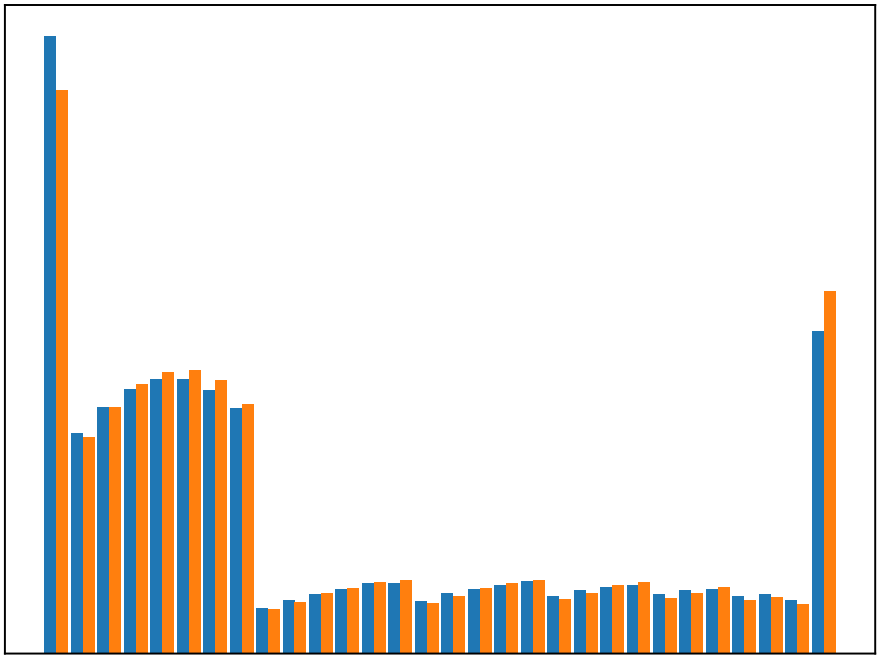
CHALMERS
UNIVERSITY OF TECHNOLOGY



Predicting Customer Behavior Using Adversarial Imitation Learning

Master's Thesis in Complex Adaptive Systems

Anton Matsson
Victor Olsson

Department of Electrical Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2020

Master's Thesis 2020

Predicting Customer Behavior Using Adversarial Imitation Learning

Anton Matsson
Victor Olsson

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Electrical Engineering
Chalmers University of Technology
Gothenburg, Sweden, 2020

Predicting Customer Behavior Using Adversarial Imitation Learning
Anton Matsson and Victor Olsson

Supervisor: Adam Andersson, Smartr
Examiner: Lennart Svensson, E2

Master's Thesis 2020
Department of Electrical Engineering
Chalmers University of Technology
SE-412 96 Gothenburg

Cover: Distribution of the last week's actions given that no purchase is made today. The blue and orange bars correspond to the average purchasing behavior of an agent trained with GAIL and 100 customers in the given data set, respectively.

Typeset in LaTeX
Gothenburg, Sweden, 2020

Everything that glitters is not gold.

Predicting Customer Behavior Using Adversarial Imitation Learning
Anton Matsson and Victor Olsson
Department of Electrical Engineering
Chalmers University of Technology

Abstract

Over the past years, there has been a rapid increase in the amount of created data. Many companies utilize customer data to improve customer experience and increase revenue. A concrete example is recommender systems that provide customers with personalized product recommendations given the customers' previous purchasing behavior. Reinforcement learning (RL) can be used to optimize such a system. To prevent a suboptimal RL algorithm from interacting with real customers, the algorithm can advantageously be trained in a realistic simulator of the real environment.

While current work put little emphasis on the explicit behavior of simulated customers, we study the possibility to simulate realistic customer purchasing behaviors in more detail. To be precise, we compare the performance of two different imitation learning algorithms: *generative adversarial imitation learning* (GAIL) and *adversarial inverse reinforcement learning* (AIRL). We train the algorithms on synthetic data in the form of time-series of daily purchases of consumer goods. In order to assess whether the simulated purchasing behaviors are realistic, we develop a method for converting time-series into distributions. The distributions are then compared using the Wasserstein metric.

We find that the performance of the algorithms is highly dependent on the choice of state representation. When using a state that simply consists of purchase history, both algorithms fail to learn individual behaviors. By extending the state with a one-hot vector that encodes individual customers, we see a great improvement in performance. The distribution of customer purchasing behaviors is in fact formed by mixture components that represent individual behaviors. To take this into account, we propose *matched mixture component trained GAIL* (MMCT-GAIL). Using this method, we manage to predict individual customer behaviors with faster convergence. Additionally, MMCT-GAIL enables the one-hot vector to be replaced with additional customer-specific purchase history, without compromising performance.

Keywords: imitation learning, reinforcement learning, inverse reinforcement learning, PPO, GAIL, AIRL, machine learning, artificial intelligence.

Acknowledgements

First of all, we would like to thank our supervisor at Smartr, Adam Andersson, for his guidance, support and never ending enthusiasm throughout the project. We would also want to thank our supervisor at Chalmers, Lennart Svensson, for his advice and continuous assistance.

Furthermore, we want to give a large thank you to Ebba Josefson Lindqvist at AI Innovation of Sweden for giving us access to their computing resources. Also, thanks to Anders Logg and Mathematical Sciences at Chalmers for giving us access to their computational resources (Ozzy). We also appreciate the help received from Carl Lundholm to get started with Ozzy.

Finally, we want to thank both of our families for their everlasting support during our studies.

Anton Matsson and Victor Olsson, Gothenburg, June 2020

Contents

List of Figures

List of Tables

Summary of Notation

A function to be optimized is denoted objective function. However, when the objective function is to be minimized we also use the term loss function. We use these terms interchangeably in this work. Moreover, we write real valued vectors using bold lower case symbols.

$:=$	equal by definition
$\mathbb{P}(X = x)$	probability that a random variable X takes the value x
$p(x)$	probability distribution over x
$x \sim p$	random variable x sampled from p
$\mathbb{E}_{x \sim p}[x]$	expected value of random variable x
$\mathbf{s}$	state
$\mathbf{a}$	action
$\mathbf{s}'$	next state
$\boldsymbol{\tau}$	trajectory of states and actions
π	policy
$R(\mathbf{s}, \mathbf{a})$	reward function
$V^{\pi}(\mathbf{s}, \mathbf{a})$	value function for policy π
$A^{\pi}(\mathbf{s}, \mathbf{a})$	advantage function for policy π
$Q^{\pi}(\mathbf{s}, \mathbf{a})$	state-action value function for policy π
λ	discount factor
G_t	discounted return at time step t
$\boldsymbol{\phi}$	reward features
$\boldsymbol{\mu}$	feature expectations
$\mathcal{L}$	objective function
ρ	occupancy measure
H	entropy
σ	activation function
α	learning rate
$\mathcal{D}$	training data set

1
Introduction

The amount of created data has rapidly increased over the past years as a result of digitalization. Consequently, many companies have realized that customer data is a valuable asset. Using customer data wisely can provide added value for both the customers and the company. As an example, consider recommender systems that provide customers with personalized recommendations. Most likely, you have noticed that, e.g., YouTube, Netflix and Spotify provide you with recommendations on videos, movies and songs based on your preferences, i.e., the data you create, when using these media services. An intelligent recommender system can improve your experience as a customer by for example letting you explore new songs that you would not have discovered on your own.

Recommender systems are also used in the retail industry. E-commerce platforms like Amazon and Alibaba use customer data to optimize their search engines in order to improve the customer experience and increase the revenue of the company. Naturally, such a recommender system is based on an interaction between the customer and the system. The customer sends a search request to the system which in turn displays a number of commodities that are intended to match the customer's request. The customer then gives feedback to the system by making a decision with respect to these recommendations.

From a business perspective, the goal is the maximize the revenue by optimizing the strategy of displaying commodities. The optimization problem is complicated by the fact that the desired feedback – the customer making a purchase – may be delayed. That is, there may be a period of time between recommendation and transaction. Perhaps the customer wants to explore different options before making a purchase. Reinforcement learning (RL) is a machine learning technique that can be used to solve problems where delayed feedback is present. Therefore, optimizing a recommender system can be done using RL.

Reinforcement learning has received a lot of attention because its ability to master the classic Atari games [1]. Among other things, RL can also be used to manage an investment portfolio [2] and design drugs [3]. The RL framework consists of a software agent who aims to develop an optimal behavior for a particular task by interacting with an environment. More specifically, the agent learns a *policy* which is a mapping from *states* to *actions*. Taking the Atari game Pong as an example, the state could be a sequence of frames from the game whilst possible actions are to move the paddle up or down.

The learning process in RL is based on a trial-and-error search. That is, the agent learns by experiencing both failures and successes when exploring the environment. The policy is iteratively updated based on the agent's experience. In the case of optimizing a recommender system, the customers constitute the environment. The agent may therefore provide the customers with suboptimal recommendations during its learning process. Obviously, experimenting with suboptimal recommendations can mean a poor user experience for the customers.

To avoid such a scenario, an alternative way to optimize a recommender system is to apply RL to a simulated customer environment. By simulating the customers, the recommender system can be optimized without risking exposing customers to poor user experience. Such a simulator can be built using *imitation learning* (IL). IL has similarities to RL, but instead of learning a behavior from scratch, the IL agent is provided with demonstrations of an expert's behavior. The idea is that the agent should learn to behave as the expert by studying these demonstrations. In the case of simulating customers, real customer data would be a natural choice of expert demonstrations.

As an example, Alibaba has created a simulator of their E-commerce platform Taobao [4]. The simulator, called Virtual Taobao, is built using a multi-agent IL technique which means that the behavior of both the platform, i.e., the recommender system, and the customers is learned simultaneously. The recommender system can then be optimized by applying RL to the simulated customers. Naturally, the end result depends on how well the customers can be simulated, i.e., how well the IL agent learns to imitate the expert demonstrations. Even though Alibaba claims that the properties of Virtual Taobao are similar to the properties of Taobao, they do not explicitly analyze the behavior of the simulated customers.

1.1 Objective

In this thesis, we study the possibility to simulate realistic customer behaviors in more detail. Unlike Alibaba, we focus only on the behavior of the customers, i.e., we have no intention to build, or for that matter optimize, a recommender system. In particular, we focus on the statistical purchasing behavior of the customers. We are given synthetic customer data from Smartr[1] in the form of time-series that represent daily purchases of consumer goods. Using this data as expert demonstrations, we evaluate the performance of two *adversarial imitation learning* (AIL) techniques, namely *generative adversarial imitation learning* (GAIL) and *adversarial inverse reinforcement learning* (AIRL).

AIL algorithms are related to so-called generative adversarial networks (GANs), where two components – the generator and the discriminator – compete against

[1]Smartr is the company that has initiated this work.

each other in order to produce data that ideally should follow the distribution of a given data set. In our case, we use AIL to generate realistic customer behaviors that follow the distribution of the synthetic customer data. The main difference between GAIL and AIRL is that AIRL recovers the underlying *reward function* of the experts. Given a state, the reward function indicates how good a certain action is. The better the IL agent learns to behave as the experts, the higher is the reward.

1.2 Contributions

As mentioned above, a critical point in simulating customers is that the simulated customers should behave similarly to the real customers. In order to fairly judge the quality of the simulated customer behaviors, we develop a method for comparing time-series. This method is intended to capture a customer's statistical purchasing behavior. More specifically, the time-series are converted into distributions of weekly purchases. We measure the difference between behaviors by measuring the distance between the corresponding distributions.

We find that the performance of both GAIL and AIRL is highly dependent on the choice of state representation. When using a state that simply consists of historical purchases, neither GAIL nor AIRL is able to predict individual customer behaviors, but only the average behavior of the customers. However, if the state is extended with a customer representative one-hot vector, we obtain a clear improvement in learning individual customer behaviors. We also propose to replace the one-hot vector with an additional sequence of purchase history – customer specific (CS) days – with the aim to generalize to new customers. However, using GAIL and AIRL, we do not obtain a desirable result with this state representation.

The standard way of training AIL algorithms assumes that the experts behave according to *one* underlying expert policy that represents the behavior of the experts. However, since customer behaviors are highly varying we instead assume that each expert behaves according to its own underlying expert policy – a mixture component of the expert policy. We propose *matched mixture component trained GAIL* (MMCT-GAIL) as a novel approach to the specific problem of learning diverse customer behaviors. Using this method, we manage to predict individual customer behaviors with faster convergence in comparison with GAIL and AIRL. In addition, MMCT-GAIL shows promising results when extending a state consisting of historical purchases with CS days.

1.3 Related Work

GAIL, proposed in 2016 by Ho et al. [5], revolutionized the field of imitation learning by proposing learning the expert policy in an adversarial framework without explicitly estimating a reward function. Many imitation learning algorithms are based on GAIL [6, 7, 8, 9, 10]. AIRL extends the work of GAIL by estimating the true reward function whereas variational adversarial imitation learning (VAIL) [8]

improves the training by proposing constraining the information flow to the discriminator. Nonetheless, apart from Virtual Taobao built by Alibaba, not much work has been presented where imitation learning is used to predict customer purchasing behaviors.

There are however other methods that have been proposed to the problem of predicting customer behaviors. For instance, predicting a customer's purchasing behavior can be seen as a time-series forecasting problem where a sequence of previous purchases are used to predict future purchases. In general, recurrent neural networks, and more specifically long short term memory networks (LSTMs), are seen as state-of-the-art methods for modeling sequential data.

In particular, LSTMs are commonly used to solve recommendation tasks in E-commerce sites. An example is the dynamic recurrent basket model (DREAM) [11], where an LSTM suggests a relevant item for a customer given historical purchases. Another example is the time-LSTM [12], a modified LSTM that is able to capture both the short term interest and the long term interest of a customer.

Other methods often used to forecast time series are Bayesian models and ARIMA models [13, 14]. In Bayesian modeling, the posterior probability of an event can be expressed in terms of a prior and a likelihood. In other words, a customer's daily action can be expressed as a probability given the customer's previous actions. ARIMA methodology on the other hand tries to describe a time series as a function of autoregressive and moving average parameters that are to be found by, e.g., a maximum likelihood approach.

1.4 Thesis Outline

The thesis is structured as follows. In Chapter 2, we present the theory behind GAIL and AIRL. In order to do so, we start by introducing artificial neural networks and generative adversarial networks. Thereafter, we give an overview of reinforcement learning, in particular trust region policy optimization (TRPO) and proximal policy optimization (PPO). Finally, we present a thorough walk-through of the field of imitation learning, including behavioral cloning and inverse reinforcement learning, ending with GAIL and AIRL.

In Chapter 3, we outline our methods. First, we briefly describe the synthetic customer data and develop a method for comparing time-series, which is needed to assess the quality of simulated behaviors. Thereafter, we explain our implementations of GAIL and AIRL, as well as the ideas behind MMCT-GAIL. We introduce three different state representations. In Chapter 4, we present and discuss the results from experimental evaluations of the algorithms, before concluding the thesis in Chapter 5.

2
Preliminaries

This chapter presents central concepts that are used in this thesis. We cover the fundamentals of artificial neural networks, reinforcement learning and imitation learning. For each concept, we gradually introduce more complex methods. The main purpose of this chapter is to provide the reader with a good understanding of GAIL and AIRL. Both GAIL and AIRL rely on deep reinforcement learning, which in turn relies on artificial neural networks.

2.1 Artifical Neural Networks

Artificial neural networks or simply neural networks (NNs) constitute a group of mathematical methods inspired by the interactions between the neurons in the brain [15]. A neural network can be used for classification tasks but has also proven to be a universal function approximator [16]. For instance, most modern reinforcement learning algorithms use NNs to parameterize, e.g., the policy. In this context, it is common to use a multilayer perceptron as presented in Section 2.1.1.

Both GAIL and AIRL are based on a generative adversarial framework which is reminiscent of so-called generative adversarial networks (GANs). The theory behind GANs is presented in Section 2.1.2. GANs consist of two networks that compete against each other. This architecture makes GANs hard to train, and they often suffer from various stability problems. Therefore, we also describe WGAN and WGAN-GP, which are improved versions of the original GAN.

2.1.1 Multilayer Perceptron

A *multilayer perceptron* (MLP) is a fully connected feed-forward neural network that consists of N layers of computational units, i.e., nodes or neurons. An MLP is a mathematical function which maps an input $\mathbf{x}_0 \in \mathbb{R}^n$ to an output $\mathbf{x}_N \in \mathbb{R}^m$. The network is usually parameterized by a weight vector $\boldsymbol{\theta} \in \mathbb{R}^p$ that consists of the matrices $(\boldsymbol{\theta}_1, \dots, \boldsymbol{\theta}_N)$ and bias vectors $(\mathbf{b}_1, \dots, \mathbf{b}_N)$. An MLP is defined as

$$\mathbf{x}_n = \sigma(\boldsymbol{\theta}_n \mathbf{x}_{n-1} + \mathbf{b}_n) \qquad n = 1, \dots, N-1 \tag{2.1}$$

and

$$\mathbf{x}_N = \boldsymbol{\theta}_N \mathbf{x}_{N-1} + \mathbf{b}_N. \tag{2.2}$$

The $\mathbf{x}_n$, $n = 0, \dots, N$, can have different dimensions and the matrices $\boldsymbol{\theta}_n$, $n = 1, \dots, N$, and bias vectors $\mathbf{b}_n$, $n = 1, \dots, N$, have the suitable dimensions. The

function σ is an *activation function* which is a vector valued extension of a scalar valued nonlinear function $\mathbb{R} \to \mathbb{R}$. Below we write σ for both the scalar valued function and its extension. Some of the most commonly used activation functions are

- logistic sigmoid: $\sigma(x) = \frac{1}{1+\exp(-x)}$
- hyperbolic tangent: $\sigma(x) = \tanh(x)$
- rectified linear unit (ReLu): $\sigma(x) = \max\{0, x\}$
- softmax: $\sigma(\mathbf{x}) = \frac{\exp(x_i)}{\sum_{j=1}^{K} \exp(x_j)}$ for $i = 1, \ldots, K$ and $\mathbf{x} = (x_1, \ldots, x_k) \in \mathbb{R}^K$.

