Ascent
  - Refit $V(s)$ using $\sum_{t=t'}^{n-1} \gamma^{t'-t} R_{s_{t'},a_{t'}}$ as the cost

suboptimal behaviour. However, REINFORCE has been used for solving complex problems, given that the training sample set was very large [37].

The REINFORCE algorithm shows how simple a policy gradient algorithm can be. While we do not use the REINFORCE algorithm for modelling human driving behaviour, the policy gradient algorithms described in the next sections are based on this very basic algorithm. The variance reduction techniques will also be incorporated into our system.

3.2 Trust Region Policy Optimization

Trust Region Policy Optimization (TRPO), is a policy gradient algorithm which ensures monotonically improving performance and makes efficient use of data. It does so by specifying a surrogate loss function which is bounded by the KL divergence between action distributions, meaning that the change in state distribution is bounded because the size of the policy update is bounded. Thus, the policy is still improved despite having non-trivial step size.

3.2.1 Preliminaries

Say $\eta(\pi)$ is the expected discounted reward of a stochastic policy π. $\eta(\pi)$ is defined as:

$$\eta(\pi) = E_{s_0,a_0,\ldots\sim\tilde{\pi}}\Big[\sum_{t=0}^{\infty}\gamma^t r(s_t)\Big] \tag{3.8}$$

where $r(s_t)$ is the reward for state s_t, s_0 is the initial state sampled from distribution p_0, a_t is the action given the state $a_t \sim \pi(a_t|s_t)$ and the next state s_{t+1} based on the probability distribution $s_{t+1} \sim P(s_{t+1|s_t,a_t})$.

Key to TRPO is that we can express a new policy $\tilde{\pi}$ in terms of the advantage over π. Intuitively, this tells us how much better or worse the new policy performs over the old one. Thus, the expected return of $\tilde{\pi}$ can be expressed as:

$$\eta(\tilde{\pi}) = \eta(\pi) + E_{s_0,a_0,\ldots\sim\tilde{\pi}}\Big[\sum_{t=0}^{\infty}\gamma^t A_\pi(s_t,a_t)\Big] \tag{3.9}$$

where A_π is the advantage and a_t is sampled from $\tilde{\pi}$: $a_t \sim \tilde{\pi}$. The proof for Equation 3.9 is given by Kakade & Langford [38]. Equation 3.9 can be rewritten in terms of states and actions:

$$\eta(\tilde{\pi}) = \eta(\pi) + \sum_s p_{\tilde{\pi}(s)} \sum_a \tilde{\pi}(a|s) A_\pi(s,a) \tag{3.10}$$

Given that $p_{\tilde{\pi}}$ are the *discounted* visitation frequencies where the sum over s is described in the following equation:

$$p_{\tilde{\pi}} = P(s_0 = s) + \gamma P(s_1 = s) + \gamma^2 P(s_2 = s) + \ldots + \gamma^n P(s_n = s) \tag{3.11}$$

Equation 3.10 shows a very interesting property: since $p_{\tilde{\pi}} > 0$ we can state that when $\sum_s p_{\tilde{\pi}(s)} \sum_a \tilde{\pi}(a|s) A_\pi(s,a) > 0$, i.e every state has a non negative value, the performance of η will always increase. If the advantage A_π is zero for every state, η will stay the same. Using this property, we can determine whether the new policy performs worse or better than the previous policy.

In the case of a deterministic policy where the action is chosen based on the highest value of A_π; $\tilde{\pi}(s) = \max_a A_\pi(s,a)$, and there is at least one state action pair for which

the advantage is positive, the policy will always be improved. If that is not the case then the algorithm has converged. In the continuous case, however, the values are approximated resulting in some states s for which the expected advantage could be negative. Equation 3.11 can be adjusted such that, instead of calculating the visitation frequency over $p_{\tilde{\pi}(s)}$ they are calculated over $p_{\pi}(s)$, as can be seen in Equation 3.12.

$$L(\tilde{\pi}) = \eta(\pi) + \sum_{s} p_{\pi(s)} \sum_{a} \tilde{\pi}(a|s) A_{\pi}(s,a) \tag{3.12}$$

If the policy is parameterised, for example in the case of a neural network, a policy can be described as π_θ where θ are the parameters. In the case of Equation 3.12 with $\pi_\theta(s,a)$ as a differentiable function of θ, if differentiated to the first order the following Equation holds [38]:

$$L_{\pi_{\theta_0}}(\pi_{\theta_0}) = \eta(\pi_{\theta_0}) \tag{3.13}$$

In conclusion, given a sufficiently small step for $\tilde{\pi}$ from π_{θ_0} (initial parameters θ), that improves $L_{\pi_{\theta_0}}$ will also improve η. This will hold for any arbitrary L_{π_θ}. This is the core idea behind TRPO.

Equation 3.13 does not determine the size of the step. The following equation is derived from an equation proposed by Kakade & Langford [38] which introduces a lower bound for Equation 3.12:

$$\eta(\pi_{\text{new}}) \geqslant L_{\pi_{\text{old}}}(\pi_{\text{new}}) - \frac{2\epsilon\gamma}{(1-\gamma)^2}\alpha^2 \tag{3.14}$$

where:

$$\epsilon = \max_{s} |E_{a\sim\pi'(a|s)}[A_{\pi}(a,s)]|$$

The lower bound for Equation 3.14 is valid when π_{new} is defined as the following *mixture* policy:

$$\pi_{\text{new}}(a|s) = (1-\alpha)\pi_{\text{old}}(a|s) + \alpha\pi'(a|s) \tag{3.15}$$

where π' is chosen based on the lowest value for $L_{\pi_{\text{old}}}$, i.e. $\pi' = \min_{\pi'} L_{\pi_{\text{old}}}(\pi')$.

The problem with mixture policies is that they are unwieldy and restrictive in practice [10]. Since the upper bound is only usable when using mixture policies it is not applicable for using it on all type of general stochastic policy classes. Hence, Schulman et al. [10] proposed a policy update scheme which can be used for any stochastic policy class. This is described in the next section.

3.2.2 TRPO

In the previous section, Equation 3.14 showed that the performance of η is improved when a policy update also improved the right-hand side. In this section, we show

how Schulman et al. [10] improve the policy bound shown in Equation 3.14 by extending it to general stochastic policies.

Instead of calculating the mixture policies from Equation 3.15, α is replaced with a distance measure for two policies, in this case π and $\tilde{\pi}$. Schulman et al. [10] propose using the total variation divergence distance $D_{\text{tv}}(p \parallel q) = \frac{1}{2}\sum_i |p_i - q_i|$ for discrete policy distribution p and q. In the case of a continous policy the sum would be replaced by an integral. Equation 3.16 calculates the maximum distance between two policies π and $\tilde{\pi}$.

$$D_{\text{tv}}^{\max}(\pi, \tilde{\pi}) = \max_s D_{\text{tv}}(\pi(\cdot \mid s) \parallel \tilde{\pi}(\cdot \mid s)) \tag{3.16}$$

If α is replaced for Equation 3.16 then the lower bound described in Equation 3.14 still holds [10]. Interestingly, the total variation divergence distance can be linked to the KL divergence with the following relationship:

$$D_{\text{tv}}(p \parallel q)^2 \leqslant D_{\text{KL}}(p \parallel q) \tag{3.17}$$

where D_{KL} is the KL divergence between two distributions p and q. The max KL distance is defined as:

$$D_{\text{kl}}^{\max} = \max_s D_{\text{KL}}(\pi(\cdot \mid s) \parallel \tilde{\pi}(\cdot \mid s)) \tag{3.18}$$

The relationship defined in Equation 3.17 can be applied to Equation 3.14 where α^2 is replaced for $D_{\text{KL}}^{\max}$ resulting in the following function:

$$\eta(\pi_{\text{new}}) \geqslant L_{\pi_{\text{old}}}(\pi_{\text{new}}) - C D_{\text{KL}}^{\max}(\pi, \tilde{\pi}), \tag{3.19}$$
$$\text{where } C = \frac{2\epsilon\gamma}{(1-\gamma)^2}$$

According to Schulman et al. [10] Equation 3.19 can be used in an algorithm which results in monotonically improved updates, $\eta(\pi_0) \leqslant \eta(\pi_1) \leqslant \eta(\pi_2) \leqslant \eta(\pi_3) \ldots \leqslant \eta(\pi_n)$. Intuitively, this means that the new policy will always have an equal or higher reward than the previous policy. The proof is given in the next equation:

$$\begin{aligned} \eta(\pi_{i+1}) \geqslant M_i(\pi_{i+1}) \implies \\ \eta(\pi_i) = M_i(\pi_i) \implies \\ \eta(\pi_{i+1}) - \eta(\pi_i) = M_i(\pi_{i+1}) - M_i(\pi_i) \end{aligned} \tag{3.20}$$

where $M_i(pi) = L_{\pi_i}(\pi) - C D_{\text{KL}}^{\max}(\pi, \pi_i)$. If M is maximised for every step, then η will never decrease.

Equation 3.19 described above can be optimised for parameterised policies π_θ. For brevity, a parameterised policy is described as θ instead of π_θ, for example, $\pi_{\theta_{\text{old}}} = \theta_{\text{old}}$. Equation 3.20 showed that M needs to be maximised. Thus, in the case of a parameterised policy, θ is maximised:

$$\underset{\theta}{\text{maximise}} \left[L_{\theta_{\text{old}}}(\theta) - C D_{\text{KL}}^{\max}(\theta_{\text{old}}, \theta) \right] \tag{3.21}$$

In practice, the C parameter results in relatively small updates. Thus, Schulman et al. [10] impose a trust region policy constraint on the KL divergence. However, a constraint on a KL divergence is in practice slow to solve. Using a heuristic approximation for the KL divergence simplifies the problem. For TRPO, the KL divergence in Equation 3.21 is replaced with the average KL divergence which is imposed by a trust region policy constraint:

$$\begin{aligned} &\underset{\theta}{\text{maximise}} \; L_{\theta_{\text{old}}}(\theta) \\ &\text{subject to } \bar{D}_{KL}^{p_{\theta_{\text{old}}}}(\theta_{\text{old}}, \theta) \leqslant \delta \end{aligned} \tag{3.22}$$

where $\bar{D}_{KL}^{p_{\theta_{\text{old}}}}$ is the average KL distance:

$$\bar{D}_{KL}^{p}(\theta_a, \theta_b) = E_{s \sim p}[D_{KL}(\pi_{\theta_a}(\cdot \mid s) \parallel \tilde{\pi_{\theta_b}}(\cdot \mid s))]$$

Intuitively, TRPO adjusts the policy parameters using this constrained optimisation problem such that the expected total reward of η is optimised. This optimisation of η is subject to a trust region policy constraint which constraints the policy change for each update. According to Schulman et al. [10] Equation 3.22 has the same empirical performance as when $D_{KL}^{\max}$ is used.