Define the training set as $\mathcal{D} = \{\mathbf{x}_0^i, \mathbf{y}^i\}_{i=1}^N$ where $\mathbf{x}_0^i$ and $\mathbf{y}^i$ represent input and corresponding target, respectively, for data point i. The training objective is to update the parameters $\boldsymbol{\theta}$ so that a loss function $\boldsymbol{\theta} \mapsto \mathcal{L}(\boldsymbol{\theta}; \mathcal{D})$ is minimized. Thus, optimal parameters $\boldsymbol{\theta}^*$ satisfy

$$\boldsymbol{\theta}^* = \underset{\boldsymbol{\theta}}{\operatorname{argmin}}\, \mathcal{L}(\boldsymbol{\theta}; \mathcal{D}). \tag{2.3}$$

Common choices for $\mathcal{L}$ is cross-entropy loss for classification tasks and mean square error loss for regression. All loss functions can be written on the form

$$\mathbb{E}_{(\mathbf{x}_0, \mathbf{y}) \sim \mathcal{U}(\mathcal{D})}[\ell(\mathbf{x}_0, \mathbf{y}; \boldsymbol{\theta})] \tag{2.4}$$

for some scalar valued function ℓ. Classically, the loss is gradually minimized through gradient descent where the update rule is defined as

$$\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t - \alpha \nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}_t; \mathcal{D}), \tag{2.5}$$

where α is the learning rate. There are three main algorithms performing (2.5), namely *batch gradient descent*, *stochastic gradient descent* (SGD) and *mini-batch gradient descent*. The learning rate for these gradient descent variants is constant for all the parameters in $\boldsymbol{\theta}$. This might cause problems if the components in $\boldsymbol{\theta}$ vary in magnitude or if the training data is sparse, i.e., contains a lot of zeros [17]. Therefore, several gradient descent optimization methods have been proposed where the learning rate varies, e.g., *momentum optimizer*, *Adagrad*, *AdaMax* and *Adam*. Adam (Adaptive moment estimation) is most commonly used due to its good performance [18].

Adam calculates an individual learning rate for each parameter using an adaptive moment estimation of the gradient [19]. The update rule in Adam is defined as

$$\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t - \frac{\alpha}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t \tag{2.6}$$

where ϵ is a smoothing term to avoid division by zero. $\hat{m}_t$ and $\hat{v}_t$ are estimates of the first and second moment of the gradient, respectively, i.e., mean and variance of

the gradient, defined as

$$\begin{aligned} \hat{m}_t &= \frac{m_t}{1-\beta_1^t}, \\ \hat{v}_t &= \frac{v_t}{1-\beta_2^t}, \end{aligned} \tag{2.7}$$

where m_t and v_t are defined as

$$\begin{aligned} m_t &= \beta_1 m_{t-1} + (1-\beta_1)\,\nabla_{\boldsymbol{\theta}}\mathcal{L}(\boldsymbol{\theta}_t), \\ v_t &= \beta_2 v_{t-1} + (1-\beta_2)\,\nabla_{\boldsymbol{\theta}}\mathcal{L}(\boldsymbol{\theta}_t)^2, \end{aligned} \tag{2.8}$$

and β_1 and β_2 are exponential decay rates for the moment estimates.

2.1.2 Generative Adversarial Networks

In generative modeling there exists an unknown true distribution $p_r(\mathbf{x})$ and the objective is to find an estimate of $p_r(\mathbf{x})$ by training a model on a finite number of samples from the true distribution. Goodfellow et al. [20] propose an approach to training generative models known as generative adversarial networks (GANs). It is based on two neural networks that are trained simultaneously: a *generator* G and a *discriminator* D. The discriminator is a classifier trained to separate samples from the true distribution from samples generated by the generator. The generator is on the other hand trained to generate samples that the discriminator classifies as samples from the true distribution.

The generator G takes noise $\mathbf{z} \sim p_z$ as input and outputs a sample $\mathbf{x} = G(\mathbf{z})$. We denote the distribution of $\mathbf{x}$ by p_g. It is the pushforward distribution of p_z under G. The discriminator takes a sample $\mathbf{x}$ from either p_r or p_g as input and outputs a probability $D(\mathbf{x})$ that the given sample $\mathbf{x}$ comes from the true distribution. The discriminator is trained to classify the samples correctly whereas the generator is trained to minimize $\log(1 - D(G(\mathbf{z})))$. Thus, the discriminator and the generator are playing a min-max game which can be summarized as

$$\min_G \max_D \mathbb{E}_{\mathbf{x}\sim p_r}[\log D(\mathbf{x})] + \mathbb{E}_{\mathbf{z}\sim p_z}[\log(1 - D(G(\mathbf{z})))]. \tag{2.9}$$

Goodfellow et al. show that (2.9) is equal to minimizing the *Jensen-Shannon* (JS) divergence[1] between p_r and p_g. Since each player in this min-max game is optimizing the objective function independently the task becomes a problem of finding a *Nash equilibrium* [21]. A Nash equilibrium is defined as a solution to a non-cooperative game with two or more players involved, for which no player gains anything by changing its own strategy [22].

In practice it can be very difficult to train a GAN since reaching a Nash equilibrium is delicate when the objective function for each player is non-convex and the parameter

[1] Let $p(x)$ and $q(x)$ be probability distributions over the discrete variable x. The Jensen-Shannon divergence is a measure of the similarity between the distributions, defined as $D_{JS}(p||q) = \frac{1}{2}D_{KL}(p||m) + \frac{1}{2}D_{KL}(q||m)$, where $m = \frac{1}{2}(p+q)$ and D_{KL} is the Kullback-Leibler divergence defined in Section 2.2.4. The JS divergence is a symmetrized and smoothed version of the KL divergence.

space is high-dimensional [21]. The most common training problems that GANs suffer from are vanishing gradients and mode collapse. If the generator experiences mode collapse it only produces a limited varieties of samples. A common example is when using GANs to generate images with numbers from one to ten. If mode collapse occurs, the generator ends up producing one, or only a few, of the numbers.

2.1.3 Wasserstein GAN

To stabilize the training of GANs, Arjovsky et al. [23] propose a different objective where the Wasserstein distance between the distributions p_r and p_g is minimized, hence the name Wasserstein GAN (WGAN). This approach prevents vanishing gradients as a result of the objective being continuously differentiable and having a smoother gradient than the original GAN objective [23].

The Wasserstein distance between two distributions can be understood as the minimum "cost" required to move a "mass" from point $\mathbf{x}$ to point $\mathbf{y}$ in order to transform $p_r(\mathbf{x})$ to $p_g(\mathbf{y})$. It is defined as

$$W\left(p_r, p_g\right) = \inf_{\gamma \in \Pi(p_r, p_g)} \mathbb{E}_{(\mathbf{x}, \mathbf{y}) \sim \gamma}[\|\mathbf{x} - \mathbf{y}\|] = \inf_{\gamma \in \Pi(p_r, p_g)} \sum_{\mathbf{x}, \mathbf{y}} \|\mathbf{x} - \mathbf{y}\| \gamma(\mathbf{x}, \mathbf{y}) \tag{2.10}$$

where $\Pi\left(p_r, p_g\right)$ denotes the set of all possible joint probability distributions with marginals p_r and p_g. By using the so called Kantorovich-Rubinstein duality [24], an equivalent formulation of the Wasserstein distance is

$$W\left(p_r, p_g\right) = \sup_{\|f\|_L \leq 1} \mathbb{E}_{\mathbf{x} \sim p_r}[f(\mathbf{x})] - \mathbb{E}_{\mathbf{x} \sim p_g}[f(\mathbf{x})], \tag{2.11}$$

where the supremum is taken over all functions f that are 1-Lipschitz. We refer to [23] for details about Lipschitz functions. In order to calculate the Wasserstein distance between p_r and p_g, the 1-Lipschitz function f which minimizes (2.11) should be found. An approximation of f can be calculated by training a neural network with weights $\mathbf{w}$.

In WGAN, the parameterized function $f_{\mathbf{w}}$ corresponds to the discriminator $D_{\mathbf{w}}$ in original GAN. A neural network that approximates a function should not have an activation function after its output layer. This means that the output of the discriminator in WGAN is no longer a probability but instead a scalar which expresses how realistic a sample from the generator is.

The objective of the min-max game between discriminator and generator can now be summarized as

$$\min_G \max_{D \in \mathcal{F}} \mathbb{E}_{\mathbf{x} \sim p_r}[D_{\mathbf{w}}(\mathbf{x})] - \mathbb{E}_{\mathbf{x} \sim p_g}\left[D_{\mathbf{w}}(\mathbf{x})\right)] \tag{2.12}$$

where $\mathcal{F}$ is the set of 1-Lipschitz functions. To enforce that $D_{\mathbf{w}}$ is a 1-Lipschitz function, WGAN introduces weight clipping to ensure that the weights $\mathbf{w}$ are within a certain range $[-c, c]$, controlled by a clipping parameter c. Nonetheless, Arjovsky et al. clearly state that using weight clipping to enforce the Lipscithz constraint is not an optimal solution. If the clipping parameter is large it can be difficult to reach convergence whereas a small clipping parameter can lead to vanishing gradients.

2.1.4 Wasserstein GAN with Gradient Penalty

Gulrajani et al. [25] propose Wasserstein GAN with gradient penalty (WGAN-GP) which uses an alternative method to enforce a Lipschitz constraint on the discriminator. A differentiable function is a 1-Lipschitz function if and only if it has gradients with norm that is at most one everywhere. Therefore, Gulrajani et al. propose to constrain the gradient norm of the discriminator output $D(\mathbf{x})$ with respect to its input $\mathbf{x}$. The new objective becomes

$$\min_G \max_{D \in \mathcal{F}} \mathop{\mathbb{E}}_{\tilde{\mathbf{x}} \sim p_g} [D(\tilde{\mathbf{x}})] - \mathop{\mathbb{E}}_{\mathbf{x} \sim p_r} [D(\mathbf{x})] + \lambda \mathop{\mathbb{E}}_{\hat{\mathbf{x}} \sim p_{\hat{\mathbf{x}}}} \left[\left(\left\| \nabla_{\hat{\mathbf{x}}} D(\hat{\mathbf{x}}) \right\|_2 - 1 \right)^2 \right] \tag{2.13}$$

which introduces a soft version of the norm constraint. The parameter λ is called the gradient penalty coefficient. The distribution $p_{\hat{\mathbf{x}}}$ is the pushforward of p_r and p_g under $\hat{\mathbf{x}} = \epsilon \mathbf{x} + (1 - \epsilon)\tilde{\mathbf{x}}$ for $\mathbf{x} \sim p_r$ and $\tilde{\mathbf{x}} \sim p_g$ where $\epsilon \in [0, 1]$. In other words, the sampling from $p_{\hat{\mathbf{x}}}$ is done uniformly along straight lines between pair of points sampled from p_r and p_g which Gulrajani et al. motivate theoretically.

2.2 Reinforcement Learning

Alongside supervised learning and unsupervised learning, reinforcement learning (RL) is one of the major paradigms in machine learning. By learning from interactions with its environment, the RL agent aims to develop an optimal behavior for achieving a certain goal. The purpose of this section is to present the theory behind proximal policy optimization (PPO), which is a particular type of RL algorithm that is used in our implementations of GAIL and AIRL. PPO belongs to the family of policy gradient methods and is inspired by trust region policy optimization (TRPO).

We start by formulating the RL problem as a finite Markov decision process in Section 2.2.1. In Section 2.2.2, we then explain the general concept of policy gradient methods. TRPO and PPO are finally presented in Section 2.2.3 and 2.2.4, respectively.

2.2.1 Problem Statement

The reinforcement learning problem is usually formulated using a *Markov decision process* (MDP). An MDP is represented by the tuple $\mathcal{M} := \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \gamma, R \rangle$, where $\mathcal{S}$ is a finite set of states, $\mathcal{A}$ is a finite set of actions, $\mathcal{P}$ is the state-transition probability function, $\gamma \in [0, 1]$ is a discount factor, and $R : \mathcal{S} \times \mathcal{A} \mapsto \mathbb{R}$ is the reward function. For all $\mathbf{s}'$, $\mathbf{s} \in \mathcal{S}$ and $\mathbf{a} \in \mathcal{A}$, the state-transition probability function

$$\mathcal{P}(\mathbf{s}'|\mathbf{s}, \mathbf{a}) := \mathbb{P}(\mathbf{S}_t = \mathbf{s}'|\mathbf{S}_{t-1} = \mathbf{s}, \mathbf{A}_{t-1} = \mathbf{a}) \tag{2.14}$$

specifies the probability of reaching a particular state $\mathbf{s}'$ given the preceding state $\mathbf{s}$ and action $\mathbf{a}$. The state-transition function defines the dynamics of the MDP.

The interaction between the *agent* and the *environment* is central in an MDP. This interaction occurs over a sequence of time steps $t = 0, 1, 2, \ldots, T$, where T may

be infinitely large[2]. At each time step, the state $\mathbf{s}_t \in \mathcal{S}$ provides the agent with some information about the environment. By executing an action $\mathbf{a}_t$, the agent is transferred into a new state $\mathbf{s}_{t+1}$ according to the dynamics of the MDP. Simultaneously, the agent receives a scalar reward r_{t+1} that indicates the quality of the action $\mathbf{a}_t$. The sequence of states and actions – the state-action pairs – forms a so-called trajectory, defined as $\boldsymbol{\tau} = (\mathbf{s}_t, \mathbf{a}_t)_{t=0}^{T}$.

The goal of the agent is to maximize the expected discounted sum of future rewards, also called discounted return. The discounted return is given by

$$G_t := \sum_{k=0}^{T} \gamma^k r_{t+k+1}, \tag{2.15}$$

where the discount factor γ specifies how much the agent should care about future rewards. If $T = \infty$, the discount factor must satisfy $\gamma < 1$. Conversely, if $\gamma = 1$, T must be finite. The main concepts of RL can be formulated in either case, but it is convenient to stick to one notation. From now on, we mostly focus on continuing tasks, i.e., we allow T to be infinite.

While interacting with the environment, the agent follows a so-called policy π. The policy is either deterministic, $\mathbf{a} = \pi(\mathbf{s})$, or stochastic, $\mathbf{a} \sim \pi(\mathbf{a}|\mathbf{s})$. A stochastic policy maps the agent's current state to probabilities of selecting each possible action, and the agent takes an action by sampling from the policy. The policy is iteratively updated based on the agent's experience. In the long run, the learned policy should allow the agent to act in a way that maximizes the expected discounted return, as given in (2.15).

Two important concepts in RL are the state-value function, or simply the value function, and the action-value function, respectively. The state-value function is defined as

$$V^{\pi}(\mathbf{s}) := \mathbb{E}_{\pi}\left[G_t | \mathbf{S}_t = \mathbf{s}\right] = \mathbb{E}_{\pi}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \middle| \mathbf{S}_t = \mathbf{s}\right] \tag{2.16}$$

and the action-value function is given by

$$Q^{\pi}(\mathbf{s}, \mathbf{a}) := \mathbb{E}_{\pi}\left[G_t | \mathbf{S}_t = \mathbf{s}, \mathbf{A}_t = \mathbf{a}\right] = \mathbb{E}_{\pi}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \middle| \mathbf{S}_t = \mathbf{s}, \mathbf{A}_t = \mathbf{a}\right]. \tag{2.17}$$

Here, the expected value is taken with respect to policy π and state-transition probability function $\mathcal{P}$. To simplify notation, we omit the explicit dependency of $\mathcal{P}$. The state-value function is the expected discounted return when the agent starts in state $\mathbf{s}$ and thereafter follows policy π. Similarly, the action-value function is the expected discounted return when the agent starts in state $\mathbf{s}$, takes the action $\mathbf{a}$ and thereafter follows policy π. Additionally, the advantage function is given by the difference between these functions:

$$A^{\pi}(\mathbf{s}, \mathbf{a}) = Q^{\pi}(\mathbf{s}, \mathbf{a}) - V^{\pi}(\mathbf{s}). \tag{2.18}$$

[2]We refer to episodic tasks (finite T) and continuing tasks (infinite T), respectively.

2.2.2 Policy Gradient Methods

There are two major approaches to solving a reinforcement learning problem: value-based RL and policy-based RL. In value-based RL, the task is to learn an approximation of the state-value function or the action-value function. A policy can then be selected by, e.g., acting greedy with respect to the values. In this context, acting greedy simply means choosing states that have the highest values. In policy gradient methods, the policy is directly parameterized and there is no need for any value estimates. Such methods are examples of policy-based RL. A great advantage of policy gradient methods is that the action probabilities change smoothly as a function of the parameters, which is not always the case in value-based methods. For this reason, there are stronger convergence guarantees for policy gradient methods [26].

In order to explain policy gradient methods in greater detail, let $\boldsymbol{\theta} \in \mathbb{R}^d$ be the parameters of the parameterized policy $\pi(\mathbf{a}|\mathbf{s};\boldsymbol{\theta})$. Now, the task is to maximize an objective function $\mathcal{L}(\boldsymbol{\theta})$ by performing stochastic gradient ascent. For episodic tasks, starting in state $\mathbf{s}_0$, the objective function $\mathcal{L}(\boldsymbol{\theta})$ is defined as

$$\mathcal{L}(\boldsymbol{\theta}) := V^{\pi_{\boldsymbol{\theta}}}(\mathbf{s}_0) = \mathbb{E}_{\pi_{\boldsymbol{\theta}}}\left[G_t|\mathbf{S}_t = \mathbf{s}_0\right]. \tag{2.19}$$

Thus, the performance is evaluated based on the expected, possibly discounted, return starting in the initial state $\mathbf{s}_0$. For continuing tasks, the objective function must be chosen in a somewhat different way, but the *policy gradient theorem* [26] states that the policy gradient is the same for both cases, namely

$$\nabla_{\boldsymbol{\theta}}\mathcal{L}(\boldsymbol{\theta}) = \mathbb{E}_{\pi_{\boldsymbol{\theta}}}\left[\sum_{t=0}^{T}\gamma^t\nabla_{\boldsymbol{\theta}}\ln\pi_{\boldsymbol{\theta}}(\mathbf{a}_t|\mathbf{s}_t)Q^{\pi_{\boldsymbol{\theta}}}(\mathbf{s}_t,\mathbf{a}_t)\right]. \tag{2.20}$$

We refer to [26] for details regarding the proof of this result.

The action-value function in (2.20) can be estimated by the return G_t. Recall from Section 2.2.1 that $Q^{\pi_{\boldsymbol{\theta}}}(\mathbf{s}_t,\mathbf{a}_t) = \mathbb{E}_{\pi_{\boldsymbol{\theta}}}\left[G_t|\mathbf{s}_t,\mathbf{a}_t\right]$. Some algorithms are based on learning a parameterized action-value function $Q_{\mathbf{w}}(\mathbf{s},\mathbf{a}) \approx Q^{\pi_{\boldsymbol{\theta}}}(\mathbf{s},\mathbf{a})$. This type of algorithms is known as actor-critic. In fact, there are actor-critic algorithms that learn a parameterized value function $V_{\mathbf{w}}(\mathbf{s}) \approx V^{\pi_{\boldsymbol{\theta}}}(\mathbf{s})$ instead of a parameterized action-value function. It is worth mentioning that the parameters $\boldsymbol{\theta}$ can be used to estimate both the policy and the action-value function or value function.

It can be shown that a baseline $B(\mathbf{s}_t)$, which does not depend on any action $\mathbf{a}_t$, can be subtracted from the action-value function in (2.20) without changing the expected value of the policy gradient [26]. That is, we are allowed to replace $Q^{\pi_{\boldsymbol{\theta}}}(\mathbf{s}_t,\mathbf{a}_t)$ with $Q^{\pi_{\boldsymbol{\theta}}}(\mathbf{s}_t,\mathbf{a}_t) - B(\mathbf{s}_t)$. Such a baseline tends to reduce the variance of the estimation. A good choice of baseline is the state-value function $V^{\pi_{\boldsymbol{\theta}}}(\mathbf{s})$ [26]. Recalling the definition $A^{\pi}(\mathbf{s},\mathbf{a}) = Q^{\pi}(\mathbf{s},\mathbf{a}) - V^{\pi}(\mathbf{s})$ of the advantage, the policy gradient becomes

$$\nabla_{\boldsymbol{\theta}}\mathcal{L}(\boldsymbol{\theta}) = \mathbb{E}_{\pi_{\boldsymbol{\theta}}}\left[\sum_{t=0}^{T}\gamma^t\nabla_{\boldsymbol{\theta}}\ln\pi_{\boldsymbol{\theta}}(\mathbf{a}_t|\mathbf{s}_t)A^{\pi_{\boldsymbol{\theta}}}(\mathbf{s}_t,\mathbf{a}_t)\right]. \tag{2.21}$$

2.2.3 Trust Region Policy Optimization

Policy gradient methods are examples of on-policy learning. In on-policy learning, the policy is iteratively updated by generating samples for gradient computing, for instance according to (2.21). Unfortunately, it is complicated to reuse the samples for consecutive calculations of the policy gradient. Empirically, such an approach tends to lead to very large policy updates that complicates the learning process [27]. Thus, policy gradient methods generally suffer from poor sample efficiency.

In order to improve the sample efficiency, importance sampling can be used. Let $p(x)$ and $q(x)$ be two different probability distributions with the same support. Importance sampling enables the expectation $\mathbb{E}_{x\sim p}[f(x)]$, where f is a sufficiently regular function, to be written as $\mathbb{E}_{x\sim q}[\frac{p(x)}{q(x)}f(x)]$. For a good choice of proposal distribution q, the rewritten expectation is easier to estimate since the samples can be drawn from q instead of p. By applying this approach to (2.21), it is possible to update the policy $\pi_{\boldsymbol{\theta}}$ by sampling trajectories $\boldsymbol{\tau}$ from a different policy $\pi_{\boldsymbol{\theta}'}$:

$$\nabla_{\boldsymbol{\theta}}\mathcal{L}(\boldsymbol{\theta}) = \mathbb{E}_{\boldsymbol{\tau}\sim\pi_{\boldsymbol{\theta}'}}\left[\sum_{t=0}^{\infty}\gamma^t\frac{p(\boldsymbol{\tau}_t|\pi_{\boldsymbol{\theta}})}{p(\boldsymbol{\tau}_t|\pi_{\boldsymbol{\theta}'})}\nabla_{\boldsymbol{\theta}}\ln\pi_{\boldsymbol{\theta}}(\mathbf{a}_t|\mathbf{s}_t)A^{\pi_{\boldsymbol{\theta}}}(\mathbf{s}_t,\mathbf{a}_t)\right]. \tag{2.22}$$

Here, $p(\boldsymbol{\tau}_t|\pi_{\boldsymbol{\theta}})$ and $p(\boldsymbol{\tau}_t|\pi_{\boldsymbol{\theta}'})$ are the probability distributions for observing the subsequence $\boldsymbol{\tau}_t$ of the trajectory $\boldsymbol{\tau}$ given $\pi_{\boldsymbol{\theta}}$ and $\pi_{\boldsymbol{\theta}'}$, respectively. Given a distribution $\mu(\mathbf{s}_0)$ over initial states $\mathbf{s}_0$, the ratio between the distributions $p(\boldsymbol{\tau}_t|\pi_{\boldsymbol{\theta}})$ and $p(\boldsymbol{\tau}_t|\pi_{\boldsymbol{\theta}'})$ can be rewritten as

$$\frac{p(\boldsymbol{\tau}_t|\pi_{\boldsymbol{\theta}})}{p(\boldsymbol{\tau}_t|\pi_{\boldsymbol{\theta}'})} = \frac{\mu(\mathbf{s}_0)\prod_{t'=0}^{t}\mathcal{P}(\mathbf{s}_{t'+1}|\mathbf{s}_{t'},\mathbf{a}_{t'})\pi_{\boldsymbol{\theta}}(\mathbf{a}_{t'}|\mathbf{s}_{t'})}{\mu(\mathbf{s}_0)\prod_{t'=0}^{t}\mathcal{P}(\mathbf{s}_{t'+1}|\mathbf{s}_{t'},\mathbf{a}_{t'})\pi_{\boldsymbol{\theta}'}(\mathbf{a}_{t'}|\mathbf{s}_{t'})} = \prod_{t'=0}^{t}\frac{\pi_{\boldsymbol{\theta}}(\mathbf{a}_{t'}|\mathbf{s}_{t'})}{\pi_{\boldsymbol{\theta}'}(\mathbf{a}_{t'}|\mathbf{s}_{t'})}. \tag{2.23}$$

These equations may look useful, but even small differences between the two policies will force the product in (2.23), and thus also the gradient in (2.22), to either explode or vanish. We would like to increase the sampling efficiency by reusing data sampled from a previous version of the policy, but we need to approach the problem in a different way. In order to do so, we define the relative policy performance identity as

$$\mathcal{L}(\boldsymbol{\theta}) - \mathcal{L}(\boldsymbol{\theta}') = \mathbb{E}_{\pi_{\boldsymbol{\theta}}}\left[\sum_{t=0}^{\infty}\gamma^t A^{\pi_{\boldsymbol{\theta}'}}(\mathbf{s}_t,\mathbf{a}_t)\right]. \tag{2.24}$$

This expression relates the policies $\pi_{\boldsymbol{\theta}}$ and $\pi_{\boldsymbol{\theta}'}$ in terms of the expected discounted sum of advantages when following the policy $\pi_{\boldsymbol{\theta}}$. We refer to [28] for details regarding the proof of the identity. Using this identity, the policy optimization problem can be formulated as

$$\underset{\boldsymbol{\theta}}{\operatorname{argmax}}\,\mathcal{L}(\boldsymbol{\theta}) = \underset{\boldsymbol{\theta}}{\operatorname{argmax}}\,\mathcal{L}(\boldsymbol{\theta}) - \mathcal{L}(\boldsymbol{\theta}') = \underset{\boldsymbol{\theta}}{\operatorname{argmax}}\,\mathbb{E}_{\pi_{\boldsymbol{\theta}}}\left[\sum_{t=0}^{\infty}\gamma^t A^{\pi_{\boldsymbol{\theta}'}}(\mathbf{s}_t,\mathbf{a}_t)\right], \tag{2.25}$$

where $\boldsymbol{\theta}$ and $\boldsymbol{\theta}'$ correspond to the parameters of the current policy and an old policy, respectively, in an iterative update sequence.

However, in order to optimize (2.25), we need to sample data from the current policy $\pi_{\boldsymbol{\theta}}$. With the goal of instead reusing data from an old policy $\pi_{\boldsymbol{\theta}'}$, we introduce the discounted future state distribution $d^{\pi}(\mathbf{s})$ [29], defined as

$$d^{\pi}(\mathbf{s}) = (1-\gamma)\sum_{t=0}^{\infty}\gamma^{t}\mathbb{P}(\mathbf{S}_t = \mathbf{s}|\pi). \tag{2.26}$$

The discounted future state distribution can be interpreted as the probability of reaching state $\mathbf{s}$ at a certain time step, when starting from an initial state $\mathbf{s}_0$ and following the policy π. Using this notation in combination with importance sampling, we can reformulate (2.24) according to

$$\begin{aligned}\mathcal{L}(\boldsymbol{\theta}) - \mathcal{L}(\boldsymbol{\theta}') &= \frac{1}{1-\gamma}\mathbb{E}_{\mathbf{s}\sim d^{\pi_{\boldsymbol{\theta}}},\,\mathbf{a}\sim\pi_{\boldsymbol{\theta}}}\left[A^{\pi_{\boldsymbol{\theta}'}}(\mathbf{s},\mathbf{a})\right] \\ &= \frac{1}{1-\gamma}\mathbb{E}_{\mathbf{s}\sim d^{\pi_{\boldsymbol{\theta}}},\,\mathbf{a}\sim\pi_{\boldsymbol{\theta}'}}\left[\frac{\pi_{\boldsymbol{\theta}}(\mathbf{a}|\mathbf{s})}{\pi_{\boldsymbol{\theta}'}(\mathbf{a}|\mathbf{s})}A^{\pi_{\boldsymbol{\theta}'}}(\mathbf{s},\mathbf{a})\right],\end{aligned} \tag{2.27}$$

which states that the policy $\pi_{\boldsymbol{\theta}}$ can be optimized by sampling actions from a different policy $\pi_{\boldsymbol{\theta}'}$. There is only one problem – the states need to be sampled from the discounted future state distribution $d^{\pi_{\boldsymbol{\theta}}}(\mathbf{s})$, which is dependent on the policy $\pi_{\boldsymbol{\theta}}$.

Fortunately, it turns out that the approximation $d^{\pi_{\boldsymbol{\theta}}} \approx d^{\pi_{\boldsymbol{\theta}'}}$ works well in practice. Using this approximation, we obtain the objective function that is used in trust region policy optimization (TRPO) [30], namely

$$\begin{aligned}\mathcal{L}^{\text{TRPO}}(\boldsymbol{\theta}) &:= \frac{1}{1-\gamma}\mathbb{E}_{\mathbf{s}\sim d^{\pi_{\boldsymbol{\theta}'}},\,\mathbf{a}\sim\pi_{\boldsymbol{\theta}'}}\left[\frac{\pi_{\boldsymbol{\theta}}(\mathbf{a}|\mathbf{s})}{\pi_{\boldsymbol{\theta}'}(\mathbf{a}|\mathbf{s})}A^{\pi_{\boldsymbol{\theta}'}}(\mathbf{s},\mathbf{a})\right] \\ &= \mathbb{E}_{\boldsymbol{\tau}\sim\pi_{\boldsymbol{\theta}'}}\left[\sum_{t=0}^{\infty}\gamma^{t}h_t(\boldsymbol{\theta})A^{\pi_{\boldsymbol{\theta}'}}(\mathbf{s}_t,\mathbf{a}_t)\right] \\ &\approx \mathcal{L}(\boldsymbol{\theta}) - \mathcal{L}(\boldsymbol{\theta}'),\end{aligned} \tag{2.28}$$

where $h_t(\boldsymbol{\theta}) = \frac{\pi_{\boldsymbol{\theta}}(\mathbf{a}_t|\mathbf{s}_t)}{\pi_{\boldsymbol{\theta}'}(\mathbf{a}_t|\mathbf{s}_t)}$ is the probability ratio between the policies $\pi_{\boldsymbol{\theta}}$ and $\pi_{\boldsymbol{\theta}'}$. This objective function can be optimized by reusing samples from an old policy $\pi_{\boldsymbol{\theta}'}$, which increases the sample efficiency. The approximation $d^{\pi_{\boldsymbol{\theta}}} \approx d^{\pi_{\boldsymbol{\theta}'}}$ is valid as long as $\pi_{\boldsymbol{\theta}}$ and $\pi_{\boldsymbol{\theta}'}$ are relatively close. It can be shown, see [31], that

$$\left|(\mathcal{L}(\boldsymbol{\theta}) - \mathcal{L}(\boldsymbol{\theta}')) - \mathcal{L}^{\text{TRPO}}(\boldsymbol{\theta})\right| \leq C\sqrt{\mathbb{E}_{\mathbf{s}\sim d^{\pi_{\boldsymbol{\theta}'}}}\left[D_{KL}(\pi_{\boldsymbol{\theta}}||\pi_{\boldsymbol{\theta}'})[s]\right]}, \tag{2.29}$$

where D_{KL} denotes the *Kullback–Leibler* (KL) divergence[3] and C is a parameter of little practical interest. In other words, the difference between the objective function in (2.25) and the objective function in TRPO is bounded by the KL divergence between the old policy and the new policy.

[3]Let $p(x)$ and $q(x)$ be probability distributions over the discrete variable x. The Kullback-Leibler divergence, defined as $D_{KL}(p||q) = \sum_x p(x)\log\left(\frac{p(x)}{q(x)}\right)$, is a measure of the similarity between the distributions. Thus, for two policies π and π', the Kullback-Leibler divergence becomes $D_{KL}(\pi||\pi') = \sum_{\mathbf{a}}\pi(\mathbf{a}|\mathbf{s})\log\left(\frac{\pi(\mathbf{a}|\mathbf{s})}{\pi'(\mathbf{a}|\mathbf{s})}\right)$.

Since the theoretical value for C is quite large when $\gamma \approx 1$, the practical TRPO algorithm uses a KL constraint rather than a KL penalty. The KL constraint is called the "trust region" since the new policy cannot diverge too much from the old policy. The final optimization problem becomes

$$\underset{\boldsymbol{\theta}}{\operatorname{argmax}}\, \mathcal{L}^{\text{TRPO}}(\boldsymbol{\theta}) \quad \text{s.t.} \quad \mathbb{E}_{\mathbf{s}\sim d^{\pi_{\boldsymbol{\theta}'}}}\left[D_{KL}(\pi_{\boldsymbol{\theta}}||\pi_{\boldsymbol{\theta}'})[s]\right] \leq \delta, \tag{2.30}$$

where δ is a small number that defines the trust region.

2.2.4 Proximal Policy Optimization

The optimization problem in (2.30) is elegantly stated, but is not trivial to solve in practice. Instead, the practical TRPO algorithm solves an approximated version of (2.30), where the objective and the constraint are linearly and quadratically approximated, respectively. Even though this approximation can be efficiently solved using the conjugate gradient algorithm, see [32], there is room for improvements.