3.2.3 Practical Algorithm

In this section, a practical implementation of the TRPO algorithm is shown and a pseudo algorithm is given. The theory described in the previous section is applied to the sample-based case. First, $L_{\theta_{\text{old}}}(\theta)$ in Equation 3.22 can be expanded to the following equation (see Section 3.2.1):

$$\begin{aligned} &\underset{\theta}{\text{maximise}} \sum_s p_{\theta_{\text{old}}}(s) \sum_a \pi_\theta(a|s) A_{\theta_{\text{old}}}(s, a) \\ &\text{subject to } \bar{D}_{KL}^{p_{\theta_{\text{old}}}}(\theta_{\text{old}}, \theta) \leqslant \delta \end{aligned} \tag{3.23}$$

Next, $\sum_s p_{\theta_{\text{old}}}(s)$ is replaced for an expectation where state s is sampled from $p_{\theta_{\text{old}}}$ i.e, $\frac{1}{1-\gamma} E_{s \sim p_{\theta_{\text{old}}}}$. The advantage function $A_{\theta_{\text{old}}}$ is replaced for a state value pair $Q_{\theta_{\text{old}}}$.

Since we do not know the distribution for the actions in an environment, we use importance sampling. Importance sampling is used as variance reduction technique where intuitively we value the actions which have the most impact the most. The sum over the actions $\sum_a$ is replaced by the importance sampling estimator q. Thus, in the sample based case TRPO updates its parameters according to the following function:

$$\underset{\theta}{\text{maximise}} \; E_{s \sim p_{\theta_{\text{old}}}, a \sim q} \left[\frac{\pi_\theta(a \mid s)}{q(a \mid s)} Q_{\pi_{\theta_{\text{old}}}}(s, a) \right] \tag{3.24}$$

$$\text{subject to } E_{s\sim p_{\theta_{\text{old}}}}\left[\bar{D}_{KL}(\theta_{\text{old}}, \theta)\right] \leqslant \delta$$

where $q(a \mid s)$ defines the sampling distribution for a. Since the policy $\pi_{\theta_{\text{old}}}$ is derived from the sampled states, for example a generated trajectory $s_0, a_0, s_1, a_1, \ldots, s_{T-1}, a_{T-1}, s_T$, the sampling distribution is the same as the derived policy, i.e. $q(a \mid s) = \pi_{\theta_{\text{old}}}$.

Schulman et al. [10] propose a practical algorithm for solving Equation 3.24 consisting of three steps:

1. Sample trajectories of state action pairs and calculate the state action Q-values over these trajectories.
2. Calculate the estimated objective and constraint from Equation 3.24 using the previously generated trajectories.
3. Maximise the policy parameters θ by solving the constrained optimization problem using the conjugate gradient algorithm followed by a line search.

The performance of the conjugate gradient algorithm is improved regarding computation speed by not calculating the matrix of gradients directly but by constructing a Fisher information matrix which analytically computes the Hessian of the KL divergence. This improves the computation speed since there is no need for all the policy gradients to be stored.

Algorithm 2 TRPO algorithm

procedure TRAIN TRPO
- Initialise θ at random

for number of training iterations **do**
- Generate Trajectory τ

for each step s_t, a_t in trajectory τ **do**

$Q_t = \sum_{t=t'}^{n-1} \gamma^{t'-t} R_{s_{t'}, a_{t'}}$ ▷ State-Action Values

$O_\tau = E_\tau\left[\frac{\pi_\theta(s \mid a)}{\pi_{\theta_{\text{old}}}(s \mid a)} Q_t\right]$ ▷ estimated objective

$\bar{D}^\tau_{KL} = E_\tau\left[\bar{D}_{KL}(\theta_{\text{old}}, \theta)\right]$ ▷ estimated constraint

$\underset{\theta}{\text{maximise}}\ O_\tau$ subject to $\bar{D}^\tau_{KL} \leqslant \delta$ ▷ perform conjugate gradient

TRPO performed well on high dimensional state spaces, can solve complex sequences of behaviour and is able to learn delayed rewards. Schulman et al. [10] state that TRPO is a good candidate for learning robotic control policies and other large, rich function approximators.

Thus, TRPO could be a good candidate for modelling human driving behaviour. However, since we approach human driving behaviour as an unsupervised learning problem and no reward is defined, TRPO is combined with a Generative Adversarial Network. This will be discussed in the next sections.

3.3 Generalized Advantage Estimation

Policy gradient methods suffer from high variance resulting in poor training performance. Generalized Advantage Estimation (GAE) proposed by Schulman et al. [29] reduces the variance while maintaining a tolerable level of bias. In Section 3.1.4 the discounted value and advantage functions were discussed. In this section, we will write the discounted advantage function in the following form to explain GAE more clearly. The discounted advantage function is described as:

$$A^{\pi,\gamma}(s_t, a_t) = Q^{\pi,\gamma}(s_t, a_t) - V^{\pi,\gamma}(s_t) \tag{3.25}$$

where Q is the state action value, V is the value function, π is the policy, γ is the discount factor and t is the current time step. A TD residual of the Value function V with discount γ is shown in Equation 3.26 [39].

$$\delta_t^V = r^t + \gamma V(s_{t+1}) - V(s_t) \tag{3.26}$$

where V is the value function and r_t is the total reward for the trajectory. The TD residual is the same as the discounted advantage function given that the correct value function $V^{\pi,\gamma}$ is used. The TD residual can be expanded such that a telescoping sum is formed over k.

$$\hat{A}_t^{(k)} := \sum_{l=0}^{k-1} \gamma^l \delta_t^V = -V(s_t) + r_t + \gamma r_{t+1} + \ldots + \gamma^{k-1} r_{t+k-1} + \gamma^k V(s_t + k) \tag{3.27}$$

$\hat{A}_t^{(k)}$ can be considered as an estimator of the advantage function where the bias is influenced by the length of k.

The GAE equation GAE(γ,λ) calculates the exponentially-weighted average over $\hat{A}_t^{(k)}$. This results in a very simple equation shown in Equation 3.28. The derivation of this equation can be found in the appendix.

$$\hat{A}_t^{GAE(\gamma,\lambda)} = \sum_{l=0}^{\infty} (\gamma\lambda)^l \delta_{t+1}^V \tag{3.28}$$

where γ discounts the reward and λ discounts the TD residual. Intuitively, the λ parameter discounts the advantage function, meaning that, if λ is close to 0 the estimator is expected to have low variance and high bias. If λ is close to 1 the estimator is expected to have low bias and high variance. Note that bias is only introduced by the λ parameter when the value function itself introduces bias, i.e. when the value function is inaccurate.

In principle, GAE can be combined with any algorithm that uses advantage estimation. For example, for gradient policy algorithms (see Equation 3.2), the GAE estimator can be incorporated as follows:

$$\nabla_\theta E_{\pi_\theta}[R_\tau] \approx E_\tau \Big[\sum_{t=0}^{n-1} \nabla_\theta \log \pi_\theta(a_t|s_t) \hat{A}_t^{GAE(\gamma,\lambda)} \Big] \tag{3.29}$$

where τ is the episode, R is the reward, and t is the timestep within the episode.

Schulman et al. [29] state that GAE improves convergence speed and results in better performance when used on OpenAI Gym learning tasks. Thus, GAE has an easy formula and is easily integrated with policy optimisation algorithms making it a good option for lowering the variance and improving policy gradient learning.

3.4 Generative Adversarial Networks

In this section, Generative Adversarial Networks (GAN) introduced by Goodfellow et al. [9] are discussed. First, an in-depth explanation is given about the inner workings of GANs. Next, the GAN algorithms are explained and finally, we discuss the shortcomings and improvements over the original GAN algorithm.

3.4.1 Preliminaries

With Generative Adversarial Networks, two models are *pitted* against each other. A generative model will generate samples and the discriminator model must distinguish a generated from a real example. Thus, the discriminator model must learn to classify between samples from the real distribution and samples from the distribution generated by the generator. The generator tries to improve the samples such that they relate as closely as possible to the samples from the real distribution.

GANs can be described intuitively using the *counterfeiter* example [12]. Say we have a counterfeiter producing fake money and a cop trying to distinguish this fake money from genuine money. Over time the cop will get better at identifying the fake money and thus the counterfeiter improves his efforts to make fake money which resembles genuine money even better. This process continues until the fake money is indistinguishable from the real money. This scenario is precisely how GANs learn to model the real distribution where the cop is the discriminator and the counterfeiter is the generator.

The generator and discriminator models can be represented as multilayer perceptrons where $G(z;\theta_g)$ maps an input z to the data space where θ_g are the parameters of the multilayer perceptrons and z is sampled based on prior $P_z(z)$. The discriminator multilayer perceptron $D(x;\theta_d)$ outputs a single scalar where θ_d are the parameters of the network and x represents the data which needs to be classified as either real or fake. $D(x)$ thus gives a probability whether x was drawn from the generators distribution p_g or from the real distribution p_r. The real distribution p_r may be represented by a certain dataset and does not have to be known explicitly. Both D and G are differentiable functions.

3.4.2 GAN

For the training of a GAN, $D(x)$ is trained such that it maximises the probability of classifying a sample x correctly. For example, given that the D multilayer perceptron outputs a sigmoid function, real samples should result in an output of one and generated (fake) samples would be classified with an output of zero given that D can classify the samples perfectly. Hence, D functions as a binary classifier. At the same time G is trained to minimise the loss function $log(1 - D(G(z)))$, i.e. the training of G is based on how well D can classify the generated samples of G.

Different cost functions can be used for both the generator and the discriminator. The optimal solution is a point in parameter space where all other points result in an equal or higher cost. Furthermore, both try to minimise their cost function while only being allowed to change one set of parameters in the case of the discriminator θ_d and in the case of the generator θ_g. This can also be described as a two-player minimax game where two players try to get the best results. Hence this optimisation problem results in a local differential Nash equilibrium which is defined by a tuple of the parameters (θ_d, θ_g).

The training scheme of D and G using the previously described loss functions is defined by the following equation:

$$\min_G \max_D V(D,G) = \mathbb{E}_{x\sim p_r(x)}[logD(x)] + \mathbb{E}_{z\sim p_z(z)}[log(1 - D(G(z)))] \tag{3.30}$$

where $V(D, G)$ is the value function of a 2 player-minimax game with $log(1-D(G(z)))$ as the loss function for G and $log(D(x))$ as the loss function of D. Figure 3.1 shows how the distributions of G and D are formed.

Algorithm 3 shows the basic training procedure for a GAN based on Equation 3.30. For every training iteration m samples are generated from $p_g(z)$ and samples are generated from $p_r(x)$. In the case where the p_r is unknown, samples can be extracted directly from the data. Line 5 and 6 update the discriminator and the generator respectively using the aforementioned loss functions.

Algorithm 3 Minibatch stochastic gradient descent training of GANs

1: **procedure** TRAIN GAN
2: **for** number of training iterations **do**
3: • Sample m noise samples $\{z^{(1)}, \dots, z^{(m)}\}$ from $p_g(z)$
4: • Sample m samples $\{x^{(1)}, \dots, x^{(m)}\}$ from $p_r(x)$
5: $\nabla_{\theta_d} \frac{1}{m} \sum_{i-1}^{m} [logD(x^{(i)}) + log(1 - D(G(z^{(i)})))]$ ▷ D update step
6: $\nabla_{\theta_g} \frac{1}{m} \sum_{i-1}^{m} log(1 - D(G(z^{(i)})))$ ▷ G update step

Generative models, GANs in particular, have some advantages which make them attractive for learning human driving behaviour or hard machine learning tasks overall. GANs can be trained when the data is incomplete and can make predictions based on inputs that are missing data. This is useful in the case of human driving

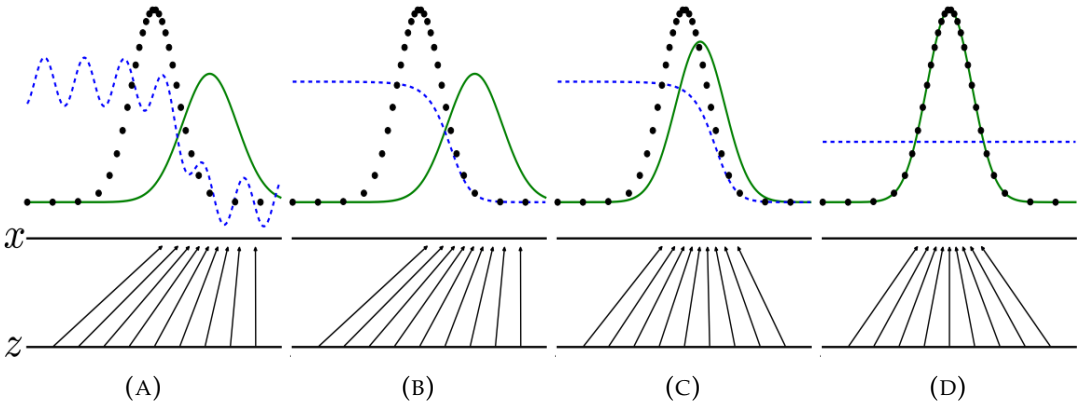


FIGURE 3.1: Displays a GAN that is close to convergence, i.e. p_g is close to p_r (see *(A)*). The black dots represent the data distribution p_r, the green line represents the distribution p_g of the generator G and the blue line represents the distribution of the discriminator D. The bottom horizontal line is the domain z and the upper horizontal line is the domain x of p_r. The vertical arrows represent the mapping from z to x by G. *(B)* The discriminator is trained where it classifies between p_g and p_r. Intuitively, this makes sense since we can see that the distribution of D divides the black dotted and green distributions. *(C)* G was trained using the gradient of D. The arrows indicate that $G(z)$ now maps z more closely to p_r. *(D)* After multiple updates of both G and D, $G(z)$ maps z to x such that it resembles p_r. Thus, G is now able to generate samples which perfectly imitate the real data. In this case discriminator D is unable to discriminate between p_r and p_g and thus $D(x) = \frac{1}{2}$. Figure from Goodfellow [12].

behaviour when we encounter states which are not widely represented in the sample data.

For many Machine Learning tasks, multiple correct outputs are possible for a single input, i.e. multimodal outputs. Human driving behaviour exhibits the same behaviour. For example, when a driver wants to overtake another car he can either do that on the left or right side. However, for some Machine Learning algorithms, this is not the case. When a mean squared error is used for training the algorithm will learn only one correct output to a certain input. For many Machine Learning tasks, this is not sufficient.

One of the disadvantages of a GAN is that they are harder to solve than problems which optimise an objective function since to solve a GAN the Nash equilibrium needs to be found. Another disadvantage entails to the training procedure of a traditional GAN shown in Algorithm 3. Using this training procedure we are unable to measure the performance of the algorithm while training without measuring the trained distribution against the real distribution, i.e. there is no loss function which indicates the performance of the generator with relation to the real distribution.

When GANs train a parameterised model, as is the case with a neural network, they suffer from the issue of non-convergence. Since GANs aim to optimise simultaneous gradient descent, they are bound to converge in function space. However, this property does not hold in parameter space [12]. In practice, GANs are oscillating in terms of output when the generated output from a certain input changes to a different output without converging to an equilibrium.

For GANs, non-convergence can lead to *mode collapse* when training. Mode collapse

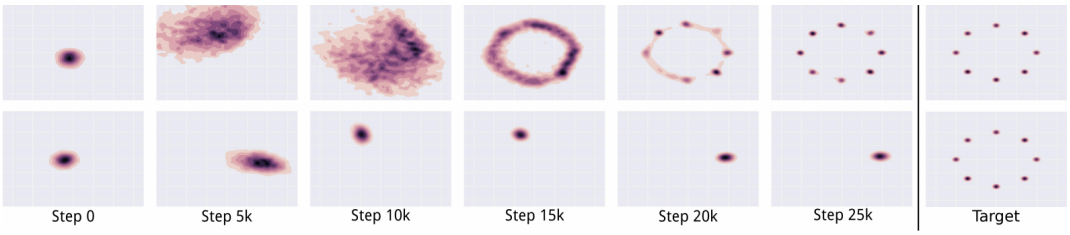


FIGURE 3.2: Shows the learning process of a GAN where the rightmost illustration is the target distribution. This is composed out of multiple 2d Gaussian distributions. *Top* depicts how a GAN should be converging. *Bottom* shows mode collapse happening when training using algorithm 3. The training procedure never converges and instead, it jumps to specific outputs for every 5k of epochs. Figure from Metz et al. [40].

happens when the generator maps multiple input (z) values to the same output value. This is also known as the *helvetica scenario*. According to Goodfellow et al. [12], full mode collapse is rare but partial mode collapse may happen. An example of mode collapse vs regular learning can be seen in Figure 3.2.

In the case of modelling human driving behaviour or any complex model, mode collapse could greatly affect performance. In the next section, we discuss the Wasserstein GAN which improves upon the classic GAN by solving some of the aforementioned disadvantages.

3.4.3 Wasserstein GAN

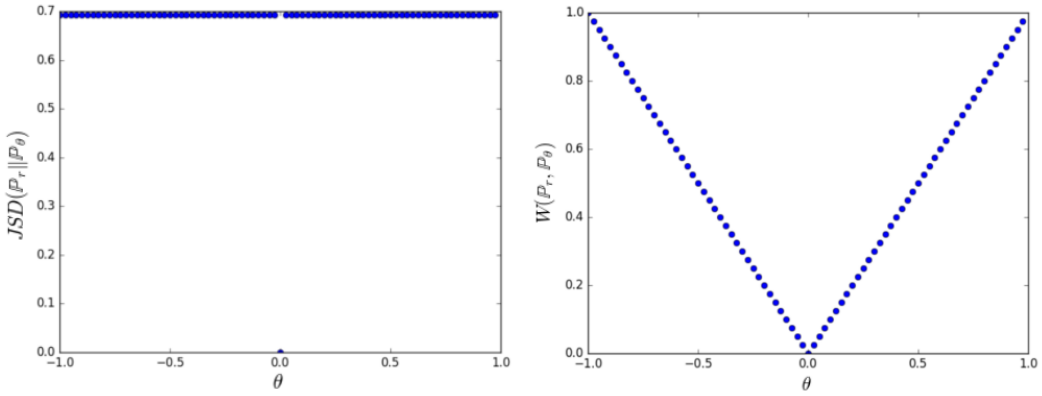
The Wasserstein GAN (WGAN) introduced by Arjovsky et al. [13] aims to solve some of the problems that traditional GANs suffer from. For GANs different divergences can be used for measuring similarities between two distributions [12]. The divergence influences how a GAN converges, meaning that the results of the training procedure may differ when a different divergence is used.

Most GANs seem to optimise the Jensen-Shannon (JS) divergence. Because of this, GANs suffer from the vanishing gradient problem. This can be illustrated by the following example.

Take a model with one parameter θ such that it generates a sample (θ, z) where the distributions overlap fully or do not overlap at all. In Figure 3.3a the JS divergence is shown for different values of θ. As can be seen, the gradient with relation to θ is zero for most values of θ, resulting in a flat graph. If this is the case for the discriminator, the generator will result in a zero gradient. As a result vanishing gradients might happen when training gradients.

Arjovsky et al. [13] propose using the Earth Mover (EM) distance or Wasserstein-1 distance to solve the vanishing gradient problem. Intuitively, the EM distance can be seen as describing how much "effort" it costs to translate one distribution into another distribution. In Figure 3.3b, the same situation is illustrated as in Figure 3.3a, however, in this case the EM distance is applied. As can be seen, the EM distance always points into the direction of the best θ.

The Wasserstein GAN incorporates the EM distance. The WGAN training procedure is shown in algorithm 4. While algorithm 4 is similar to algorithm 3 there are some differences. For every training iteration, the discriminator is now updated n_{critic}



(A) JS divergence

(B) EM distance

FIGURE 3.3: In Figure *(A)* the JS divergence is shown for a model with one parameter θ. The samples in the model are uniformly distributed along one 'line'. If $\theta = 0.0$, we would have found the optimal value. However, the gradient for the JS divergence is mostly zero. In Figure *(B)* we have the same model but now the EM distance is applied. The gradient moves to the optimal value. Figure from Arjovsky et al. [13].

times, for every update θ_d is also clipped. For the Kantorovich-Rubinstein duality to hold the weights of the discriminator are clipped [13]. Meaning that, in this case, the EM distance can only be calculated when it is K-Lipschitz (See Appendix A.1).

Algorithm 4 Minibatch stochastic gradient descent training of WGAN with $n_{\text{critic}} = 5$ and $c = 0.01$

1: **procedure** TRAIN WGAN
2: **for** number of training iterations **do**
3: **for** $t = 0, \ldots, n_{\text{critic}}$ **do**
4: • Sample m noise samples $\{z^{(1)}, \ldots, z^{(m)}\}$ from $p_g(z)$
5: • Sample m samples $\{x^{(1)}, \ldots, x^{(m)}\}$ from $p_r(x)$
6: $\nabla_{\theta_d} \frac{1}{m} \sum_{i-1}^{m} [D(x^{(i)}) - D(G(z^{(i)}))]$ ▷ D update step
7: clip$(\theta_d, -c, c)$ ▷ Clip Weights
8: • Sample m noise samples $\{z^{(1)}, \ldots, z^{(m)}\}$ from $p_g(z)$
9: $\nabla_{\theta_g} \frac{1}{m} \sum_{i-1}^{m} D(G(z^{(i)}))$ ▷ G update step

WGANs promises stable training since the gradients "behave" much better since due to the EM distance. According to Arjovsky no mode collapse was encountered in their experiments which makes WGANs a serious candidate for solving mode collapse [13].

Another benefit of WGANs is that a meaningful loss metric can be calculated for training the discriminator (which is not the case for a traditional GAN). Since line 4 to 7 in algorithm 4 estimate the EM distance, we can plot the negative loss of the discriminator. Thus, we now have an indication of when the system has converged.

3.5 Generative Adverserial Imitation Learning

In the previous sections, policy gradient algorithms and GANs were introduced. In this section, we describe how GANs and TRPO can be combined to create an Imitation Learning (IL) algorithm which performs well on complex Machine Learning tasks.

3.5.1 Preliminaries

As mentioned previously, Inverse Reinforcement Learning (IRL) learns a cost function based on observed trajectories. A policy is then trained using this cost function. Ho & Ermon [8] describe the occupancy measure as the distribution of state-action pairs which are generated by the policy π. Mathematically the occupancy measure is defined as:

$$p_\pi(s,a) = \pi(a|s) \sum_{t=0}^{\infty} \gamma^t P(s_t = s|\pi) \tag{3.31}$$

Given this definition of the occupancy measure, IRL procedures can also be described as a procedure which finds a policy that matches the expert policy's occupancy measure.

A problem with IRL algorithms is that they are expensive to run because of the reinforcement step in the inner loop. Ho & Ermon propose a method which approximates the occupancy measure where the inner step is not necessary [8]. From an apprenticeship learning algorithm, which is a type of IRL algorithm, they are able to derive an Imitation Learning algorithm.