Proposed in 2017, the proximal policy optimization (PPO) algorithm does not require any higher-order approximations [27]. The algorithm approximately enforces the KL constraint in TRPO by utilizing the so-called clip function, which given an interval clips arguments outside the interval to the interval edges. More specifically, PPO replaces the expression $h_t(\boldsymbol{\theta})A^{\pi_{\boldsymbol{\theta}'}}(\mathbf{s}_t, \mathbf{a}_t)$ in (2.28) with

$$\mathcal{L}_t^{\text{CLIP}}(\boldsymbol{\theta}) := \min\left(h_t(\boldsymbol{\theta})A^{\pi_{\boldsymbol{\theta}'}}(\mathbf{s}_t, \mathbf{a}_t), \text{clip}(h_t(\boldsymbol{\theta}), 1-\epsilon, 1+\epsilon)A^{\pi_{\boldsymbol{\theta}'}}(\mathbf{s}_t, \mathbf{a}_t)\right), \tag{2.31}$$

where ϵ is a parameter. The first term inside the minimum function is simply the same term that is included in the TRPO objective. The other term, $\text{clip}(h_t(\boldsymbol{\theta}), 1-\epsilon, 1+\epsilon)A_t^{\pi_{\boldsymbol{\theta}'}}$, includes the clip function, which ensures that $h_t(\boldsymbol{\theta})$ stays within the range $[1-\epsilon, 1+\epsilon]$. By taking the minimum of these terms, the PPO objective becomes a lower bound on the TRPO objective.

Now, in order to obtain a practical algorithm, we need to estimate the advantage function $A_t^{\pi_{\boldsymbol{\theta}'}} = A^{\pi_{\boldsymbol{\theta}'}}(\mathbf{s}_t, \mathbf{a}_t)$ in some way. In fact, it is enough to execute the policy for T' time steps, where T' can be much less than the actual episode length, and use these samples to update the policy. Taking an actor-critic approach and estimating the value function[4], the advantage function can be estimated by

$$\hat{A}_t^{\pi_{\boldsymbol{\theta}'}} = -V_{\boldsymbol{\theta}'}(\mathbf{s}_t) + r_t + \gamma r_{t+1} + \ldots + \gamma^{T'-t-1} r_{T'-1} + \gamma^{T'-t} V_{\boldsymbol{\theta}'}(\mathbf{s}_{T'}), \tag{2.32}$$

where r_t is the reward at time t [33]. This estimation can be generalized. Using the notation

$$\delta_t = r_t + \gamma V_{\boldsymbol{\theta}'}(\mathbf{s}_{t+1}) - V_{\boldsymbol{\theta}'}(\mathbf{s}_t), \tag{2.33}$$

we rewrite (2.32) as

$$\hat{A}_t^{\pi_{\boldsymbol{\theta}'}} = \sum_{k=0}^{T'-t-1} \gamma^k \delta_{t+k}. \tag{2.34}$$

[4]For simplicity's sake, we use the same parameters for both the actor (the policy) and the critic (the value function).

Schulman et al. [34] define the so-called generalized advantage estimation (GAE) as

$$\hat{A}_t^{\mathrm{GAE}} = \sum_{k=0}^{\infty} (\lambda\gamma)^k \delta_{t+k}, \tag{2.35}$$

where λ is a parameter between 0 and 1. The GAE is in fact a weighted average of the estimatation (2.34) for $T' - t \in \{1, 2, 3, \ldots\}$. Using a truncated version of the GAE we obtain the advantage estimation that is used in PPO, namely

$$\hat{A}_t^{\pi_{\boldsymbol{\theta}'}} = \delta_t + (\gamma\lambda)\delta_{t+1} + \ldots + \ldots + (\gamma\lambda)^{T'-t-1}\delta_{T'-1}. \tag{2.36}$$

Note that if the parameter λ equals 1, the generalized form of the estimation reduces to (2.32).

Usually, the objective function in PPO contains an additional entropy bonus that encourages the agent to explore different policies during the training. Moreover, if a shared network is used for both the policy and the value function, the objective function should also include a value function error term, e.g., a square error loss

$$\mathcal{L}_t^{\mathrm{VF}}(\boldsymbol{\theta}) := (V_{\boldsymbol{\theta}'}(\mathbf{s}_t) - V_t^{\mathrm{target}})^2, \tag{2.37}$$

where the target can be estimated by the return G_t. We finally obtain

$$\mathcal{L}^{\mathrm{PPO}}(\boldsymbol{\theta}) = \mathbb{E}_{\boldsymbol{\tau}\sim\pi_{\boldsymbol{\theta}'}}\left[\sum_{t=0}^{T'} \mathcal{L}_t^{\mathrm{CLIP}}(\boldsymbol{\theta}) - c_1\mathcal{L}_t^{\mathrm{VF}}(\boldsymbol{\theta}) + c_2 S[\pi_{\boldsymbol{\theta}}](\mathbf{s}_t)\right], \tag{2.38}$$

where S is an entropy bonus, and c_1, c_2 are hyperparameters. As entropy bonus, we can use the causal entropy defined in (2.53) in Section 2.3.5. The algorithm is summarized in Algorithm 1.

Algorithm 1 PPO, Actor-Critic Style

for iteration $= 1, 2, \ldots$ **do**
  for actor $= 1, 2, \ldots, N$ **do**
    Run policy $\pi_{\boldsymbol{\theta}'}$ for T' time steps
    Compute advantage estimates $\hat{A}_1, \ldots, \hat{A}_{T'}$
  end for
  Optimize objective function $\mathcal{L}^{\mathrm{PPO}}(\boldsymbol{\theta})$ w.r.t. $\boldsymbol{\theta}$ over K epochs and mini-batch size $M \leq NT$
  $\boldsymbol{\theta}' \leftarrow \boldsymbol{\theta}$
end for

2.3 Imitation Learning

In our description of RL, the reward function is assumed to be known. In fact, the reward function must be manually designed for the particular task. This is not trivial for all problems. Imitation learning is useful when it is harder to manually design a reward function than to let an expert demonstrate the desired behavior. The

imitation learning agent aims to behave optimally by imitating the expert demonstrations. There are two subfields of imitation learning – behavioral cloning (BC) and inverse reinforcement learning (IRL).

In this section, we start by presenting BC before defining the problem of IRL as a Markov decision process. Thereafter, we introduce three IRL methods, namely feature expectation matching, maximum entropy IRL and maximum causal IRL. After that we present GAIL and AIRL. Both GAIL and AIRL are based on maximum causal entropy IRL which is an extension to maximum entropy IRL. Maximum entropy IRL is based on feature expectation matching.

The thorough walk-through of the field of IRL is mostly of theoretical interest and is intended to give the reader a possibility to fully understand the foundations of GAIL and AIRL. Although the section can be read from beginning to end, we recommend readers who are already familiar with IRL concepts to focus on Section 2.3.6 and 2.3.7, which concern GAIL and AIRL.

2.3.1 Behavioral Cloning

Behavioral cloning is the simplest form of imitation learning where the behavior of the experts is learned through supervised learning [35]. The aim of behavioral cloning is to learn a deterministic policy π which maps states $\mathbf{s}$ to actions $\mathbf{a}$ so that

$$\mathbf{a}_t = \pi(\mathbf{s}_t) \tag{2.39}$$

for all state-action pairs in the training set $\mathcal{D} = (\mathbf{s}_t, \mathbf{a}_t)_{t=0}^{N}$. The state-action pairs are assumed to be independent and identically distributed (i.i.d.). The policy can be approximated by a neural network trained on $\mathcal{D}$. More specifically, the state and action in each state-action pair is used as input and target, respectively, so that the policy is learned by minimizing a loss function. In some cases, BC shows good results but the assumption about independent state-action pairs is in general too naive to learn a complex behavior.

2.3.2 Problem Statement of IRL

Instead of learning a mapping between demonstrated states and actions through supervised learning a more general solution is to learn the underlying reward function by using IRL [36]. The reward function defines the objective of the experts. Thus, learning the reward function can be interpreted as learning the experts' objective for the desired task. The reward function can thereafter be used to estimate the expert policy using RL.

An IRL problem is usually formulated using a Markov decision process, defined as $\mathcal{M} := \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \gamma, R \rangle$, where $\mathcal{S}$ is a finite set of states, $\mathcal{A}$ is a finite set of actions, $\mathcal{P}$ is the state-transition probability function, $\gamma \in [0, 1)$ is a discount factor and $R : \mathcal{S} \times \mathcal{A} \mapsto \mathbb{R}$ is the reward function. This setting is identical to the RL problem formulation in Section 2.2.1. However, the reward function in an IRL problem is

unknown and we denote an MDP without known reward function as $\mathcal{M}\backslash_R$.

In IRL the experts are assumed to behave according to an underlying expert policy π^E and the agent follows a stochastic policy $\pi_{\boldsymbol{\theta}}$ parameterized by $\boldsymbol{\theta}$. The agent observes demonstrations of expert behavior in the form of trajectories. Given a training set $\mathcal{D} = (\boldsymbol{\tau}_i^E)_{i=1}^N$, where each trajectory $\boldsymbol{\tau}_i^E$ is an expert demonstration, the objective of IRL is to find an estimate $\hat{R}$ of the reward function that best explains the observed behavior of the experts. In addition, the agent policy should not outperform the expert policy in terms of the expected cumulative discounted reward, i.e.,

$$\mathbb{E}_{\boldsymbol{\tau}\sim\pi^E}\left[\sum_{t=0}^{\infty}\gamma^t\hat{R}\left(\mathbf{s}_t,\mathbf{a}_t\right)\right] \geq \mathbb{E}_{\boldsymbol{\tau}\sim\pi_{\boldsymbol{\theta}}}\left[\sum_{t=0}^{\infty}\gamma^t\hat{R}\left(\mathbf{s}_t,\mathbf{a}_t\right)\right]. \tag{2.40}$$

In IRL, the estimated reward $\hat{R}$ is usually found through an iterative learning process which alternates between estimating the reward function and solving an RL problem. In other words, $\mathcal{M}\backslash_R$ is first solved to find an estimate $\hat{R}$ of the true reward function. Then, $\mathcal{M}$ is solved using RL for the current reward estimate. This is done iteratively until the agent policy that best explains the expert demonstrations is found. However, the problem statement is "ill-posed" which means that many optimal policies can explain the set of expert demonstrations and one optimal policy can be optimal for various reward estimates [37].

2.3.3 Feature Expectation Matching

In feature expectation matching, the unknown reward function is expressed as a linearly-weighted combination of reward features $\boldsymbol{\phi} : \mathcal{S} \mapsto \mathbb{R}^n$ so that

$$R_{\mathbf{w}}(\mathbf{s},\mathbf{a}) = w_1\phi_1(\mathbf{s},\mathbf{a}) + w_2\phi_2(\mathbf{s},\mathbf{a}) + \ldots + w_n\phi_n(\mathbf{s},\mathbf{a}) = \mathbf{w}^\top\boldsymbol{\phi}(\mathbf{s},\mathbf{a}) \tag{2.41}$$

where $\mathbf{w} \in \mathbb{R}^n$ is the reward weight vector. Using $R_{\mathbf{w}}$, the expected cumulative discounted reward for a given policy π can be rewritten as

$$\begin{aligned}\mathbb{E}_\pi\left[\sum_{t=0}^{\infty}\gamma^t R_{\mathbf{w}}\left(\mathbf{s}_t,\mathbf{a}_t\right)\right] &= \mathbb{E}_\pi\left[\sum_{t=0}^{\infty}\gamma^t\mathbf{w}^\top\boldsymbol{\phi}\left(\mathbf{s}_t,\mathbf{a}_t\right)\right] \\ &= \mathbf{w}^\top\mathbb{E}_\pi\left[\sum_{t=0}^{\infty}\gamma^t\boldsymbol{\phi}\left(\mathbf{s}_t,\mathbf{a}_t\right)\right] \\ &= \mathbf{w}^\top\boldsymbol{\mu}(\pi),\end{aligned} \tag{2.42}$$

where $\boldsymbol{\mu}(\pi)$ is the expected cumulative discounted feature values or *feature expectations*. The feature expectations are in many IRL methods used as an alternative representation of the policy and they can be used to calculate the similarity between the current policy estimate and the expert policy.

Given that the experts solve $\mathcal{M}\backslash_R$ with $\hat{R} = R_{\mathbf{w}}$, Abbeel et al. [38] show that the feature expectations of the expert policy and the agent policy must match in order to find the true reward R. The IRL problem is reduced to finding a policy $\pi_{\boldsymbol{\theta}}$ that satisfies

$$||\boldsymbol{\mu}(\pi^E) - \boldsymbol{\mu}(\pi_{\boldsymbol{\theta}})||_2 \leq \epsilon \tag{2.43}$$

for a given threshold ϵ. The feature expectations of the experts can be estimated according to

$$\hat{\boldsymbol{\mu}}(\pi^E) = \frac{1}{N}\sum_{i=1}^{N}\sum_{t=0}^{\infty}\gamma^t\boldsymbol{\phi}\left(\mathbf{s}_t^i, \mathbf{a}_t^i\right) \tag{2.44}$$

where $\mathbf{s}_t^i$ and $\mathbf{a}_t^i$ is a state and an action, respectively, from the ith demonstration at time step t.

2.3.4 Maximum Entropy IRL

Due to the ill-posedness of the IRL problem, matching feature expectations is sufficient to guarantee that $\pi_{\boldsymbol{\theta}}$ performs as well as π^E but a unique solution cannot be guaranteed. However, there exists a unique solution that maximizes the entropy. By introducing an additional constraint to the IRL problem, Ziebart et al. [37] propose maximum entropy IRL to find the unique solution.

In maximum entropy IRL, the policy $\pi_{\boldsymbol{\theta}}$ is represented by a distribution over trajectories $p(\boldsymbol{\tau})$ where $\boldsymbol{\tau} \sim \pi_{\boldsymbol{\theta}}$. The principle of maximum entropy, proposed by E.T. Jaynes in 1957, states that the distribution which is most appropriate to model a given data set is the one with the highest entropy [39]. To apply this principal to IRL, Ziebart et al. state that among the distributions over trajectories that match the feature expectations of the expert, the distribution that maximizes the entropy should be chosen. Using this, the objective of maximum entropy can be defined as learning the policy $\pi_{\boldsymbol{\theta}}$ that satisfies

$$\underset{\boldsymbol{\theta}}{\operatorname{argmax}}\, H(p) = \underset{\boldsymbol{\theta}}{\operatorname{argmax}} \sum_{\boldsymbol{\tau}\sim\pi_{\boldsymbol{\theta}}} p(\boldsymbol{\tau}) \ln \frac{1}{p(\boldsymbol{\tau})} \tag{2.45}$$

subject to

$$||\boldsymbol{\mu}(\pi^E) - \boldsymbol{\mu}(\pi_{\boldsymbol{\theta}})||_2 \leq \epsilon, \tag{2.46}$$

$$\sum_{\boldsymbol{\tau}\sim\pi_{\boldsymbol{\theta}}} p(\boldsymbol{\tau}) = 1, \tag{2.47}$$

$$p(\boldsymbol{\tau}) > 0 \quad \forall \boldsymbol{\tau}. \tag{2.48}$$

The distributions that maximize the entropy H belong to the exponential family of distributions [40]. Therefore, the conditional probability of the distribution of maximum entropy can be written as

$$p(\boldsymbol{\tau}|\mathbf{w}) = \frac{1}{Z(\mathbf{w})}\exp\left(\mathbf{w}^\top\boldsymbol{\phi}(\boldsymbol{\tau})\right) = \frac{1}{Z(\mathbf{w})}\exp(R_{\mathbf{w}}(\boldsymbol{\tau})), \tag{2.49}$$

where $Z(\mathbf{w}) = \sum_\tau \exp\left(\mathbf{w}^\top\boldsymbol{\phi}(\boldsymbol{\tau})\right)$ is the *partition function*, see [37].

In general, transitions between states in an MDP are non-deterministic and follow the state-transition distribution function $\mathcal{P}$. The transition probabilities are not considered in (2.49) and the equation thus only holds for deterministic environments. When adapting the solution to stochastic environments the transition probabilities

need to be taken into account. Therefore, we redefine the conditional probability in (2.49) as

$$p(\boldsymbol{\tau}|\mathbf{w}) = \frac{\exp\left(\mathbf{w}^\top \boldsymbol{\phi}(\boldsymbol{\tau})\right)}{Z(\mathbf{w})} \prod_{\mathbf{s}_{t+1},\, \mathbf{a}_t,\, \mathbf{s}_t \in \boldsymbol{\tau}} \mathcal{P}\left(\mathbf{s}_{t+1}|\mathbf{a}_t, \mathbf{s}_t\right) \tag{2.50}$$

where $\mathcal{P}$, defined in (2.14), denotes the dynamics of the underlying MDP. Using this, Ziebart et al. show that the reward becomes

$$\tilde{R}_{\mathbf{w}}(\boldsymbol{\tau}) = \mathbf{w}^\top \boldsymbol{\phi}(\boldsymbol{\tau}) + \sum_{\mathbf{s}_{t+1},\, \mathbf{a}_t,\, \mathbf{s}_t \in \boldsymbol{\tau}} \log \mathcal{P}\left(\mathbf{s}_{t+1}|\mathbf{a}_t, \mathbf{s}_t\right), \tag{2.51}$$

where the additional term is accounting for the non-deterministic nature of the environment.

The problem of finding the reward weights $\mathbf{w}$ can now be formulated as a maximum likelihood problem, i.e.,

$$\mathbf{w}^* = \underset{\mathbf{w}}{\operatorname{argmax}} \sum_{\boldsymbol{\tau} \in \mathcal{D}} \ln p\left(\boldsymbol{\tau}|\mathbf{w}\right). \tag{2.52}$$

The sum in (2.52) is convex and can thus be maximized by gradient-based methods. The maximum entropy IRL method performs well in many MDP problems but assuming that the state transition probabilities are known is not realistic.