Apprenticeship Learning (AL) algorithms find a policy that performs better than the expert across a class of cost functions. The class of cost functions C encodes the expert policy π_E. If C does not include a cost function which describes the underlying behaviour of the expert policy, AL will be unable to learn that expert policy. Hence, if C is too restrictive, AL algorithms generally do not recover the expert policy.

The optimisation objective for entropy regularised apprenticeship learning can be seen in Equation 3.32.

$$\min_\pi -H(\pi) + \max_{c \in C} \mathbb{E}_\pi[c(s,a)] + \mathbb{E}_{\pi_E}[c(s,a)] \tag{3.32}$$

where C is a class of cost functions, $c(s,a)$ is a cost function with the state and action as an input, π_E is the expert policy, π is the policy which is trained and $H(\pi)$ is the γ-discounted causal entropy.

3.5.2 GAIL

Generative Adversarial Imitation Learning (GAIL) introduced by Ho & Ermon [8] is an IL algorithm which imitates arbitrarily complex expert behaviour and scales to large state and action spaces. From Equation 3.32 they derive an IL algorithm

which optimises the JS divergence D_{js} by defining a cost regulariser $\psi_{\text{GA}}(c)$. GAIL is defined in Equation 3.33.

$$\min_{\pi} \psi_{\text{GA}}(p_\pi - p_{\pi_E}) - \lambda H(\pi) = D_{\text{JS}}(p_\pi, p_{\pi_E}) - \lambda H(\pi) \tag{3.33}$$

where λ controles the policy regulariser $H(\pi)$.

Since GAIL optimises the JS divergence, a connection can be drawn between GAIL and GAN. The occupancy measure p_π is analogous to the data distribution of the generator G and the expert's occupancy measure is analogous to the true data distribution. The goal of GAIL is to find the saddle point of Equation 3.34 which is the same objective as Equation 3.30 excluding the policy regulariser.

$$\mathbb{E}_\pi[logD(x)] + \mathbb{E}_{\pi_e}[log(1 - D(a, s))] - \lambda H(\pi) \tag{3.34}$$

where π is the parameterised model which GAIL trains, π_E is the expert model, $D(a, s)$ is the discriminator network with weights θ_d and H is a policy regulariser where $\lambda > 0$.

GAIL is a two-step algorithm. First, the weights of the discriminator network D are updated by an ADAM [41] gradient step based on the generated trajectories. Secondly, a TRPO update step is performed on weights θ_g with respect to π. Ho & Ermon [8] have chosen TRPO because it limits the change for the policy per step. This is needed due to the noise in the policy gradient.

The training procedure for GAIL is described in algorithm 5. At line six the weights of the discriminator network are updated. Lines seven and eight update the policy weights θ_g using the TRPO update step. As explained in Section 3.2 this update step is limited by the Kullback-Leibler divergence.

Algorithm 5 GAIL training procedure

1: **procedure** TRAIN GAIL
2: • Initialise τ_E from π_E
3: • Initialise θ and w at random
4: **for** number of training iterations i **do**
5: • Generate Trajectory τ_i from π_E
6: $\mathbb{E}_{\tau_i}[\nabla_{\theta_d} log(D(s, a))] + \mathbb{E}_{\tau_E}[\nabla_{\theta_d} log(1 - D(s, a))]$ ▷ D update step
7: $\mathbb{E}[\nabla_{\theta_g} log\pi_{\theta_g}(a|s)Q(s, a)] - \lambda\nabla_{\theta_g} H(\pi_{\theta_g})$, ▷ TRPO update step
8: where $Q(s, a) = E_{\tau_i}[log(D(s, a))]$

GAIL has significant performance gains over IL methods and can imitate complex behaviour in large, high-dimensional environments. It can solve a wide range of complex learning tasks and outperforms Behavioural Cloning algorithms regarding data efficiency. Thus, GAIL is a good candidate for modelling human driving behaviour.

3.5.3 WGAIL

The author proposes an improved GAIL algorithm where the techniques used by WGAN methods are applied to GAIL which is properly named Wasserstein Generative Adversarial Imitation Learning (WGAIL). WGAIL aims to combine the advantages of WGAN methods with the advantages of the GAIL algorithm.

Since GAIL uses the same methods for modelling the discriminator, it also suffers from mode-collapse. As mentioned in Section 3.4.3, WGANs solve the problem of mode collapse. WGANs also results in more stability during training due to the EM distance being used.

The discriminator objective function for GAIL is switched for the WGAN discriminator objective function:

$$\mathbb{E}_{\tau_i}[\nabla_{\theta_d}(D(s,a))] - \mathbb{E}_{\tau_E}[\nabla_{\theta_d}(D(s,a))] \tag{3.35}$$

The difference with the original discriminator objective function (Equation 3.34) is that the log is not taken since the discriminator D does not output probabilities. The discriminator update function is changed from ADAM to RMSProp since RMSProp [42] is reported to improve training performance in the case of WGANs.

To satisfy the K-Lipschwitz property, the weights θ_d are clipped between $[-0.01, 0.01]$. Algorithm 6 shows the changes applied to the original GAIL algorithm.

Algorithm 6 WGAIL training procedure with $c = 0.01$

1: **procedure** TRAIN WGAIL
2: • Initialise τ_E from π_E
3: • Initialise θ and w at random
4: **for** number of training iterations i **do**
5: • Generate Trajectory τ_i from π_E
6: $\mathbb{E}_{\tau_i}[\nabla_{\theta_d}(D(s,a))] - \mathbb{E}_{\tau_E}[\nabla_{\theta_d}(D(s,a))]$ ▷ D update step
7: clip$(\theta_d, -c, c)$
8: $\mathbb{E}[\nabla_{\theta_g} log \pi_{\theta_g}(a|s) Q(s,a)] - \lambda \nabla_{\theta_g} H(\pi_{\theta_g})$, ▷ TRPO update step
9: where $Q(s,a) = E_{\tau_i}[(D(s,a))]$

At line 6 the new discriminator update step is displayed. After the discriminator step, the discriminator weights are clipped.

3.5.4 WGAIL-GP

While clipping the weights ensures the K-Lipschwitz property, Arjovsky et al. [13] and Gulrajani et al. [43] state that weight clipping might be problematic when used in a WGAN discriminator. Weight clipping introduces a bias for the discriminator resulting in much simpler modelled functions and pathological behaviour. An optimal discriminator under the WGAN loss function has unit gradient norm almost

everywhere [43]. When weight clipping is performed, the discriminator network can only obtain their maximum gradient norm when simple functions are learned. I.e. when a complex system is modelled, weight clipping models very simple approximations to the optimal functions.

Gulrajani et al. [43] also state that with, the original WGAN vanishing and exploding gradients are possible where the gradient either grows or shrinks exponentially in the later layers in the network.

These behaviours are undesirable. However, without weight clipping, the K-Lipschwitz property will not be enforced. Thus, Gulrajani et al. [43] propose a gradient penalty to enforce the Lipschitz constraint stating: *"A differentiable function is 1-Lipschtiz if and only if it has gradients with norm at most 1 everywhere"*.

The gradient penalty will constrain the discriminator gradient norm of the output with respect to the input. The gradient penalty is added to the loss of the discriminator resulting in the following function:

$$\mathbb{E}_{\tau_i}[\nabla_{\theta_d}(D(s,a))] - \mathbb{E}_{\tau_E}[\nabla_{\theta_d}(D(s,a))] + \lambda \mathbb{E}_{\hat{x}\sim P_{\hat{x}}}[(||\nabla_{\hat{x}} D(\hat{x})_2 - 1||)^2] \tag{3.36}$$

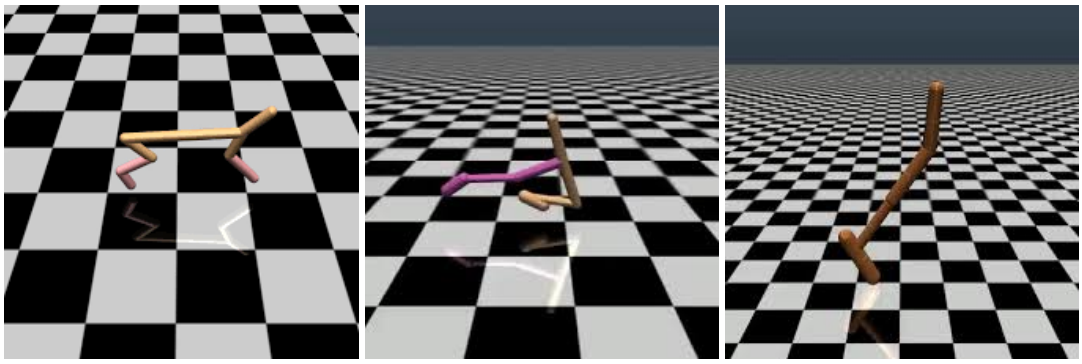
where $\hat{x}$ is sampled randomly from a uniform distribution $p_{\hat{x}}$ and λ is the hyperparameter for setting the *impact* of the penalty. For all experiments $\lambda = 10$ is chosen as it is reported to result in a good discriminator performance [43].

Chapter 4

Experiments

In this section, we present two different sets of experiments. First, the WGAIL and WGAIL-GP algorithms are compared to the GAIL algorithms by learning three environments part from the OpenAI Gym program. Second, WGAIL-GP and GAIL are used to model a driving trajectory extracted from the NGSIM dataset. In this section, every experiment will be discussed individually within its scope.

4.1 OpenAI Gym experiments



(A) (B) (C)

FIGURE 4.1: The OpenAI Gym environments are displayed. *(A) HalfCheetah-v1* environment where half of a 2d cheetah is simulated. *(B) Walker2d-v1* environment where a 2d bipod is simulated. *(C) Hopper-v1* environment which simulates a robot leg which hops.

The performance of the WGAIL algorithms (Section 3.5.3) were evaluated by comparing it to the original GAIL algorithm (algorithm 5). Multiple experiments have been run on three different Gym environments, *Hopper-v1*, *HalfCheetah-v1* and *Walker2D-v1*. These environments have been chosen because they are inherently more difficult than the classic control problems like the Cartpole and Mountain car problem [14]. Each environment has a different objective to solve. Every environment produces a reward which indicates the performance of the learning algorithm, i.e. the trained policy is evaluated by the reward produced by the environment and will be an indication about how well the algorithm has learned to "navigate" the environment.

For every environment eighteen trajectories were generated from a previously trained expert. These three trajectory sets will function as the expert data.

For the original GAIL algorithm, the parameters, as well as the network structure as described by Ho & Ermon [8] are used. These parameters are reported to work well for the original GAIL algorithm. For the TRPO, discriminator and value estimator networks standardisation was applied to the input of the networks. The policy was trained on a fully connected network of three layers (two layers of 100 nodes and an output layer). The first two layers are activated by a *tanh* activation function. The last layer is a linear layer where in the case of continuous action values a combination of output values is used to form a Gaussian function from which an action value is sampled. If the environment uses discrete values, a softmax output is used to calculate the action values.

The results were scaled from zero to one where zero is the performance of a random actor and one is the value at which the environment is considered to be solved. Some environments are considered unsolved, in this case, a value of one will represent the maximum reward that was achieved in the training process. The scaling is done so that a fair comparison between the different environments is possible. Multiple experiments were run using the same parameters. The results from the experiments are plotted using error bands representing the standard deviation and a line representing the mean results. At least three trials were performed for every experiment from which the mean and the standard deviation were calculated.

The Gym experiments were conducted on a 4th generation I3 intel CPU and an Nvidia GTX 1070 graphics processor. Every experiment took approximately six hours.

4.1.1 HalfCheetah-v1

The *HalfCheetah-v1* environment (Figure 4.1a) is an environment where half of a running cheetah is simulated. The goal is to travel the furthest distance within 1000 steps. The state and action space have a dimension of 17 and six respectively. The *HalfCheetah* environment is considered solved when the reward after 1000 steps is higher than 4800 (scaled value 1.0).

Multiple experiments were performed using the *HalfCheetah* environment. A learning rate of 0.01 for the GAIL algorithm and WGAIL-GP algorithm was set using the ADAM optimiser and a learning rate of 0.01 was set for the WGAIL algorithm using the RMSprop optimiser.

The results of the *HalfCheetah* experiment are seen in Figure 4.2. The GAIL algorithm can solve the environment in less than 1000 iterations. While the WGAIL-GP algorithm is also able to solve the environment, it did result in less stable training as indicated by the larger error bars and higher volatility in the mean return. The WGAIL algorithm slowly converges to a solution, however, was unable to solve the environment within 2000 steps. This is probably due to different hyperparameters being used and could be a result of weight clipping.

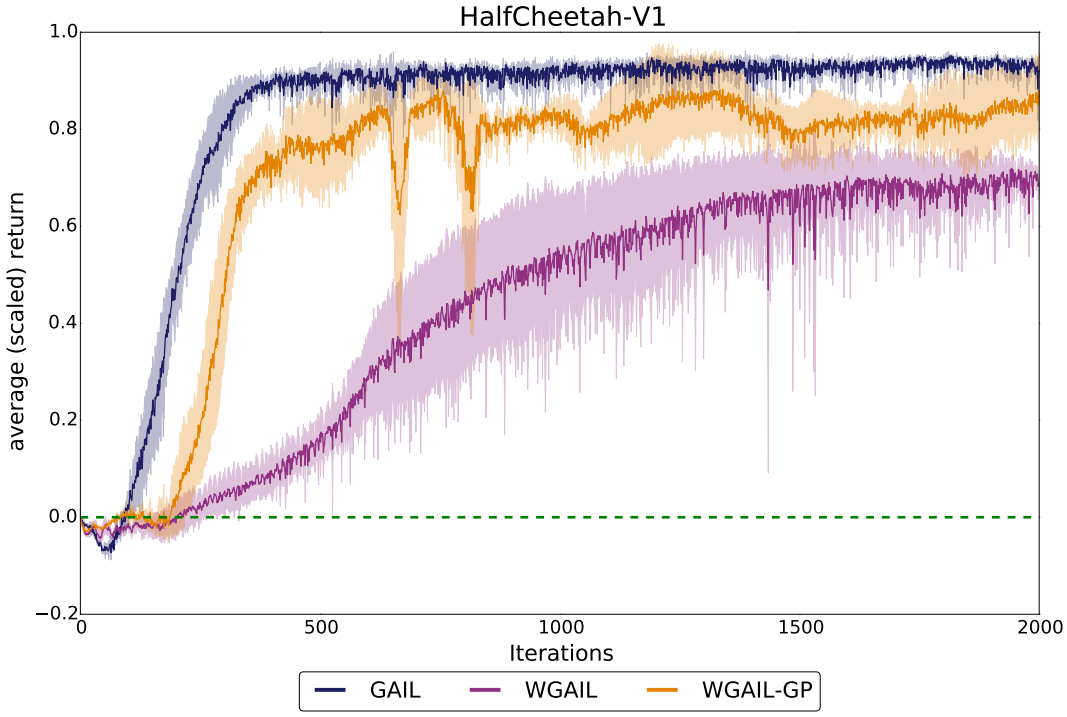


FIGURE 4.2: Results for the *HalfCheetah-v1* environment trained using the GAIL (blue), WGAIL (purple) and WGAIL-GP (oange). The dotted green line is the performance of a random policy.

4.1.2 Walker2d-v1

In the *Walker2d* environment (Figure 4.1b) a bipedal robot is simulated. The goal for the robot is to travel the longest distance over 1000 steps. The *Walker2d* environment is generally considered more difficult to learn than the *HalfCheetah* environment since the actor not only needs to learn to run but also learn to balance the bipedal robot. It has the same state and action space dimension as the *HalfCheetah* environment, 17 and six respectively. The *HalfCheetah* environment is considered an unsolved environment and has no reward specified for which it is considered solved. Thus, the scaled return 1.0 value refers to the highest reward achieved while training.

Walker2d is trained on GAIL, WGAIL and WGAIL-GP. The latter has been trained with both ADAM and RMSprop to show the difference in training performance. A learning rate of 0.095 is used for WGAIL and WGAIL-GP. For GAIL standardisation and a learning rate of 0.01 are set.

The WGAIL and WGAIL-GP RMSprop performance are similar in terms of speed of convergence. The WGAIL-GP RMSprop has a lower average return in comparison to the GAIL algorithm. This is probably due to the parameter setup since both algorithms use the same network in both layer sizes and activation types. The WGAIL algorithm performs lower, not surpassing 0.5 on the average scaled return. While it does not perform worse than GAIL and WGAIL-GP RMSprop for the first 100 iterations, it seems to get stuck in a local minimum.

For the first 50 iterations, the WGAIL-GP ADAM experiment performs the same as the other experiments. However, after the first 50 iterations, it reaches a local

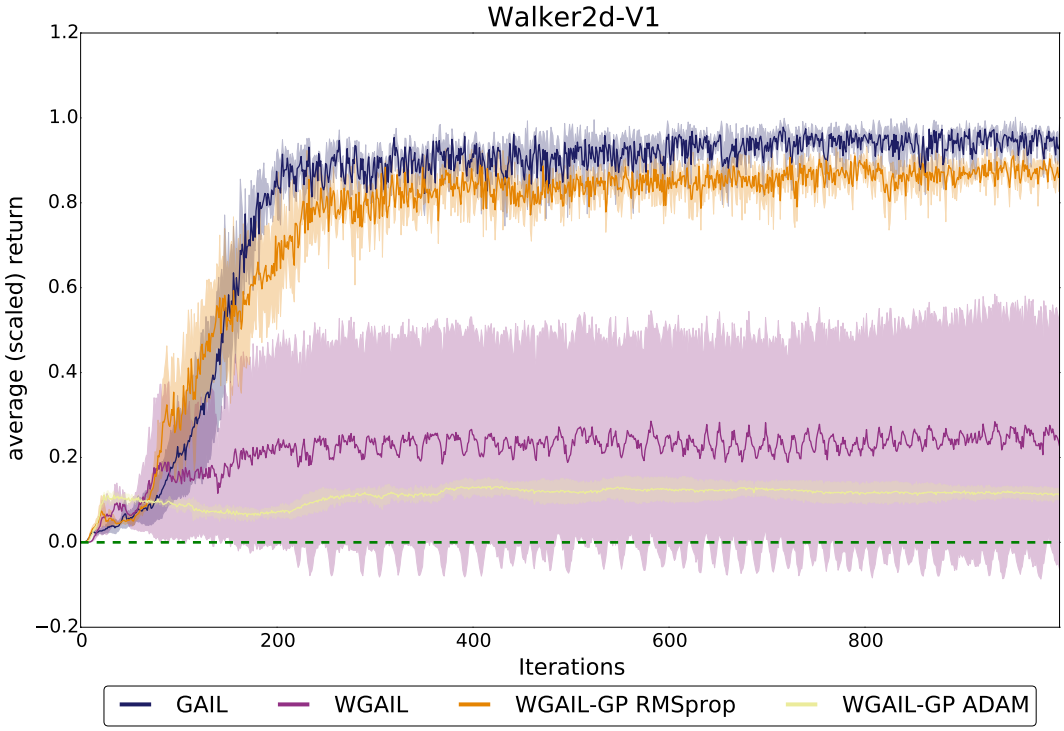


FIGURE 4.3: Results for the *Walker2d-v1* environment trained on the GAIL, WGAIL and WGAIL-GP algorithms using the ADAM optimiser and the RMSprop optimiser. The dotted green line is the performance of a random policy.

minimum from which it is unable to progress any further. The algorithm can learn to stand up and jump, but not able to move forward. This problem is inherent to the Walker environment and the reward function it generates. It can be solved by implementing a penalty for "jumpy" behaviour [10]. Since the GAIL and WGAIL-GP RMSprop algorithm are able to solve the environment without the penalty it should in theory also be possible for the WGAIL-GP ADAM algorithm to solve the environment without the jump penalty. We were, however, unable to find the correct set of parameters to achieve this.

4.1.3 Hopper-v1

The *Hopper-v1* (Figure 4.1c) environment is a 2d environment where a robot leg is simulated. The actor must learn to hop as fast as possible over a certain number of steps. The Hopper environment is considered relatively simple, with an action space of three and a state space of 11. The environment is solved when a reward of 3800 (scaled return 1.0) has been achieved by the actor. WGAIL and WGAIL-GP were trained with a learning rate of 0.095, while GAIL was trained with a learning rate of 0.01.

In Figure 4.4 the results are shown for the Hopper experiments. While GAIL stabilises after 200 iterations, WGAIL-GP on average takes longer. WGAIL and WGAIL-GP both have larger error bars than GAIL indicating that the training is less stable. While WGAIL-GP at some points performs better then GAIL, it converges to just under the original GAIL algorithm. It is the author's opinion that with tweaking the

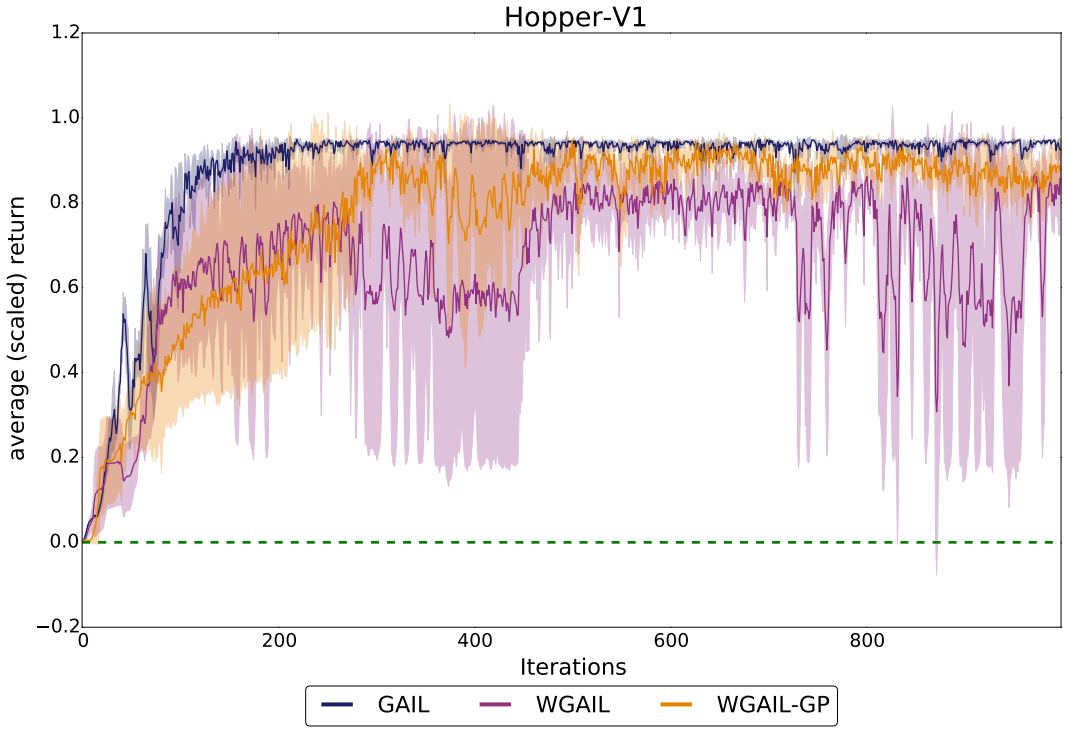


FIGURE 4.4: Results for the *Hopper-v1* environment trained on the GAIL, WGAIL and WGAIL-GP algorithms using the ADAM optimiser and the RMSprop optimiser. The dotted green line is the performance of a random policy.

hyperparameters the same results can be achieved for WGAIL-PG in comparison with GAIL.