2.3.5 Maximum Causal Entropy IRL

In [41], Ziebart et al. extend their previous work on maximum entropy IRL to avoid dependency on the state transition probabilities. They propose the method of maximum causal entropy IRL. The concept of maximal causal entropy is important in maximum causal entropy IRL but we refrain from presenting details about this concept. We refer the interested reader to [41].

Maximum causal entropy IRL assumes causal action choices which means that each action taken has to be independent of any future states in trajectory $\boldsymbol{\tau}$. By using this, Ziebart et al. show that the stochasticity of the environment can be incorporated into the reward estimation. The objective of maximum causal entropy IRL is to find the policy $\pi_{\boldsymbol{\theta}}$ that maximizes the causal entropy $H\left(\mathbf{a}_{1:T} \| \mathbf{s}_{1:T}\right)$ defined as

$$\begin{aligned} H\left(\mathbf{a}_{1:T} \| \mathbf{s}_{1:T}\right) &= \sum_{t=1}^{T} H\left(\mathbf{a}_t | \mathbf{a}_{1:t-1}, \mathbf{s}_{1:t}\right) \\ &= -\sum_{t=1}^{T} \sum_{\mathbf{a}_{1:t},\, \mathbf{s}_{1:t}} p\left(\mathbf{a}_{1:t}, \mathbf{s}_{1:t}\right) \ln\left(\pi_{\boldsymbol{\theta}}\left(\mathbf{a}_t | \mathbf{a}_{1:t-1}, \mathbf{s}_{1:t}\right)\right). \end{aligned} \tag{2.53}$$

Here, $p\left(\mathbf{a}_{1:t}, \mathbf{s}_{1:t}\right)$ is the joint distribution over all states and actions up to time t for the trajectory $\boldsymbol{\tau} \sim \pi_{\boldsymbol{\theta}}$ and $H\left(\mathbf{a}_t | \mathbf{a}_{1:t-1}, \mathbf{s}_{1:t}\right)$ is the conditional entropy until time step t. Assuming that the process is Markovian, i.e., assuming that the probability of the current action $\mathbf{a}_t$ only depends on the last state $\mathbf{s}_t$, we can reduce $\pi_\theta(\mathbf{a}_t|\mathbf{s}_{1:t}, \mathbf{a}_{1:t-1})$ to $\pi_{\boldsymbol{\theta}}(\mathbf{a}_t|\mathbf{s}_t)$.

The objective of maximum causal entropy IRL can be formulated as

$$\underset{\mathbf{s},\,\mathbf{a}\sim\pi_{\boldsymbol{\theta}}}{\operatorname{argmax}} H\left(\mathbf{a}_{1:T}||\mathbf{s}_{1:T}\right) = \underset{\boldsymbol{\theta}}{\operatorname{argmax}} H(\pi_{\boldsymbol{\theta}}) \tag{2.54}$$

subject to

$$||\boldsymbol{\mu}(\pi^E) - \boldsymbol{\mu}(\pi_{\boldsymbol{\theta}})||_2 \leq \epsilon, \tag{2.55}$$

$$\sum_{\mathbf{s},\,\mathbf{a}\sim\pi_{\boldsymbol{\theta}}} \pi_{\boldsymbol{\theta}}(\mathbf{a}|\mathbf{s}) = 1, \tag{2.56}$$

$$\pi_{\boldsymbol{\theta}}(\mathbf{a}|\mathbf{s}) \geq 0. \tag{2.57}$$

The objective in maximum causal entropy IRL can be optimized without knowledge of the transition probabilites between states through dynamical programming, see [41].

To simplify the notation of the objective of maximum causal entropy IRL it is commonly reformulated as

$$\underset{R\in\mathcal{R}}{\operatorname{argmax}} \left(\underset{\pi_{\boldsymbol{\theta}}\in\Pi}{\operatorname{argmin}} \left(-H\left(\pi_{\boldsymbol{\theta}}\right) - \mathbb{E}_{\pi_{\boldsymbol{\theta}}}[R(\mathbf{s},\mathbf{a})]\right) - \mathbb{E}_{\pi^E}[R(\mathbf{s},\mathbf{a})] \right) \tag{2.58}$$

where $\mathcal{R}$ is a family of reward functions and Π is a family of policies. The objective in (2.58) states that the problem of maximum causal IRL reduces to finding the reward function which assigns high reward to the expert policy and low rewards to all other policies. At the same time, this reward function should maximize the causal entropy and the expected reward of the corresponding policy. Many IRL methods are based on (2.58) so that the problem of finding the reward weights narrows down to a maximum likelihood problem [5, 6, 42].

2.3.6 Generative Adversarial Imitation Learning

In 2016, Ho et al. [5] proposed *generative adversarial imitation learning* (GAIL) where a generative adversarial model structure is used to solve the IRL problem. Recall from Section 2.1.2 that the idea behind GANs is to have a *generator* network, G, that generates samples that should follow the true data distribution and a *discriminator* network, D, that is trained to separate samples from true data from samples generated from G.

In GAIL, the generator network represents a stochastic policy $\pi_{\boldsymbol{\theta}}(\mathbf{a}|\mathbf{s})$ and the discriminator is trained to separate expert state-action pairs from state-action pairs generated by the policy. The output from the discriminator $D_{\mathbf{w}}(\mathbf{s},\mathbf{a}) \in [0,1]$ is the probability that a given state-action pair comes from the generator. GAIL uses the output signal from the discriminator as a reward to update the generator using a policy gradient method. However, the discriminator signal in GAIL does *not* represent an estimate of the true reward.

Abbeel et al. [43] show that in the same way that an IRL problem can be written as a problem of matching feature expectations, the IRL problem can also be rewritten

as a problem of matching *occupancy measures*. The occupancy measure ρ of $\pi_{\boldsymbol{\theta}}(\mathbf{a}|\mathbf{s})$ is defined as

$$\rho_\pi(\mathbf{s}, \mathbf{a}) = \pi_{\boldsymbol{\theta}}(\mathbf{a}|\mathbf{s}) \sum_{t=0}^{\infty} \gamma^t \mathbb{P}(\mathbf{s}_t = \mathbf{s}|\pi_{\boldsymbol{\theta}}). \tag{2.59}$$

Ho et al. use occupancy measure matching together with Ziebart's maximum causal entropy IRL formulation to propose a new formulation for the problem of IRL. The new objective is to learn the policy which minimizes the distance between the occupancy measures of the agent and the expert with a regularizing term for the causal entropy. The optimization problem reads:

$$\min_{\pi_{\boldsymbol{\theta}} \in \Pi} \quad \mathbf{d}\left(\rho_{\pi_{\boldsymbol{\theta}}}(\mathbf{s}, \mathbf{a}), \rho_{\pi^E}(\mathbf{s}, \mathbf{a})\right) - \lambda H(\pi_{\boldsymbol{\theta}}). \tag{2.60}$$

Here $\mathbf{d}(.)$ is a distance function between the occupancy measure of the agent and the expert, λ is a weighting factor and H is the causal entropy of policy $\pi_{\boldsymbol{\theta}}$. Ho et al. propose $\mathbf{d}$ to be the Jensen-Shannon divergence between the two occupancy measures in an adversarial context. That is, they propose a GAN structure that minimizes the JS divergence to find the Nash equilibrium of the objective function. The objective in GAIL is defined as

$$\min_G \max_D \mathbb{E}_{\pi_{\boldsymbol{\theta}}}[\log(D_{\mathbf{w}}(\mathbf{s}, \mathbf{a}))] + \mathbb{E}_{\pi^E}[\log(1 - D_{\mathbf{w}}(\mathbf{s}, \mathbf{a}))] - \lambda H(\pi_{\boldsymbol{\theta}}). \tag{2.61}$$

The discriminator is trained by maximizing (2.61) with respect to $\mathbf{w}$ using an Adam gradient step whereas the generator is trained by minimizing (2.61) with respect to $\boldsymbol{\theta}$ using a policy gradient update step which simultaneously maximizes the entropy. The iterative training process of GAIL makes the discriminator better at distinguishing agent state-action pairs from expert state-action pairs while the policy becomes better at generating state-action pairs that are more expert-like. GAIL is summarized in Algorithm 2.

Algorithm 2 Generative Adversarial Imitation Learning

Input Expert trajectories $\boldsymbol{\tau}^E \sim \pi^E$, initial G and D parameters $\boldsymbol{\theta}_0$ and $\mathbf{w}_0$
for $i = 0, 1, 2 \ldots$ **do**
  Sample trajectories $\boldsymbol{\tau}_i \sim \pi_{\boldsymbol{\theta}_i}$
  Calculate $\mathbf{w}_{i+1} \leftarrow \mathbf{w}_i$ using the gradient
    $\mathbb{E}_{\tau_i}\left[\nabla_w \log\left(D_w(\mathbf{s}, \mathbf{a})\right)\right] + \mathbb{E}_{\tau_E}\left[\nabla_w \log\left(1 - D_w(\mathbf{s}, \mathbf{a})\right)\right]$
  Calculate $\boldsymbol{\theta}_{i+1} \leftarrow \boldsymbol{\theta}_i$ using TRPO/PPO with loss function $\log\left(D_{\mathbf{w}_{i+1}}(\mathbf{s}, \mathbf{a})\right)$
end for
Return $\hat{\pi}^E$

2.3.7 Adversarial Inverse Reinforcement Learning

Finn et al. [44] show that it is possible to learn the true reward function in an adversarial context by modeling the discriminator as the reward function. Fu et al. [6] present an implementation of this theoretical concept, called adversarial inverse reinforce learning (AIRL). The algorithm is summarized in Algorithm 3.

Algorithm 3 Adversarial Inverse Reinforcement Learning

Input Expert trajectories $\boldsymbol{\tau}^E \sim \pi^E$, initial G and D parameters $\boldsymbol{\theta}_0$, $\mathbf{w}_0$ and $\boldsymbol{\phi}_0$
for $i = 0, 1, 2 \ldots$ **do**
  Sample trajectories $\boldsymbol{\tau}_i \sim \pi_{\boldsymbol{\theta}_i}$
  Train $D_{\mathbf{w},\boldsymbol{\phi}}$ via binary logistic regression to classify $\boldsymbol{\tau}_i^E$ from $\boldsymbol{\tau}_i$
  Update reward $R_{\mathbf{w}_{i+1},\boldsymbol{\phi}_{i+1}} \leftarrow \log D_{\mathbf{w},\phi}(\mathbf{s},\mathbf{a},\mathbf{s}') - \log(1 - D_{\mathbf{w},\phi}(\mathbf{s},\mathbf{a},\mathbf{s}'))$
  Calculate $\pi_{\boldsymbol{\theta}_{i+1}}$ w.r.t. $R_{\mathbf{w}_{i+1},\boldsymbol{\phi}_{i+1}}$ using any policy optimization method
end for
Return $\hat{\pi}_{\boldsymbol{\theta}}$ and $\hat{R}^E_{\mathbf{w},\boldsymbol{\phi}}$

The discriminator $D_{\mathbf{w},\boldsymbol{\phi}}(\mathbf{s},\mathbf{a},\mathbf{s}')$ is trained as a binary classifier to separate demonstrated samples from generated samples. The policy $\pi_{\boldsymbol{\theta}}$ is on the other hand trained to optimize

$$\underset{\mathbf{w}}{\operatorname{argmax}} \log\left(D_{\mathbf{w},\boldsymbol{\phi}}(\mathbf{s},\mathbf{a},\mathbf{s}') - \log\left(1 - D_{\mathbf{w},\boldsymbol{\phi}}(\mathbf{s},\mathbf{a},\mathbf{s}')\right)\right) \tag{2.62}$$

by using any policy optimization method which simultaneously maximizes the entropy.

In order to recover the true reward function, the discriminator in AIRL consists of two MLPs parameterized by $\mathbf{w}$ and $\boldsymbol{\phi}$, respectively. A large difference between GAIL and AIRL, apart from AIRL learning the true reward function, is that the discriminator $D_{\mathbf{w},\boldsymbol{\phi}}$ in AIRL is explicitly defined, i.e., it is not just the output from the discriminator network as it is in GAIL. The discriminator in AIRL is defined as

$$D_{\mathbf{w},\boldsymbol{\phi}}(\mathbf{s},\mathbf{a},\mathbf{s}') = \frac{\exp\left(f_{\mathbf{w},\boldsymbol{\phi}}(\mathbf{s},\mathbf{a},\mathbf{s}')\right)}{\exp\left(f_{\mathbf{w},\boldsymbol{\phi}}(\mathbf{s},\mathbf{a},\mathbf{s}')\right) + \pi_{\boldsymbol{\theta}}(\mathbf{a}|\mathbf{s})} \tag{2.63}$$

where $f_{\mathbf{w},\boldsymbol{\phi}}(\mathbf{s},\mathbf{a},\mathbf{s}')$ is an estimation of the true reward function defined as

$$f_{\mathbf{w},\boldsymbol{\phi}}(\mathbf{s},\mathbf{a},\mathbf{s}') = g_{\mathbf{w}}(\mathbf{s}) + \gamma h_{\boldsymbol{\phi}}(\mathbf{s}') - h_{\boldsymbol{\phi}}(\mathbf{s}). \tag{2.64}$$

Here, $\gamma h_{\boldsymbol{\phi}}(\mathbf{s}') - h_{\boldsymbol{\phi}}(\mathbf{s})$ is a reward shaping term and $g_{\mathbf{w}}(\mathbf{s})$ is a *disentangled* reward term. A disentangled reward is a reward function which is robust to changes in the dynamics, i.e., it does not depend on changes in the state-transition probabilities. In other words, the true reward function can be learned by AIRL even if the algorithm is trained on non-stationary expert data. Non-stationary data changes over time due to, e.g., seasonal variation or a pandemic.

3
Method

The purpose of this thesis is to investigate how adversarial imitation learning algorithms, in particular GAIL and AIRL, can be used to learn the purchasing behavior of a customer population from a given data set – the expert data. In order to evaluate the performance of the algorithms, a method for comparing the similarity between generated data and expert data is needed. Essentially, this method could be any method for comparing time-series, but it is desirable to choose a method that not only measures the similarity, but also enables visualization of it. The given data set is introduced in Section 3.1 and a method for comparing time-series is developed in Section 3.2.

In Section 3.3, we formulate the problem of learning customer behaviors using a Markov decision process. In particular, we present three different state representations. Given a well-defined state, the agent should be able to learn to behave like a real customer. We start by proposing a basic state representation in Section 3.3.1. Then, in Section 3.3.2 and 3.3.3, we suggest two different extensions to the basic state: a one-hot vector that encodes individual customers and so-called customer-specific days.

In Section 3.4, we outline the standard way of training GAIL and AIRL. However, due to the diverse customer population, this procedure is not likely to be optimal in our case. Therefore, we propose matched mixture component trained GAIL (MMCT-GAIL) in Section 3.5.

3.1 Customer Data

In order to train GAIL and AIRL, we have access to synthetic customer data in the form of multivariate time-series. The data is generated from a simulation model that has been provided to us from Smartr in the form of a ”black box”. The simulation model generates customers of different sex and age. Sex and age affect a customer's behavior, but all generated customers have a nonzero probability of behaving in the same way.

The customer data represents a customer's purchasing behavior over time. Six products are available and the products could for instance be groceries or some other kind of items that are consumed on a daily basis. Figure 3.1 shows the purchasing behavior of a particular customer in the data set. Each day, the customer decides

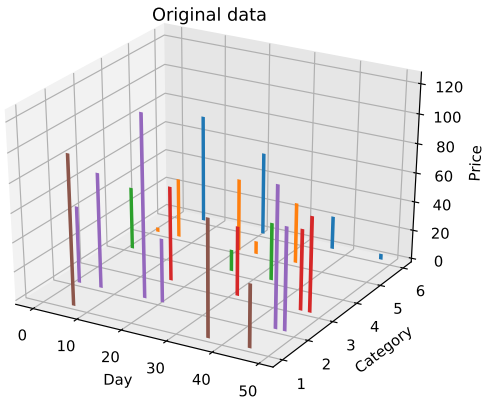


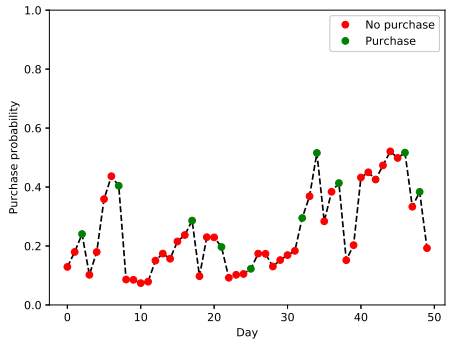
Figure 3.1: A sample from the original data set. The sample is a multivariate time-series that represents a particular customer's purchasing behavior over a time period of 50 days. The customer can buy items from six different product categories. The height of the bars shows how much money the customer has spent on a particular product on a particular day.

whether he or she needs to buy any of the products. As we can see in the figure, the customer does not necessarily have to buy all products at the same time but can instead choose to only buy one or a few products.