The WGAIL algorithm performs less stable than WGAIL-GP. While it does not converge after 1000 steps, it is able to perform better than random. Changing the parameters could improve performance. However, we were unable to find values which performed better due to the unstable training.

4.2 NGSIM experiments

The NGSIM dataset introduced in Section 2.4.2 is used for performing experiments which learn human driving behaviour. The NGSIM data is loaded in a highway traffic simulator developed by fortiss GmbH which can use different feature sets and sensor models.

The goal of the NGSIM experiments is to show the performance between the GAIL and the newly introduced WGAIL algorithms in order to determine whether WGAIL is a viable option for learning human driving behaviour. The experiments from *Imitating driver Behavior with Generative Adversarial Networks* by Kuefler et al. [11] are replicated using both the GAIL and WGAIL-GP algorithms. However, in this case, we only train on a single trajectory using the same parameters and features used by Kuefler et al.

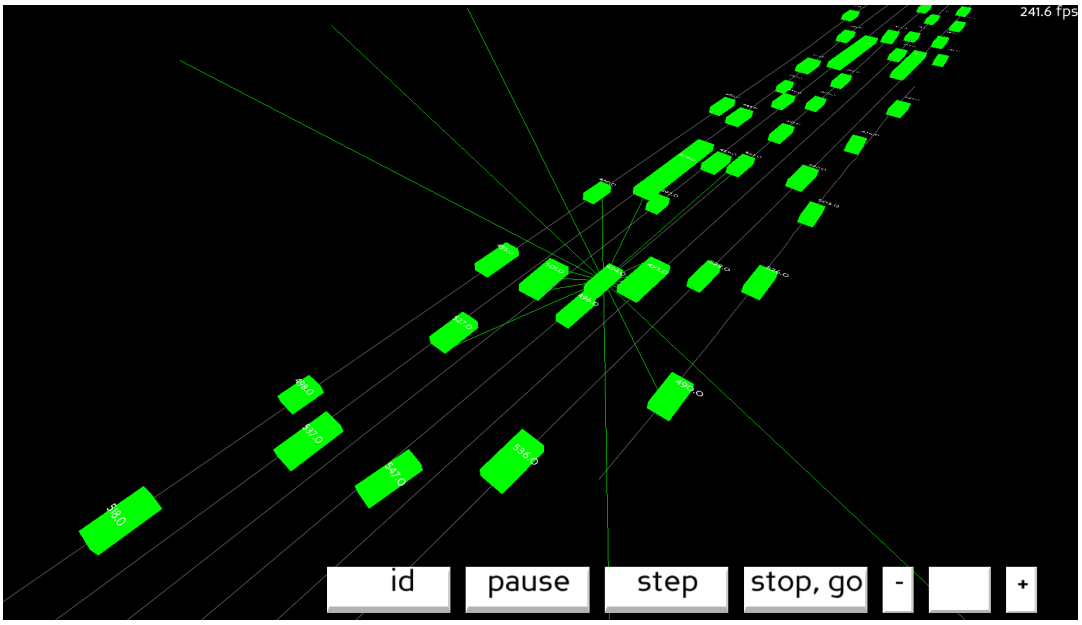


FIGURE 4.5: Traffic simulator developed by fortiss GmbH with lidar beam simulation.

4.2.1 Features

From the NGSIM data, three datasets are extracted. A *core* feature set containing eight scalar values which describe positional information as well as information about the vehicle itself. The vehicle is represented as a rectangular bounding box. The core features can be found in the first eight entries in Table 4.1.

Twenty lidar beams are simulated by calculating the intersection of the beam and the surrounding vehicles. Within the feature set, the lidar beams are stored as the distance from the ego vehicle until the intersection point and the change rate between every step. If no intersection occurs, the total length of the beam is stored. Thus, 40 scalar values are generated for every step. The max length of the beams is set to 100 meters. The 20 beams form a partial 360 degree view. While in an actual lidar-radar the beams would be rotating, we follow the simulation used by Kuefler et al. [11] to make a fair comparison.

Three more values are added to the feature set which indicates whether the car is in a collision, offroad or travels in reverse. These are states that we want to avoid and are represented as a boolean value or a one and zero value in the feature vector. When a state is encountered with one of these values marked as true, the traffic simulator considers the current trajectory as finished. In Table 4.1 the full feature vector is described.

TABLE 4.1: The feature vector used for training on the NGSIM data

#	Feature	Unit	Description
1	Speed	m/s	Speed of vehicle
2	Length Vehicle	m	Box vehicle length
3	Width Vehicle	m	Box vehicle width
4	Lane Offset	m	Distance to the center of the lane
5	Relative Heading	rad	Angle of car in comparisson with lane
6	Lane angle	rad	Angle between two lane waypoints
7	Lane offset left	m	Distance to the left lane
8	Lane offset right	m	Distance to the right lane
9-48	Lidar Beams	m & m/s	Lidar beam 2 scalar values length and change rate respectively.
49	Collision	Boolean	Vehicle in collision
50	Offroad Beams	Boolean	Vehicle is offroad
51	Reverse Beams	Boolean	Vehicle travelling in reverse

Kuefler et al. [11] use the same feature set as described in table 4.1 and is known to work for training GAIL on the NGSIM data. Kuefler et al. state that they do not include the previous action taken in the set of features since policies can develop an over-reliance on previous actions and ignore other features.

4.2.2 Results

The training progress is plotted by calculating the distance between the ego vehicle in the expert trajectory (NGSIM data) and the generated trajectory (trained policy). While this is a simplified statistic, it does capture how well the algorithms can learn a single trajectory and should be sufficient for comparing the different algorithms.

For the comparison between GAIL and WGAIL-GP, a single trajectory was chosen of 100 steps. The trajectory contains a simple overtaking manoeuvre. A network of three layers is used decreasing in size with respectively 256, 128 and 32 nodes containing a *tanh* activation function for both the GAIL and WGAIL-GP algorithms. The ADAM optimiser is used by the GAIL algorithm with a learning rate of 0.0095. The RMSprop optimiser is used by the WGAIL-PG algorithm with a learning rate of also 0.0095. In Figure 4.6 the results are given for the single trajectory experiment.

Both for GAIL and WGAIL the return shows large spikes while training. This can be explained by the nature of the problem and the fact that the error shown is a derived value. When learning a trajectory, small changes in the ego vehicle behaviour can lead to significant error spikes. For example, a small change in steering might result in a collision and the trajectory being cut off. It could also be that the large spikes are a result of the complicated feature set. While the training progress is volatile, both algorithms converge at approximately the same value.

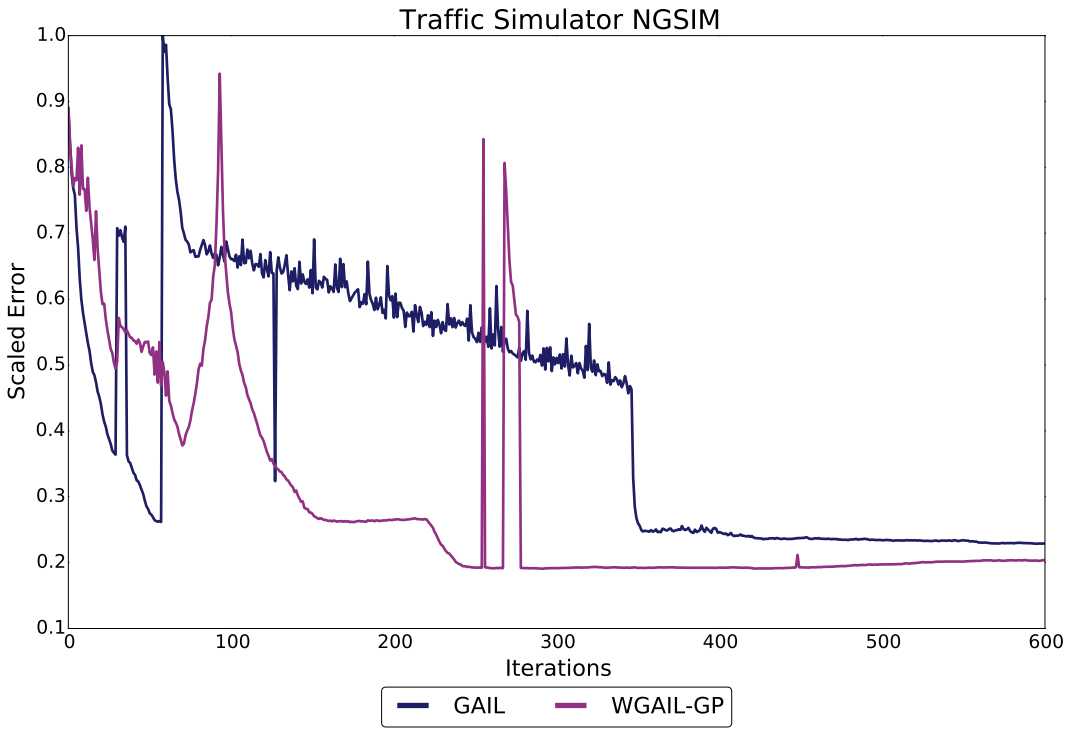


FIGURE 4.6: Error between the expert trajectory and the trained trajectory. The error is plotted for the GAIL and WGAIL-GP algorithms using the ADAM optimiser and the RMSprop optimiser.

Chapter 5

Discussion

In this chapter, the results presented in Chapter 4 will be discussed in depth and suggestions are given for further research.

5.1 Gym experiments

As demonstrated in Section 4.1 GAIL and WGAIL-GP perform quite even. The GAIL performance is also similar to the results shown by Ho et al. [8]. However, it is clear that the performance of WGAIL is not optimal. Since WGAIL was tested with the ADAM and RMSprop optimisers resulting in mixed results, we conclude that while it does affect training somewhat, it is not the main reason why WGAIL was underperforming.

One of the problems with GAIL is the high hyperparameter sensitivity, meaning that a small change in parameter setup often leads to unstable training. This problem was also very noticeable for the WGAIL and WGAIL-GP algorithms and might explain why the WGAIL algorithm was performing worse than the GAIL and WGAIL-GP algorithm due to simply not finding the right parameters.

While training the *Ant-v1* environment WGAIL and WGAIL-GP would not converge within 2000 iterations with results comparable to GAIL. The experiments for the *Ant-v1* environment where executed over a range of hyperparameters. While it is the author's opinion that WGAIL-GP is able to solve the *Ant-v1* environment, it is an indication that the hyperparameter sensitivity problem influences learning and optimisations could be done in stabilising training.

To stabilise training, multiple experiments where done adding new methods to stabilise training. Henderson et al. [44] show that using ReLU or Leaky ReLU activations within the network usually improve the performance of the TRPO algorithm over *tanh* activations when trained on OpenAI Gym environments. Both ReLU and Leaky ReLU activations did however not result in increased stability for training on the *Walker2d-v1* environment using the WGAN or WGAN-PG algorithms. Further research should be done on whether this is the case for every environment.

Kuefler et al. [11] apply standardisation to environment states where for every state the running mean is subtracted and divided by the standard deviation. While this has a positive effect on the TRPO algorithm (the generator), it did not influence the performance of our experiments for the GAIL discriminator. In fact, for the WGAIL and WGAIL-GP discriminator, the performance worsened when standardisation was applied.

The input for Wasserstein GANs is often scaled between zero and one [13], [43]. We applied feature scaling to the input states where the input was scaled between -1 and 1, and 0 and 1 for the *Walker2d-v1* and *HalfCheetah-v1* environments. On both environments, the performance of WGAIL and WGAIL-GP worsened with feature scaling. Overall, the best performance for WGAIL and WGAIL-GP was achieved without any modifications to the input states.

When weights are initialised either too small or too large, the resulting activations could negatively influence learning. Glorot initialisation has been known to fix this problem and improve the speed of convergence [45]. Random Uniform, as well as Glorot initialisation, was used for the WGAIL and WGAIL-GP experiments. Using Glorot initialisation over Random Uniform initialisation did not result in a significant difference in learning performance.

The *Walker2d-v1* and *Hopper-v1* environment stop when the generated state is undesirable and the trajectory is cut short, for example, when the walker or hopper loses their balance. If we take a trajectory of 100 steps where the last three steps are undesirable, the number of undesirable states is relatively small in comparison to the rest of the states in the trajectory. The environment can be forced to complete the trajectory resulting in more undesirable states in the trajectory. When training the discriminator would encounter more undesirable states in the generated trajectories and would thus be able to better classify the trajectory as fake instead of real. Experiments were done using this modification for the *Walker2d-v1* environment and while the training did stabilise it would get stuck in local minima.

For WGAIL the weight clipping parameters also need to be set which further increases the hyperparameter sensitivity problem. However, the action of weight clipping itself could also influence the learning performance negatively as shown by Gulrajani et al. [43]. This could be the case since the parameters excluding the learning rate for the discriminator are the same for WGAIL and WGAIL-GP where the results produced by WGAIL are consistently lower.

5.2 NGSIM experiments

While the experiments done with the traffic simulator are limited, the results do indicate that WGAIL-GP performs just as well or better as GAIL. While both algorithms show high volatility for the learning progress, they do converge approximately around the same value. From this, we can conclude that WGAIL-GP is a viable option for learning driving behaviour (as defined by Kuefler et al. [11]) compared to GAIL.

While both GAIL and WGAIL converge, it is clear that there is room for improvement. It would be interesting to see how WGAIL would perform if it was trained on multiple trajectories and if the results produced by Kuefler et al. [11] will be similar.

A surrogate reward based on prior knowledge of the environment could also improve the learning performance. This could be done by creating an incentive for the agent when learning while having no impact on the Imitation Learning. In the case of WGAIL, a surrogate value would be added to the discriminator output which is "fed" into the generator. Intuitively, the resulting algorithm is a mixture of Imitation and Reinforcement learning.

As stated by Kuefler et al. [11] their trained agent would show small oscillations in the steering angle while the expert data does not contain such behaviour. A surrogate reward could help here, penalising small oscillations in the steering angle happen. Due to time constraints we where unable to finish these experiments, however, this would be an interesting topic for further research.

5.3 WGAIL performance

While GAIL performed slightly better for the Gym experiments, we think that with tweaking WGAIL-GP could perform just as well. However, the WGAIL and WGAIL-GP algorithms have shown to be sensitive to small changes in hyperparameter setup. In the case of the Gym experiments, we were unable to find the correct parameters settings for the Open AI Gym *Ant-v1* environment where the results were comparable to the GAIL environment. Thus, further research should be done in making the algorithms perform better on a wider range of hyperparameters.

Whilst training we noticed that inherently GAIL seems to prefer longer trajectories. In an environment where a very short trajectory leads to the highest reward or a longer trajectory leading to a lower reward, the resulting actor trained with GAIL would often only learn to take the longer trajectory. This was both observed in our implementation as well as the official implementation provided by Ho & Ermon. According to Ho [46], this could be due to the lack of expert states where the higher reward is occurring. Interestingly this could be the case for many learning problems. For example, if a car in a trajectory would perform a short evasive manoeuvre, this manoeuvre would be ignored by the learning algorithm. It is unclear how this affects overall learning and how this might be fixed for both GAIL and the WGAIL alternatives.

From the results presented in Section 4.1 it is clear that WGAIL-GP outperformed WGAIL in terms of learning performance. Since WGAIL and WGAIL-GP both introduce an extra hyperparameter, a parameter for weight clipping and a parameter for setting the gradient penalty weight respectively, this further impacts the hyperparameter problem. We found that determining the weight clipping parameter was quite difficult and changes often resulted in increased learning performance. Learning performance was less influenced by changes in the penalty gradient weight parameter.

Arjovsky et al. [13] state that momentum based optimisers result in unstable training due to the loss function being non-stationary. This includes the ADAM optimiser and thus the RMSprop optimiser should be used. This claim, however, was disputed by Gulrajani et al. [43] stating that in their case the ADAM optimiser performed better than the RMSprop optimiser. The use of the RMSprop and ADAM optimiser resulted in a mixed performance for the Gym experiments. While for the *Walker2d-v1* environment the RMSprop optimiser performed better than the ADAM optimiser. For WGAIL-GP, the ADAM optimiser had the best overall performance. The author notes that while training, the RMSprop progress was more stable in comparison to the ADAM optimiser but would often get stuck in a local minimum.

The speed of convergence for GAIL, WGAIL and WGAIL-GP was approximately equal. While for the NGSIM experiments the WGAIL-GP convergence was faster in comparison with GAIL, it was slower for the Gym experiments. As noted by

Ho & Ermon [8], the learning speed could significantly be increased by initialising the policy parameters using Behavioural Cloning. This could also be applied to WGAIL and WGAIL-GP and should improve the learning speed. Regarding runtime, both GAIL and WGAIL algorithms performed equally well. It should be noted that WGAIL-GP performed slightly slower due to the custom gradient penalty function. The longer run time, however, was negligible.

The experiments shown in Section 4 show the performance of WGAIL and WGAIL-GP, however, do not show whether WGAIL or WGAIL-GP actually solves mode collapse. While this was already shown by Arjovsky et al. [13] for generating images, further research should be done in showing that mode collapse will not happen for WGAIL and WGAIL-GP.

Replay buffers are known to improve training performance for actor-critic methods [47]. Replay buffers can be applied to GAIL and the WGAIL algorithms by storing the generated trajectories over multiple iterations and train the discriminator on trajectories randomly picked from the replay buffer. Picking trajectories from the replay buffer ensures that the discriminator does not fit too closely to the current generator [48]. Due to time constraints, we were unable to implement a replay buffer and perform experiments.

Another optimisation for stabilising WGAIL and WGAIL-GP training is to add noise to the generated and expert data [49]. The motivation for adding noise to the data is that the generated data's distribution does not overlap with the expert data which could result in overfitting in the discriminator. If noise is added from the same distribution to the expert and generated data, the distributions should overlap.

Chapter 6

Conclusions

In this thesis, we introduce a novel algorithm which combines techniques used by Wasserstein Generative Adversarial Networks (WGAN) with the Generative Adversarial Imitation learning algorithm (GAIL). This novel algorithm called Wasserstein Generative Adversarial Imitation learning (WGAIL) and a variant of WGAIL which is trained using a gradient penalty function dubbed WGAIL-GP should solve the problem of mode collapse and vanishing gradients from which Generative Adversarial Networks and in turn, GAIL suffers. The contributions of this thesis have been to (1) combine GAIL with a Wasserstein GAN, (2) show that WGAIL-PG is a viable method for solving canonical control problems, (3) show that WGAIL-PG performs just as well as GAIL on the complex problem of modelling driving trajectories.

Although WGAIL resulted in unstable training and underperformed in comparison to the GAIL algorithm, WGAIL-PG was able to achieve nearly the same results as GAIL. These results were consistent for both the OpenAI Gym environments and the driving trajectory modelling. Both WGAIL and WGAIL-GP did suffer from high sensitivity to hyperparameters resulting in unstable training. While multiple techniques were applied aimed to fix this problem we have yet to find a method which reduces the high sensitivity and stabilises training.

To stabilise training, a replay buffer could be implemented from which the input for the discriminator is randomly drawn. This will result in the discriminator not fitting too closely to the generator and could result in increased learning stability. Noise could also be added to the generated and expert data to make their distributions overlap. This could increase the performance of the discriminator. The driving trajectory experiments could be extended by adding a surrogate reward function to the output of the discriminator in order to penalise bad behaviour for the generated policy.

It should be noted that the work in this thesis is not complete. While the results presented in this thesis give an indication that WGAIL-PG can solve complex control problems, the driving trajectory experiments are limited and only show results where WGAIL-PG is trained on a single trajectory. It is the author's opinion that these results lack the expressiveness needed to form a well-built conclusion about the performance of the WGAIL-PG algorithm for modelling driving behaviour.

Appendix A

Methods Appendix

A.1 Lipschitz

A function f is L-Lipschitz when $L \geq 0$ given equation A.1.

$$|f(a) - f(b)| \leq L|a - b| \tag{A.1}$$

where a and b any pair of points within the domain. Intuitively a L-Lipschitz constraint limits the slope of every pair of points, i.e the constraint limits how fast the function can change.

Bibliography

[1] P. Gipps, "A behavioural car-following model for computer simulation", *Transportation Research Part B: Methodological*, vol. 15, no. 2, pp. 105–111, Apr. 1981, ISSN: 01912615. DOI: 10.1016/0191-2615(81)90037-0. [Online]. Available: http://linkinghub.elsevier.com/retrieve/pii/0191261581900370.

[2] D. C. Gazis, R. Herman, and R. W. Rothery, "Nonlinear Follow-The-Leader Models of Traffic Flow", *Source: Operations Research*, vol. 9, no. 4, pp. 545–567, 1960. [Online]. Available: http://www.jstor.orghttp://www.jstor.org.

[3] M. Treiber, A. Hennecke, and D. Helbing, "Congested Traffic States in Empirical Observations and Microscopic Simulations", *Physical Review E*, vol. 62:1805-1824, Feb. 2000. DOI: 10.1103/PhysRevE.62.1805. [Online]. Available: http://arxiv.org/abs/cond-mat/0002177http://arxiv.org/abs/cond-mat/0002177.

[4] M. Bojarski, 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, "End to End Learning for Self-Driving Cars", *arXiv preprint*, Apr. 2016. [Online]. Available: http://arxiv.org/abs/1604.07316.