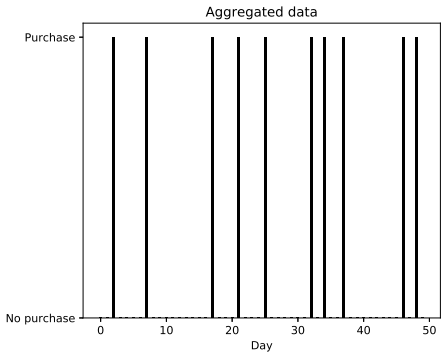
The idea of the underlying simulation model is that each customer has its own pantry where the levels of the products decrease as the customer randomly consumes the products according to some costumer specific distribution. The probability that the customer decides to go to the store naturally increases as the levels of the products in the pantry decrease. In Figure 3.2a, this probability for a specific customer is shown over a time period of 50 days. The red and green markers indicate whether the customer makes a purchase on a particular day.

In order to reduce the complexity of the problem, we mainly work with aggregated data, where all products have been merged together. That is, considering the original data as a matrix with six rows and n columns, where n is the number of days, we obtain the aggregated data by summing the matrix columns. We further reduce the complexity of the data by only considering binary actions, i.e., ignoring the actual purchase amounts. Figure 3.2b shows the aggregated form of the data in Figure 3.1. Note that the purchase events correspond to the green markers in Figure 3.2a.

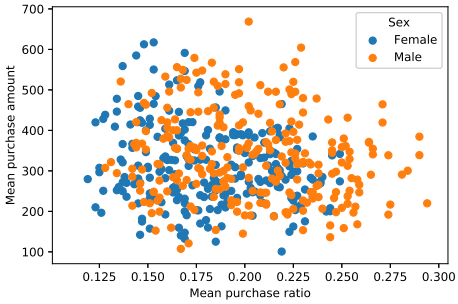
Figure 3.2c and 3.2d indicate how sex and age impact a customer's purchasing behavior. The figures show mean purchase amount against mean *purchase ratio* given sex and age, respectively, for a population of 500 customers. Each customer is represented by a sequence of 1000 days and the purchase ratio is simply the number of purchases divided by the length of the sequence. As we can see, men buy more frequently, and consume more money, than women. There is a similar trend for



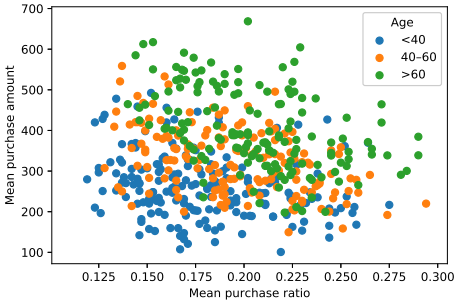
(a) Purchase probability vs time.



(b) Aggregated data.

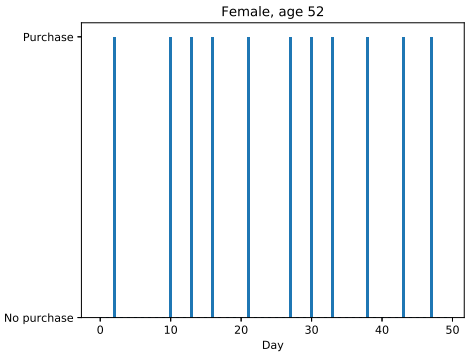


(c) Influence of gender.

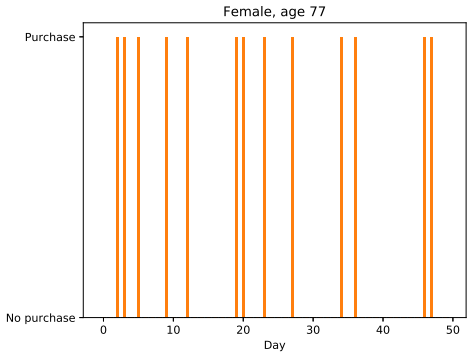


(d) Influence of age.

Figure 3.2: **(a)**: The purchase probability over a time period of 50 days for a particular customer in the underlying simulation model. Green and red markers indicate purchase and no purchase, respectively. **(b)**: The aggregated form of the original data in Figure 3.1. **(c)** and **(d)**: An indication of how gender and age impact a customer's purchasing power in the synthetic data set.



(a) A female customer of age 52.



(b) A female customer of age 77.

Figure 3.3: Samples from two different customers in the aggregated data set.

older people.

3.2 Comparing Customer Behaviors

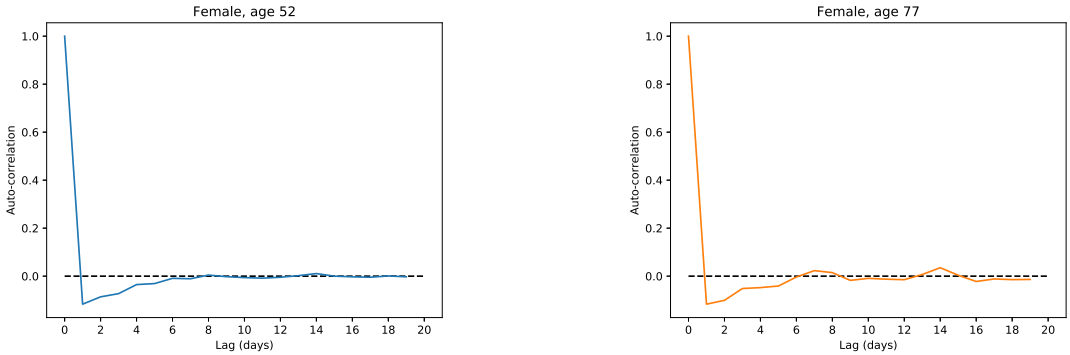
With the goal of learning realistic customer actions, a metric for measuring the similarity between sequences of actions is needed. Figure 3.3 shows the purchasing behavior of two different customers – a female of age 52 and a female of age 77 – over a period of 50 days. Apparently, these customers seem to behave somewhat differently – but how can this intuition be verified? A common method for comparing time-series is to extract features that characterize the time-series and then measure the distance between the feature vectors with a suitable distance function. However, it is not obvious which features that characterize a sequence of realistic actions.

In order to understand how the customers take future actions conditional on their previous actions, we look at the auto-correlation of the time-series. By defining a time-series as a sequence of observations $y_1, y_2, \ldots, y_N$, the lag k auto-correlation is given by

$$r_k = \frac{\sum_{i=1}^{N-k}(y_i - \overline{y})(y_{i+k} - \overline{y})}{\sum_{i=1}^{N}(y_i - \overline{y})^2}, \tag{3.1}$$

where $\overline{y}$ denotes the average value of the observations. In other words, the auto-correlation of a time-series is the correlation between the time-series and a shifted copy of itself. The lag specifies how many time steps the copy is shifted.

Figure 3.4 shows the auto-correlation for the behavior of the customers in Figure 3.3. By definition, the auto-correlation equals 1 when $k = 0$, i.e., the correlation is 1 for a time-series compared with itself. It is clear from the figure that the auto-correlation, for both customers, is negative for a lag between 1 and 6 days. In our case, a negative auto-correlation means that if a customer makes a purchase on a particular day, he or she is more likely *not* to make a purchase the following day, and vice versa. In Figure 3.4, we see that if the lag is greater than approximately one week, the auto-correlation approaches zero, i.e., the decision (to buy or not to



(a) A female customer of age 52.

(b) A female customer of age 77.

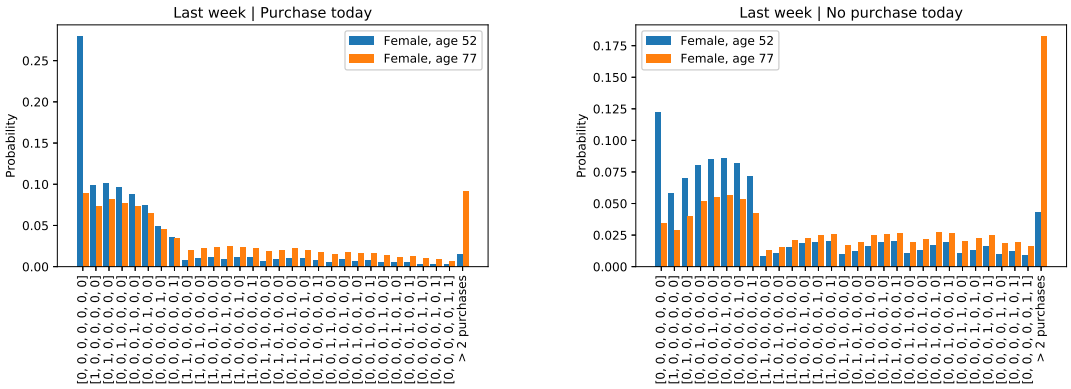
Figure 3.4: The auto-correlation of two time-series corresponding to two different customers in the data set.

buy anything) that the customer made more than a week ago has no influence on the decision that the customer makes today. In other words, the customer's daily decision only depends on the customer's decisions of the last week.

Given this information, we create the categorical distributions shown in Figure 3.5. In panel (a), we see the distribution of the last week's actions given that the customer *makes* a purchase today. In panel (b), we see the distribution of the last week's actions given that the customer *does not make* a purchase today. The distributions are created by first sampling long sequences of data. For each day in such a sample, depending on that day's action, the actions of the previous seven days are sorted into one of the bins that are shown in Figure 3.5. For example, the leftmost bar in each panel corresponds to the case when the customer has made no purchases during the last week. In order to reduce the complexity of the distributions, all cases when the customer has made more than two purchases during the last week are put into the rightmost bins.

As we can see in Figure 3.5, the behavior of the customers corresponds to distributions that clearly can be visually distinguished. Besides enabling a visual assessment of the similarity between the behaviors, it is now possible to explicitly measure the distance between the distributions, and thereby also the behaviors, by a metric for comparing distributions. We have found that the Wasserstein distance works well in our case. For details we refer to Section 2.1.3. The Wasserstein distance between the conditional distributions in Figure 3.5 is 0.017 and 0.014, respectively.

The distributions in Figure 3.5 are informative, but in order to automate the process of comparing different behaviors, it is practical to look at the unconditional distribution in Figure 3.6. This distribution contains all information in the distributions in Figure 3.5. In what follows, we mainly use the unconditional distribution when comparing behaviors. Figure 3.7 shows how the Wasserstein distance between the distributions of the customers in Figure 3.3 depends on the length of the sampled time-series. As we can see, in order to accurately measure the distance between the



(a) Last week given purchase today.

(b) Last week given no purchase today.

Figure 3.5: Conditional categorical distributions of customer behaviors for two different customers.

distributions, the sampled sequences need to be as long as 100000 days. For practical reasons, we sample only 10000 days when comparing behaviors which seems reasonable according to Figure 3.7.

3.3 MDP Representations

We formulate the problem of learning individual customer behaviors using a Markov decision process (MDP). The MDP is described by the tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \gamma, R \rangle$, where $\mathcal{S}$ is a finite set of states, $\mathcal{A}$ is a finite set of actions, $\mathcal{P}$ is the state-transition probability function, $\gamma \in [0, 1)$ is a discount factor, and $R : \mathcal{S} \times \mathcal{A} \mapsto \mathbb{R}$ is the reward function. In our case, the reward is unknown. In order to solve the MDP, we need to choose a state representation that captures a customer's purchase history. Optimally, the state should have the Markov property, meaning that the transition from state $\mathbf{s}_t$ to state $\mathbf{s}_{t+1}$ only depends on $\mathbf{s}_t$. In the following sections, we introduce three different state representations.

3.3.1 Basic Representation

A customer's daily decision to either make a purchase or not make a purchase depends on his or her purchase history. A customer that made a purchase yesterday is less likely to make a purchase today. We propose a basic state representation that includes the customer's purchase history from the last N days. The idea is that such a state should provide the agent with enough information for taking realistic actions. The larger N is, the more of the customer's statistical behavior is included in the state.

In order to train the algorithms, we must convert the time-series in the expert data set into state-action pairs. In practice, since we use synthetic data, the time-series can be infinitely long. However, we are interested in reducing the number of state-

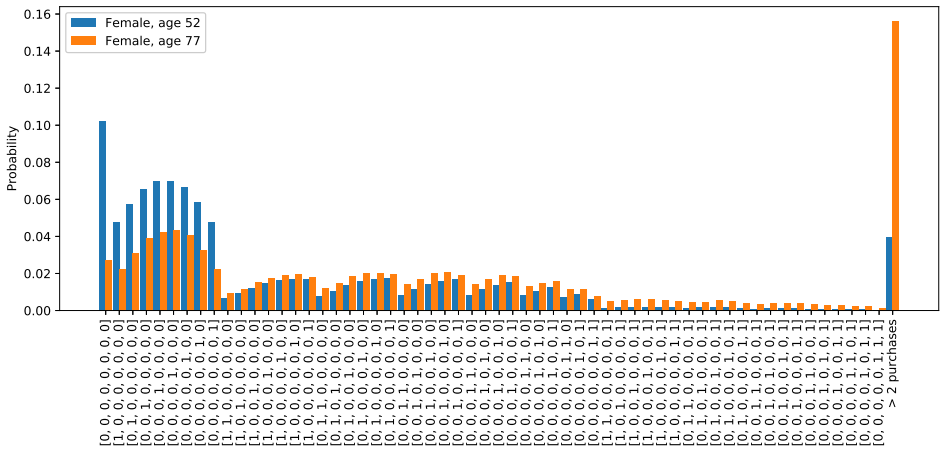


Figure 3.6: The unconditional distribution corresponding to the conditional distributions in Figure 3.5.

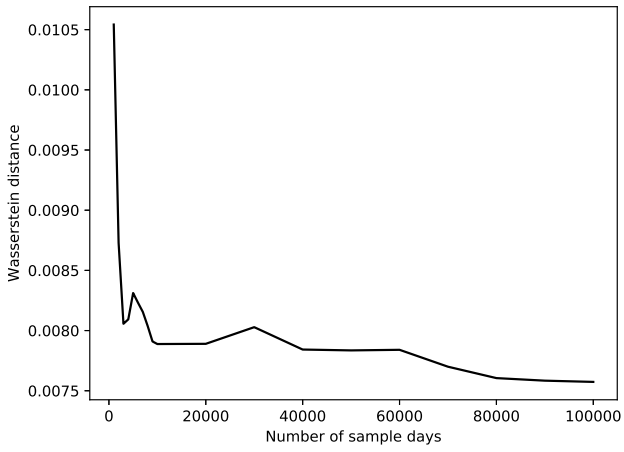


Figure 3.7: The Wasserstein distance between the distributions for the different customers in Figure 3.6 as a function of the length of the time-series used for generating the distributions.

action pairs that are used for training the algorithms. Assuming that we limit each expert time-series to a length of M days, we form $M - N$ state-action pairs for each expert. Using $M = 50$ and $N = 40$ as an example, the time-series in Figure 3.3a would be converted into the following state-action pairs:

$$\begin{aligned}
\mathbf{s}_0 &= \Big[0,0,1,0,0,0,0,0,0,0,\ldots,1,0,0,1,0,0,0,0,1,0\Big], \quad a_0 = 0 \\
\mathbf{s}_1 &= \Big[0,1,0,0,0,0,0,0,0,1,\ldots,0,0,1,0,0,0,0,1,0,0\Big], \quad a_1 = 0 \\
\mathbf{s}_2 &= \Big[1,0,0,0,0,0,0,0,1,0,\ldots,0,1,0,0,0,0,1,0,0,0\Big], \quad a_2 = 0 \\
&\vdots \\
\mathbf{s}_7 &= \Big[0,0,0,1,0,0,1,0,0,1,\ldots,0,1,0,0,0,0,1,0,0,0\Big], \quad a_7 = 1 \\
\mathbf{s}_8 &= \Big[0,0,1,0,0,1,0,0,1,0,\ldots,1,0,0,0,0,1,0,0,0,1\Big], \quad a_8 = 0 \\
\mathbf{s}_9 &= \Big[0,1,0,0,1,0,0,1,0,0,\ldots,0,0,0,0,1,0,0,0,1,0\Big], \quad a_9 = 0.
\end{aligned} \tag{3.2}$$

Here, every state consists of $N = 40$ elements.

When training the algorithms, the agent is initialized into a state from a particular expert's trajectory. By executing the agent's policy, i.e., sampling from the agent's policy given an initial state, a trajectory of state-action pairs is collected, which optimally should be similar to the expert's trajectory. As an example, let us initialize the agent with a state from the customer represented by the time-series in Figure 3.3a. Assuming we randomly select state $\mathbf{s}_2$ in (3.2) as the agent's initial state, the first part of the agent's trajectory could be created according to

$$\begin{aligned}
&\mathbf{s}_0 = \Big[1,0,0,0,0,0,0,0,1,0,\ldots,0,1,0,0,0,0,1,0,0,0\Big] \\
&\pi(a_0 = 0|\mathbf{s}_0) = 0.94 \quad \pi(a_0 = 1|\mathbf{s}_0) = 0.06 \\
&a_0 = 0 \\
&\mathbf{s}_1 = \Big[0,0,0,0,0,0,0,1,0,0\ldots,1,0,0,0,0,1,0,0,0,0\Big] \\
&\pi(a_1 = 0|\mathbf{s}_1) = 0.83 \quad \pi(a_1 = 1|\mathbf{s}_1) = 0.17 \\
&a_1 = 0 \\
&\mathbf{s}_2 = \Big[0,0,0,0,0,0,1,0,0,1\ldots,0,0,0,0,1,0,0,0,0,0\Big] \\
&\pi(a_2 = 0|\mathbf{s}_2) = 0.67 \quad \pi(a_2 = 1|\mathbf{s}_2) = 0.33 \\
&a_2 = 1 \\
&\mathbf{s}_3 = \Big[0,0,0,0,0,1,0,0,1,0\ldots,0,0,0,1,0,0,0,0,0,1\Big] \\
&\pi(a_3 = 0|\mathbf{s}_3) = 0.97 \quad \pi(a_3 = 1|\mathbf{s}_3) = 0.03 \\
&a_3 = 0.
\end{aligned} \tag{3.3}$$

Note that the probability of making a purchase drastically decreases when the agent makes a purchase (a_2). When the algorithms are fully trained, it should be possible to predict the behavior of a new customer by initializing the agent into a state consisting of the new customer's purchase history and then execute the policy. The larger N is, the more data is required from a new customer before its future behavior can be predicted.

3.3.2 One-Hot Vector Extension

The basic state representation only contains a customer's historical purchases. Since the behavior of individual customers is varying, it is likely that the basic state representation must be extended in some way to include additional information about the specific customer. As a first suggestion, to evaluate the concept, we use one-hot encoding to represent each individual customer in the expert data set. More specifically, each state is extended with a one-hot vector that explicitly indicates which customer the purchase history belongs to.

As an example, assume that the algorithms are trained with expert data from K different customers. Furthermore, assume that the customer represented by the time-series in Figure 3.3a is the ith expert in the expert data set. Using $N = 40$ days of purchase history, the state $\mathbf{s}_0$ in (3.2) would for instance be extended according to

$$\Big[\underbrace{0,\ldots,0,1,0,\ldots 0}_{\text{One-hot vector of size } K},\underbrace{0,0,1,0,0,0,0,0,0,0,0,\ldots,1,0,0,1,0,0,0,0,1,0}_{N \text{ historical days}}\Big], \tag{3.4}$$

where the ith element in the one-hot vector is set to one, whereas the others are set to zero. The idea is that this additional piece of information should improve the agent's ability to learn individual customer behaviors.

3.3.3 Customer-Specific Days Extension

Obviously, including each customer in the expert data in the one-hot vector is not a scalable extension of the basic state representation. In a real-world scenario, the expert data set could consist of data from thousands of customers. A possible solution to this problem is to group similar customers together using some clustering method and represent each group as an element in the one-hot vector. In this way, the length of the one-hot vector could be significantly less than the number of experts. However, it is not obvious how to perform the clustering, and any segmentation would contain less information compared to using an individual based representation.

Another drawback of using one-hot encoding is that predicting the behavior of a new customer can be difficult. In order to predict the behavior of a new customer, the agent must be initialized into a state that includes a particular one-hot vector. A possible approach is to identify an expert that behaves similarly to the new customer and use that expert's one-hot encoding. However, if there is only a limited amount of data available for the new customer, it can be troublesome to accurately compare the new customer with the experts.

Instead, we replace the one-hot vector with a statistical representation of the customer's behavior, which we call customer-specific (CS) days. Essentially, the CS days are nothing else but a sequence of L historical purchases. However, in contrast to the sequence of N historical purchases, the CS days are not updated according to (3.2). Instead, the CS days are slightly modified in each update step to prevent

the agent from memorizing a fixed sequence, which would be more or less the same as using one-hot encoding. In practice, a sequence of, e.g., $2L$ historical purchases could be used to generate $L+1$ different sets of CS days by simply moving a window of length L across the sequence. In each update step, the statistical representation is modified by choosing a random sample from the $L+1$ different sets of CS days.

As a concrete example, assume $L = 5$ (in practice, L would be much larger) and that the sequence of $2L$ historical purchases looks like

$$\Big[1, 0, 0, 0, 0, 1, 0, 0, 1, 0\Big]. \tag{3.5}$$

From this sequence we obtain six different sets of CS days, namely

$$\begin{aligned}
&\Big[1, 0, 0, 0, 0\Big], \\
&\Big[0, 0, 0, 0, 1\Big], \\
&\Big[0, 0, 0, 1, 0\Big], \\
&\Big[0, 0, 1, 0, 0\Big], \\
&\Big[0, 1, 0, 0, 1\Big], \\
&\Big[1, 0, 0, 1, 0\Big].
\end{aligned} \tag{3.6}$$

Replacing the one-hot vector in the state in (3.4) with CS days, we could for instance obtain the state

$$\Big[\underbrace{0, 0, 1, 0 \ldots, 0, 0, 0, 0, \ldots 0, 0, 0, 1}_{L \text{ CS days}}, \underbrace{0, 0, 1, 0, 0, \ldots, 0, 0, 0, 1, 0}_{N \text{ historical days}}\Big]. \tag{3.7}$$

Using CS days, the agent's decision to make a purchase or not depends on the last N days of purchase history as well as the customer's statistical behavior. Hopefully, the use of CS days should increase the algorithms' ability to interpolate between experts and generalize better to new customers. CS days have a different statistical meaning than one-hot encoding. The behavior of a new customer can be predicted by initializing the agent only with data from that particular customer, i.e., there is no need for comparing the new customer with the experts in the training data set.

3.4 Standard Training of GAIL and AIRL

Proposed in 2016, GAIL revolutionized the field of imitation learning by directly extracting a policy from a given data set without explicitly learning a reward function. AIRL recovers the true reward function in an adversarial context and can be seen as an extension to GAIL. However, while the discriminator in GAIL can be represented by a single network, two networks are used to estimate the discriminator signal (2.63) in AIRL. Therefore, AIRL is more computationally expensive to train.

Both GAIL and AIRL belong to the class of adversarial imitation learning (AIL) algorithms. The standard way of training AIL algorithms follows a similar scheme. Trajectories are collected by executing the policy and the discriminator is trained to separate sampled trajectories from expert trajectories. The policy is then updated by any policy optimization method, for instance TRPO or PPO, using the feedback signal from the discriminator as reward. Traditionally, each update of the discriminator is accomplished by randomly selecting a subset of expert trajectories. Furthermore, the agent trajectories are collected after initializing the agent into randomly chosen expert states.

As discussed in Section 2.1.2, GANs are difficult to train. For instance, GANs often suffer from mode collapse – a problem that prevents the generator from learning the full complexity of the real data. Several methods have been proposed to reduce mode collapse in GANs, e.g., Wasserstein GANs (WGANs) which tend to work well in practice. In order to reduce the risk of mode collapse, we train the discriminator in GAIL by using a WGAN based objective together with gradient penalty to enforce a 1-Lipschitz constraint, see Algorithm 4. In AIRL, the discriminator, explicitly given by (2.63), is trained by minimizing the cross-entropy loss according to Algorithm 3. In both cases, we update the policy using the PPO rule, see Algorithm 1.

Algorithm 4 GAIL using WGAN-GP objective

Input Expert trajectories $\boldsymbol{\tau}^E \sim \pi^E$, initial G and D parameters $\boldsymbol{\theta}_0$ and $\mathbf{w}_0$
for $t = 0, 1, 2 \dots$ **do**
  Sample trajectories $\boldsymbol{\tau} \sim \pi_{\boldsymbol{\theta}}$ starting from a random expert state
  $\hat{\boldsymbol{\tau}} \leftarrow \epsilon\boldsymbol{\tau}^E + (1-\epsilon)\boldsymbol{\tau}$
  Calculate $\mathbf{w}'_{t+1} \leftarrow \mathbf{w}'_t$ using the gradient
    $\mathbb{E}_\tau \left[\nabla_{\mathbf{w}} D_{\mathbf{w}}(\mathbf{s},\mathbf{a})\right] - \mathbb{E}_{\tau^E}\left[\nabla_{\mathbf{w}} D_{\mathbf{w}}(\mathbf{s},\mathbf{a})\right] + \mathbb{E}_{\hat{\tau}}\left[\nabla_{\mathbf{w}} \lambda[(\|\nabla_{\hat{\boldsymbol{\tau}}} D_{\mathbf{w}}(\mathbf{s},\mathbf{a})\|_2 - 1)^2\right]$
  Calculate $\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t$ using TRPO/PPO with loss function $\log\left(D_{\mathbf{w}_{t+1}}(\mathbf{s},\mathbf{a})\right)$
end for
Return $\hat{\pi}_{\boldsymbol{\theta}}{}^E$

3.5 MMCT-GAIL

The standard training procedure of AIL algorithms assumes that there is a single expert policy π^E that describes the behavior of the experts. This assumption is perfectly fine when imitation learning is applied to the problem of training, e.g., a self-driving car. In such a case, the algorithm should learn to drive at least as good as a human driver, and learning the behavior of the average human driver is thus a good objective. In our case, we instead want to train the algorithms to learn behaviors of multiple experts. This could be compared to training a self-driven car to learn the individual behaviors of multiple drivers.

As seen in Section 3.1, the behavior of the customers in the population is varying. In fact, the customer behavior distribution is formed by several mixture components that represent individual behaviors. If the mixture components are to be learned,

the standard way of training the algorithms is most likely suboptimal. In particular, feeding the discriminator with randomly chosen data may slow down the learning process. To improve the algorithms ability to learn diverse individual behaviors we redefine the objective of the imitation learning problem.

We assume that each expert i, where $i = 0, \ldots, N_{\text{experts}}$, behaves according to an individual policy π_i^E – a mixture component of the expert policy π^E. We want to train a model so that the learned policy $\pi_{\boldsymbol{\theta}}$ generates realistic behavior for every expert. We suggest to learn individual customer behaviors by modifying Algorithm 4 to match expert demonstrations with the corresponding agent samples. The new algorithm, which we call matched mixture component trained GAIL (MMCT-GAIL), is outlined in Algorithm 5.

In each iteration t, the expert data set is divided into j random subsets of equal size. Consider a particular subset of experts $\mathcal{H}_j$. For each expert i in the subset, the agent is initialized into a randomly chosen state from that particular expert, and a sample is collected by executing the current policy $\pi_{\boldsymbol{\theta}}$. When samples for all experts in $\mathcal{H}_j$ are collected, the discriminator is trained to distinguish expert trajectories from the *corresponding* policy samples by batch-wise optimization of the WGAN-GP objective. Once the discriminator is updated, a policy optimization method, e.g., TRPO or PPO, is used for updating the policy with the discriminator signal as reward.

3.6 Related Methods

Halfway into our work we learned about sequence generative adversarial nets with policy gradient (SeqGAN) [45]. SeqGAN is not categorized as an imitation learning algorithm but is a model with an adversarial structure, adapted for sequential data, which uses a policy gradient to update the generator. More specifically, the generator is represented by a recurrent neural network, in particular an LSTM, updated by a gradient policy method. The discriminator is preferably a convolutional network that takes a full sequence as input and outputs a RL reward signal that the LSTM uses as loss. Choosing an LSTM as a policy representation is instinctively a convenient choice since LSTMs have shown good performance on sequential data, e.g., text translation. SeqGAN could be an alternative to the methods used in this thesis but has not been further investigated.

Algorithm 5 MMCT-GAIL

Input Set of expert trajectories $\mathcal{D} = \{\boldsymbol{\tau}_i^E\}_{i=1}^{N_{\text{experts}}}$ where $\boldsymbol{\tau}_i^E \sim \pi_i^E$,
initial G and D parameters $\boldsymbol{\theta}_0$ and $\mathbf{w}_0$, k
for $t = 0, 1, 2, \ldots$ **do**
$\mathbf{w}'_0 \leftarrow \mathbf{w}_t$
$\boldsymbol{\theta}'_0 \leftarrow \boldsymbol{\theta}_t$
Split $\mathcal{D}$ into M random disjoint subsets $\mathcal{H}_j$ of equal size k
for $j = 0, 1, 2, \ldots, M-1$ **do**
for every expert i in subset $\mathcal{H}_j$ **do**
Sample an initial state $\mathbf{s}_i^0$ from $\boldsymbol{\tau}_i^E$
Sample a trajectory $\boldsymbol{\tau}_i \sim \pi_{\boldsymbol{\theta}'_j}$ starting from $\mathbf{s}_i^0$
$\hat{\boldsymbol{\tau}}_i \leftarrow \epsilon\boldsymbol{\tau}_i^E + (1-\epsilon)\boldsymbol{\tau}_i$
end for
Compute $\mathbf{w}'_{j+1} \leftarrow \mathbf{w}'_j$ using the gradient
$$\sum_{i\in\mathcal{H}_j} \Big(\mathbb{E}_{\tau_i}\left[\nabla_{\mathbf{w}} D_{\mathbf{w}}(\mathbf{s},\mathbf{a})\right] - \mathbb{E}_{\tau_i^E}\left[\nabla_{\mathbf{w}} D_{\mathbf{w}}(\mathbf{s},\mathbf{a})\right] + \mathbb{E}_{\hat{\tau}_i}\left[\nabla_{\mathbf{w}} \lambda[(\|\nabla_{\hat{\boldsymbol{\tau}}_i} D_{\mathbf{w}}(\mathbf{s},\mathbf{a})\|_2 - 1)^2\right] \Big)$$
Compute $\boldsymbol{\theta}'_{j+1} \leftarrow \boldsymbol{\theta}'_j$ using policy optimization with loss function $D_{\mathbf{w}'_{j+1}}(\mathbf{s},\mathbf{a})$.
end for
$\mathbf{w}_{t+1} \leftarrow \mathbf{w}'_M$
$\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}'_M$
end for
Return $\hat{\pi}_{\boldsymbol{\theta}}^E$

4
Experimental Evaluation

The purpose of this thesis is to evaluate the performance of the imitation learning algorithms presented in Chapter 3 on the synthetic purchase data described in Section 3.1. In this chapter, we present our major findings. We specify the general implementation details in Section 4.1. Then, in Section 4.2, we evaluate the performance of GAIL and AIRL, trained in a standard way. In Section 4.3, we evaluate the performance of MMCT-GAIL. Each algorithm is evaluated using each of the MDP representations described in Section 3.3.

When evaluating the performance of the algorithms, we utilize the method for comparing time-series into distributions which we developed in Section 3.2. The algorithms learn a policy that can be used to predict the behavior of individual customers. In doing so, the agent is initialized into a state from the customer and the policy is executed over 10000 time steps. The sequence of actions is then converted into a distribution which we compare to the true distribution of the customer. We use the Wasserstein metric to measure the difference between distributions. If nothing else is stated, we compare unconditional distributions. To simplify language, we write in terms of comparing the behavior of the agent and the customer, or simply, comparing the agent and the customer (when we in fact compare the corresponding distributions).

We denote customers that are included in the training data set as experts. The average behavior of the experts is obtained by averaging the corresponding distributions. We obtain the average behavior of the agent by initializing the agent into a state from each of the experts, executing the policy and averaging the corresponding distributions. If the agent is initialized into a state from the ith expert, we sometimes denote its behavior as "Agent i".

4.1 Experimental Details

Our implementations of the algorithms are based on an open source code [46], which in turn is heavily dependent on the deep learning framework Chainer. In all algorithms, the policy is updated using PPO implemented in an actor-critic way. We represent the policy (the actor) and the value function (the critic) as separate networks. More precisely, we use multilayer perceptrons with two hidden layers with 64 neurons each. The ReLu function is used as activation function after each hidden layer. Naturally, the softmax function is applied to the outputs of the policy

network, while there is no activation function after the output layer of the value network. We update the network parameters using the Adam optimizer with learning rate 0.0003. There are a lot of hyperparameters that need to be set in PPO. In general, we use the default values summarized in Table A.1.

The discriminators in all algorithms are represented as multilayer perceptrons with two hidden layers. Each hidden layer consists of 64 neurons. In fact, the discriminator in AIRL utilizes two networks: a reward network as well as a value network. Again, the Adam optimizer is used to update the network parameters, but for the discriminator networks, the learning rate is set to 0.00001. In MMCT-GAIL, the size k of each subset $\mathcal{H}_j$ is set to 10. For further implementation details we refer to Table A.1.

When training the algorithms using the basic state representation or the CS days extension, we use expert data from 100 experts. When using one-hot encoding, we limit ourselves to 10 experts. Each expert trajectory has a length of three years, i.e., it consists of 1095 days of purchase history. Similarly, we collect agent trajectories by executing the policy over 1095 time steps (each time step corresponds to one day), which defines the length of one episode. In total, we train the algorithms over 20000 episodes.

Figures B.1–B.10 in Appendix B.1 show the behaviors, represented as the conditional distributions, of the first ten experts in the expert data set. In addition, the heatmap in Figure 4.1 shows the pairwise Wasserstein distance between the unconditional distributions of the first ten experts. As an example, we clearly see that the second and third expert behave in a similar way whereas the first and sixth expert behave very differently. In the following sections, we compare the behavior of the agent with the behaviors of the experts in a similar way. If the policy is optimally learned, the agent should be able to behave as each of the experts and the corresponding heatmap would be identical to the heatmap in Figure 4.1.

4.2 Standard Training

In this section, we evaluate the performance of the standard algorithms using three different state representations. In Section 4.2.1, we use a state that consists of three months (90 days) of historical purchases. Then, in Section 4.2.2, we extend this state with a one-hot vector that represents each individual customer in the expert data set. Finally, we evaluate the customer-specific (CS) days extension in Section 4.2.4.