[5] D. A. Pomerleau, "ALVINN: An autonomous land vehicle in a neural network", *NIPS'88*, vol. 1, pp. 305–313, 1988. [Online]. Available: https://papers.nips.cc/paper/95-alvinn-an-autonomous-land-vehicle-in-a-neural-network.pdf.

[6] S. Ross and J. A. Bagnell, "Efficient Reductions for Imitation Learning", *JMLR W&CP*, vol. 9, 2010. [Online]. Available: http://ri.cmu.edu/pub_files/2010/5/Ross-AIStats10-paper.pdf.

[7] P. Abbeel and A. Y. Ng, "Apprenticeship Learning via Inverse Reinforcement Learning", *ICML '04 Proceedings of the twenty-first international conference on Machine learning*, 2004.

[8] J. Ho and S. Ermon, "Generative Adversarial Imitation Learning", *Advances in Neural Information Processing Systems*, vol. 29, Jun. 2016. [Online]. Available: http://arxiv.org/abs/1606.03476.

[9] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative Adversarial Networks", *eprint arXiv*, Jun. 2014. [Online]. Available: http://arxiv.org/abs/1406.2661.

[10] J. Schulman, S. Levine, P. Moritz, M. I. Jordan, and P. Abbeel, "Trust Region Policy Optimization", *ICML*, pp. 1889–1897, Feb. 2015. [Online]. Available: http://arxiv.org/abs/1502.05477.

[11] A. Kuefler, J. Morton, T. Wheeler, and M. Kochenderfer, "Imitating Driver Behavior with Generative Adversarial Networks", *arXiv preprint*, Jan. 2017. [Online]. Available: http://arxiv.org/abs/1701.06699.

[12] I. Goodfellow, "NIPS 2016 Tutorial: Generative Adversarial Networks", *arXiv preprint*, Dec. 2016. [Online]. Available: http://arxiv.org/abs/1701.00160.

[13] M. Arjovsky, S. Chintala, and L. Bottou, "Wasserstein GAN", *arXiv preprint*, Jan. 2017. [Online]. Available: http://arxiv.org/abs/1701.07875.

[14] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "OpenAI Gym", Jun. 2016. [Online]. Available: http://arxiv.org/abs/1606.01540.

[15] Federal Highway Administration (FHWA), *Traffic Analysis Tools: Next Generation Simulation - FHWA Operations*. [Online]. Available: https://ops.fhwa.dot.gov/trafficanalysistools/ngsim.htm.

[16] D. Lenz, F. Diehl, M. T. Le, and A. Knoll, "Deep neural networks for Markovian interactive scene prediction in highway scenarios", in *2017 IEEE Intelligent Vehicles Symposium (IV)*, IEEE, Jun. 2017, pp. 685–692, ISBN: 978-1-5090-4804-5. DOI: 10.1109/IVS.2017.7995797. [Online]. Available: http://ieeexplore.ieee.org/document/7995797/.

[17] A. Kesting, M. Treiber, and D. Helbing, "General Lane-Changing Model MOBIL for Car-Following Models", *Transportation Research Record: Journal of the Transportation Research Board*, vol. 1999, pp. 86–94, Jan. 2007, ISSN: 0361-1981. DOI: 10.3141/1999-10. [Online]. Available: http://trrjournalonline.trb.org/doi/10.3141/1999-10.

[18] P. Kumar, M. Perrollaz, S. Lefevre, and C. Laugier, "Learning-based approach for online lane change intention prediction", in *2013 IEEE Intelligent Vehicles Symposium (IV)*, IEEE, Jun. 2013, pp. 797–802, ISBN: 978-1-4673-2755-8. DOI: 10.1109/IVS.2013.6629564. [Online]. Available: http://ieeexplore.ieee.org/document/6629564/.

[19] H. Woo, Y. Ji, H. Kono, Y. Tamura, Y. Kuroda, T. Sugano, Y. Yamamoto, A. Yamashita, and H. Asama, "Lane-Change Detection Based on Vehicle-Trajectory Prediction", *IEEE Robotics and Automation Letters*, vol. 2, no. 2, pp. 1109–1116, Apr. 2017, ISSN: 2377-3766. DOI: 10.1109/LRA.2017.2660543. [Online]. Available: http://ieeexplore.ieee.org/document/7835731/.

[20] T. KUMAGAI and M. Akamatsu, "Prediction of Human Driving Behavior Using Dynamic Bayesian Networks", *IEICE Transactions on Information and Systems*, vol. E89-D, no. 2, pp. 857–860, Feb. 2006, ISSN: 0916-8532. DOI: 10.1093/ietisy/e89-d.2.857. [Online]. Available: http://search.ieice.org/bin/summary.php?id=e89-d_2_857&category=D&year=2006&lang=E&abst=.

[21] T. A. Wheeler, P. Robbel, and M. J. Kochenderfer, "A Probabilistic Framework for Microscopic Traffic Propagation", in *2015 IEEE 18th International Conference on Intelligent Transportation Systems*, IEEE, Sep. 2015, pp. 262–267, ISBN: 978-1-4673-6596-3. DOI: 10.1109/ITSC.2015.52. [Online]. Available: http://ieeexplore.ieee.org/document/7313144/.

[22] D. Silver, J. A. Bagnell, and A. Stentz, "High Performance Outdoor Navigation from Overhead Data using Imitation Learning", *Robotics: Science and Systems IV*, 2008. DOI: 10.15607/RSS.2008.IV.034. [Online]. Available: https://www.ri.cmu.edu/pub_files/pub4/silver_david_2008_1/silver_david_2008_1.pdf.

[23] U. Syed and R. E. Schapire, "A Game-Theoretic Approach to Apprenticeship Learning", *NIPS'07*, pp. 1449–1456, 2007. [Online]. Available: https://papers.nips.cc/paper/3293-a-game-theoretic-approach-to-apprenticeship-learning.pdf.

[24] D. S. Gonzalez, J. S. Dibangoye, and C. Laugier, "High-speed highway scene prediction based on driver models learned from demonstrations", in *2016 IEEE 19th International Conference on Intelligent Transportation Systems (ITSC)*, IEEE, Nov. 2016, pp. 149–155, ISBN: 978-1-5090-1889-5. DOI: 10.1109/ITSC.2016.7795546. [Online]. Available: http://ieeexplore.ieee.org/document/7795546/.

[25] J. Ho, J. K. Gupta, and S. Ermon, "Model-Free Imitation Learning with Policy Optimization", *JMLR W&CP*, vol. 48, 2016. [Online]. Available: https://arxiv.org/pdf/1605.08478.pdf.

[26] C. Ledig, L. Theis, F. Huszar, J. Caballero, A. Cunningham, A. Acosta, A. Aitken, A. Tejani, J. Totz, Z. Wang, and W. Shi, "Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network", Sep. 2016. [Online]. Available: http://arxiv.org/abs/1609.04802.

[27] V. Dumoulin, I. Belghazi, B. Poole, O. Mastropietro, A. Lamb, M. Arjovsky, and A. Courville, "Adversarially Learned Inference", Jun. 2016. [Online]. Available: http://arxiv.org/abs/1606.00704.

[28] L. Mescheder, S. Nowozin, and A. Geiger, "The Numerics of GANs", May 2017. [Online]. Available: http://arxiv.org/abs/1705.10461.

[29] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel, "High-Dimensional Continuous Control Using Generalized Advantage Estimation", *arXiv preprint*, Jun. 2015. [Online]. Available: http://arxiv.org/abs/1506.02438.

[30] Federal Highway Administration (FHWA), *Interstate 80 Freeway Dataset*, *FHWA-HRT-06-137*. [Online]. Available: https://www.fhwa.dot.gov/publications/research/operations/06137/.

[31] M. Treiber and A. Kesting, "How Much does Traffic Congestion Increase Fuel Consumption and Emissions? Applying a Fuel Consumption Model to the NGSIM Trajectory Data", Jan. 2018.

[32] X.-Y. Lu and A. Skabardonis, "Freeway Traffic Shockwave Analysis: Exploring the NGSIM Trajectory Data", Jan. 2018.

[33] C. Thiemann, M. Treiber, and A. Kesting, "Estimating Acceleration and Lane-Changing Dynamics Based on NGSIM Trajectory Data", 2008. [Online]. Available: http://www.traffic-simulation.dehttp://www.akesting.de.

[34] *Providentia - fortiss*. [Online]. Available: https://www.fortiss.org/en/research/projects/providentia-1/.

[35] E. Greensmith, P. L. Bartlett, and J. Baxter, "Variance Reduction Techniques for Gradient Estimates in Reinforcement Learning", [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.19.9583&rep=rep1&type=pdf.

[36] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning", *Machine Learning*, vol. 8, no. 3-4, pp. 229–256, May 1992, ISSN: 0885-6125. DOI: 10.1007/BF00992696. [Online]. Available: http://link.springer.com/10.1007/BF00992696.

[37] V. Mnih, A. Puigdomènech Badia, M. Mirza, A. Graves, T. Harley, T. P. Lillicrap, D. Silver, K. Kavukcuoglu, K. Com, and G. Deepmind, "Asynchronous Methods for Deep Reinforcement Learning", [Online]. Available: https://arxiv.org/pdf/1602.01783.pdf.

[38] S. Kakade, S. Kakade, and J. Langford, "Approximately Optimal Approximate Reinforcement Learning", *IN PROC. 19TH INTERNATIONAL CONFERENCE ON MACHINE LEARNING*, pp. 267–274, 2002. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.7.7601.

[39] R. S. Sutton and A. G. Barto, *Reinforcement learning : an introduction*. MIT Press, 1998, p. 322, ISBN: 0262193981. [Online]. Available: http://dl.acm.org/citation.cfm?id=551283.

[40] L. Metz, B. Poole, D. Pfau, and J. Sohl-Dickstein, "Unrolled Generative Adversarial Networks", Nov. 2016. [Online]. Available: https://arxiv.org/abs/1611.02163.

[41] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization", Dec. 2014. [Online]. Available: http://arxiv.org/abs/1412.6980.

[42] T. Tieleman and G. Hinton, "Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude", *COURSERA: Neural networks for machine learning*, vol. 4, no. 2, pp. 26–31, 2012.

[43] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. Courville, "Improved Training of Wasserstein GANs Montreal Institute for Learning Algorithms", [Online]. Available: https://arxiv.org/pdf/1704.00028.pdf.

[44] P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup, and D. Meger, "Deep Reinforcement Learning that Matters", Sep. 2017. [Online]. Available: http://arxiv.org/abs/1709.06560.

[45] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks", [Online]. Available: http://proceedings.mlr.press/v9/glorot10a/glorot10a.pdf.

[46] J. Ho and D. Greveling, *Personal Email Conversation*, 2017.

[47] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning", Sep. 2015. [Online]. Available: http://arxiv.org/abs/1509.02971.

[48] D. Pfau and O. Vinyals, "Connecting Generative Adversarial Networks and Actor-Critic Methods", Oct. 2016. [Online]. Available: http://arxiv.org/abs/1610.01945.

[49] C. K. Sønderby, J. Caballero, L. Theis, W. Shi, and F. Huszár, "Amortised MAP Inference for Image Super-resolution", Oct. 2016. [Online]. Available: http://arxiv.org/abs/1610.04490.