4.2.1 Basic State Representation

In Figure 4.2, we compare the average behavior of the agent with the average behavior of the experts after 20000 training episodes. The upper and lower panels show the result obtained with GAIL and AIRL, respectively. As we see, GAIL and AIRL show similar performance. The Wasserstein distance between the unconditional dis-

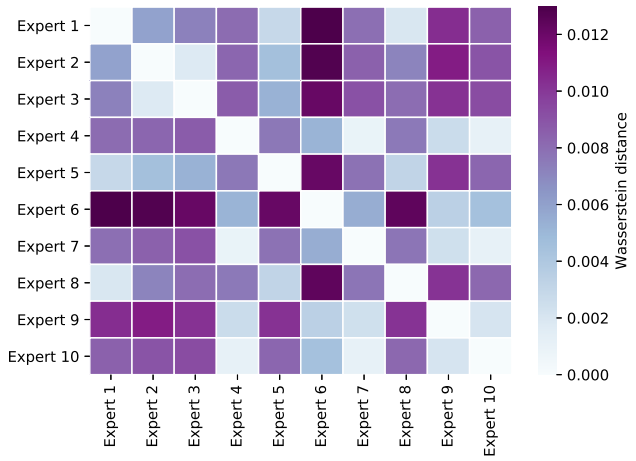
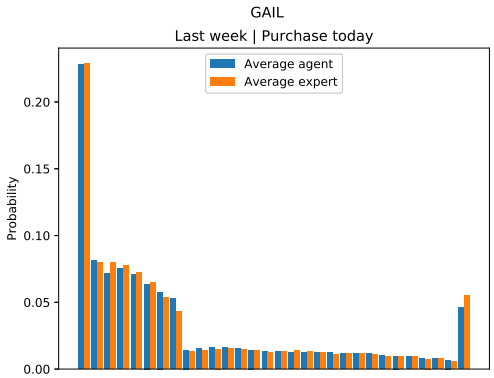


Figure 4.1: A comparison of the ten first experts in the data set. The heatmap is created by sampling time-series for each of these experts and then converting the time-series into distributions, which are compared using the Wasserstein metric. In particular, we use the unconditional distributions, see Figure 3.6.

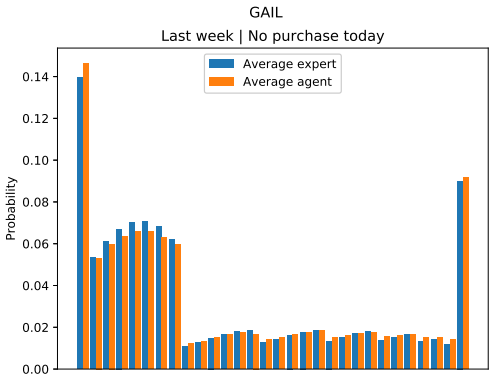
tributions is 0.0013 (GAIL) and 0.0015 (AIRL), respectively. Clearly, the agent has learned to behave as the average customer in the expert data set.

Although the result in Figure 4.2 is promising, we are interested in learning individual customer behaviors. Figure 4.3 indicates that this is not the case. Here, the agent is initialized into a state from the second expert, see Figure B.2. The second expert behaves quite differently from the average expert, whose behavior is represented by the green bars in the figure. As we see in Figure 4.3, neither GAIL nor AIRL learns a policy that enables the agent to behave as the second expert. Instead, the agent imitates the average expert. Comparing the unconditional distributions, the Wasserstein distance between the agent and the average expert is 0.0010 (GAIL) and 0.0014 (AIRL), respectively, while the distance between the agent and the second expert is 0.0059 (GAIL) and 0.0052 (AIRL), respectively. In Figure 4.4, the agent is compared to a larger number of experts. Regardless of initialization, we see that the agent behaves as the average expert. This is reminiscent of mode collapse, which is a common problem when training GANs, see Section 2.1.2.

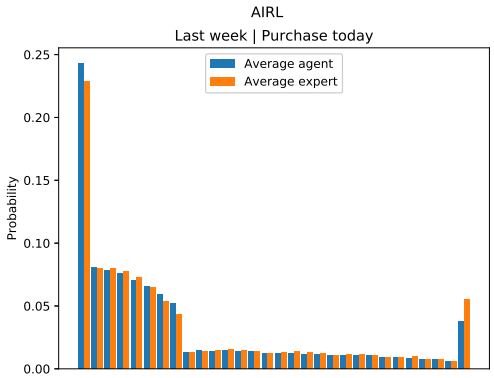
In Figure 4.5, we show the training process of GAIL and AIRL, respectively. The green line shows the change in distance between the average expert and the average agent. The blue and orange lines show the distance change between the agent and individual customers. More specifically, the blue line compares the agent to the experts, whilst the orange line compares the agent to 50 new customers. The error bands show the 95 % empirical confidence interval around the mean (solid line). Again, we see that neither GAIL nor AIRL learns a policy that can predict individual behaviors.



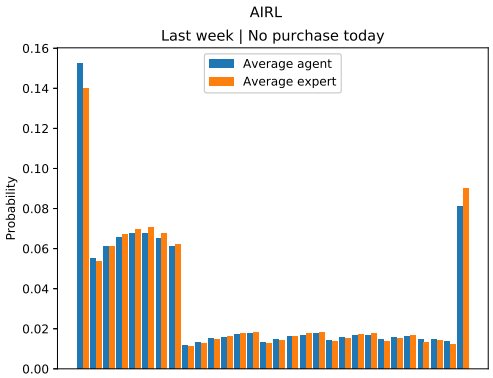
(a) GAIL: purchase today.



(b) GAIL: no purchase today.

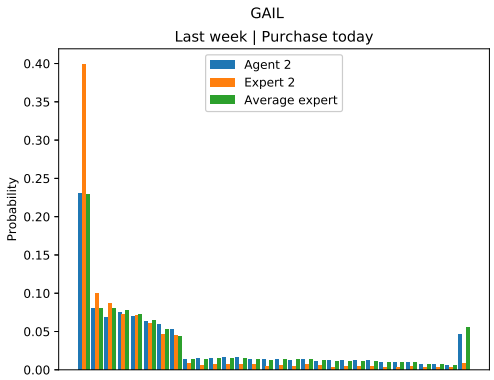


(c) AIRL: purchase today.

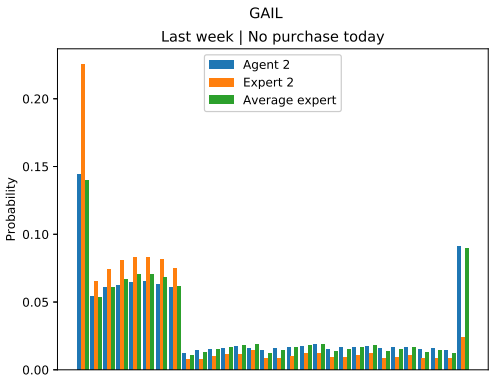


(d) AIRL: no purchase today.

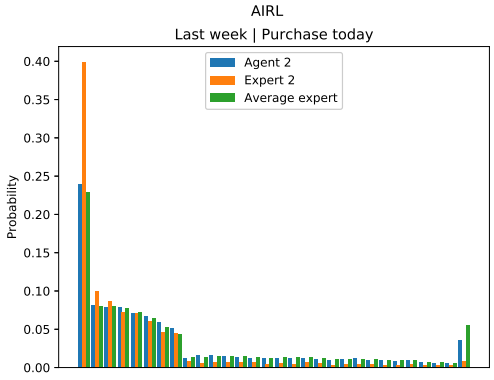
Figure 4.2: The average purchasing behavior of the agent compared to the average purchasing behavior of 100 experts, using the basic state representation. Both GAIL and AIRL learn a policy that enables the agent to behave very similarly to the true average behavior.



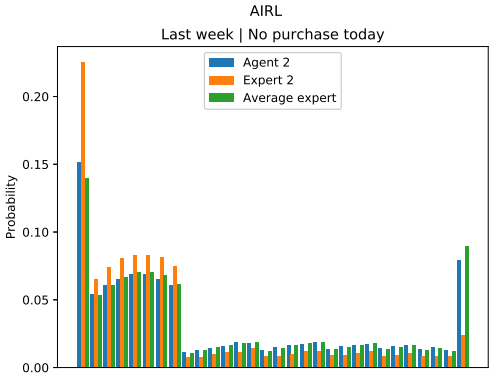
(a) GAIL.



(b) GAIL.

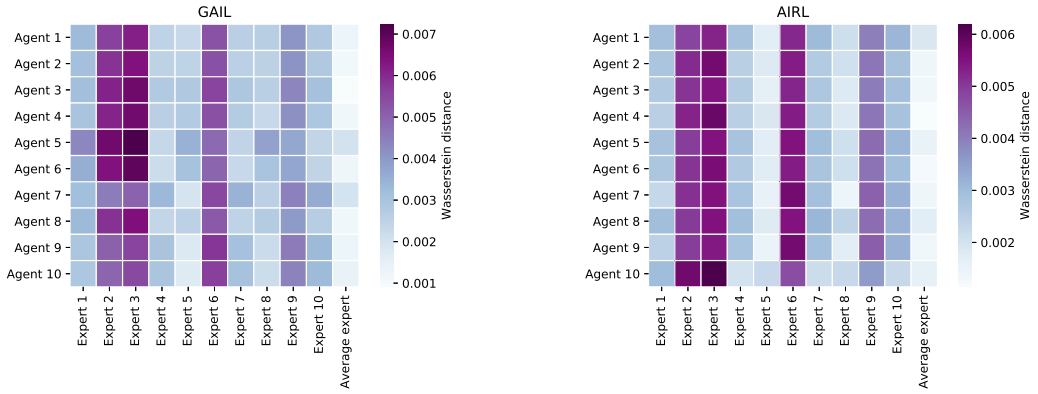


(c) AIRL.



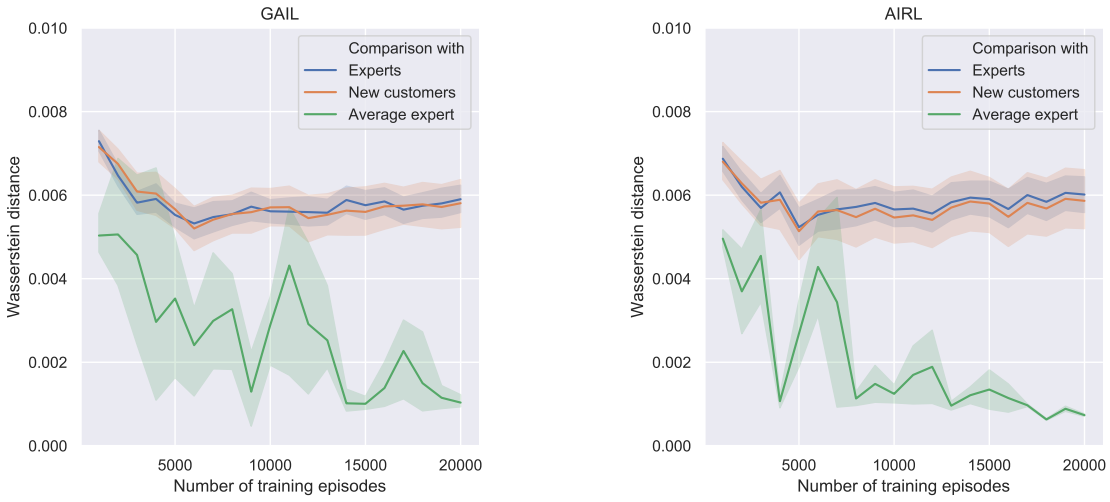
(d) AIRL.

Figure 4.3: A comparison between the agent (blue bars) and the second expert (orange bars) using a basic state representation. Neither GAIL nor AIRL learns a policy that enables the agent to behave as the second expert. Instead, the agent imitates the behavior of the average expert (green bars). Figure B.11 in Appendix B shows a similar comparison with the ninth expert.



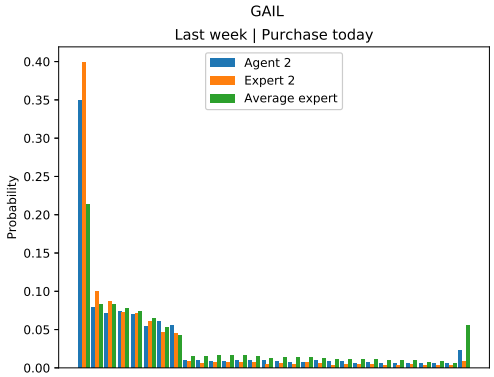
(a) GAIL. **(b)** AIRL.

Figure 4.4: Using the basic state representation, both GAIL and AIRL fail in learning a policy that predicts individual purchasing behaviors. The agent is initialized into a state from each of the experts ("Agent 1", "Agent 2", etc.) but always behaves as the average expert.

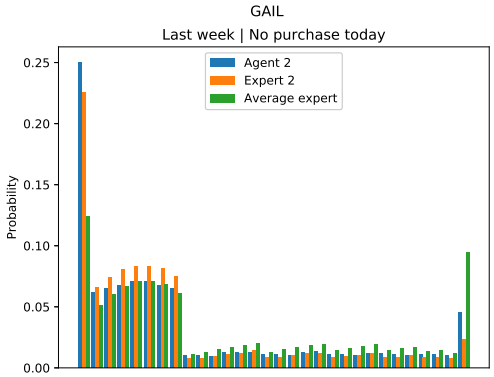


(a) GAIL. **(b)** AIRL.

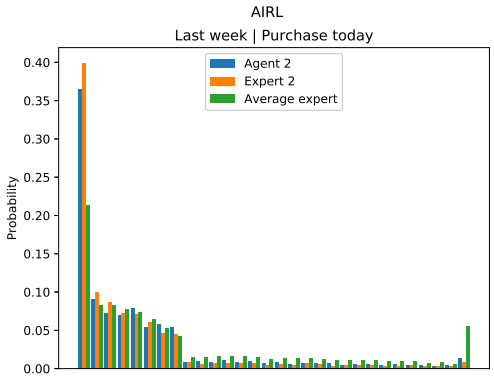
Figure 4.5: The training process for GAIL and AIRL using the basic state representation. The policy network is regularly saved during the training. At each evaluation, the agent is initialized into states from each expert as well as 50 new customers. Executing the policy, the corresponding distributions are compared using the Wasserstein distance.



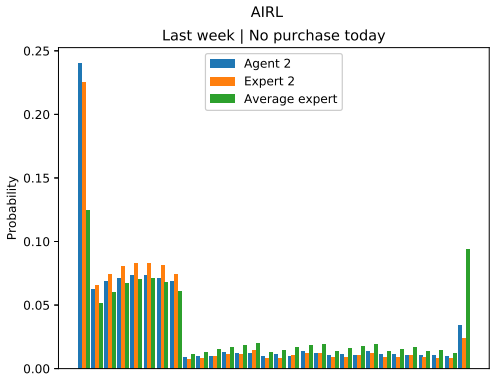
(a) GAIL.



(b) GAIL.



(c) AIRL.



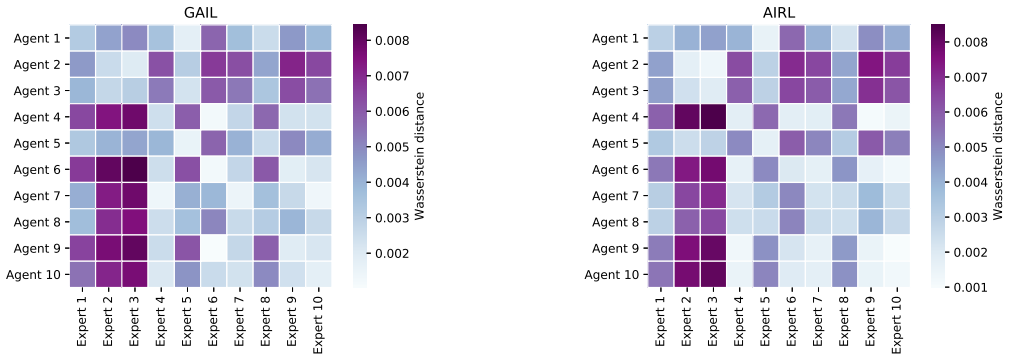
(d) AIRL.

Figure 4.6: A comparison between the agent and the second expert, when extending the state with a one-hot vector. Figure B.12 in Appendix B shows a similar comparison with the ninth expert.

4.2.2 One-Hot Vector Extension

In order to improve prediction of individual behaviors, we extend the basic state with a one-hot vector that encodes each expert. In Figure 4.6, the agent is initialized into a state from the second expert and its behavior is then compared to the second expert as well as the average expert. In contrast to Figure 4.3, we now see that the agent behaves quite differently from the average customer and more like the second expert. The results obtained with GAIL and AIRL are similar.

Figure 4.7 indicates that the agent is able to behave as almost all of the experts. Like Figure 4.1, the diagonal elements are clearly prominent. There are some details worth commenting on. For instance, we see that the behavior of the first expert is hard to predict. Furthermore, essentially the same predictions are made for both expert 7 and expert 8, even though these experts are quite different according to Figure 4.1. In Figure B.13 in Appendix B, the heatmap for GAIL is shown for every thousand episode during the training process.



(a) GAIL. **(b)** AIRL.

Figure 4.7: A comparison between the agent and all experts. Note that the diagonal elements are prominent, meaning that the learned policy enabled prediction of individual behaviors. These heatmaps should be compared to the heatmap in Figure 4.1, where experts are compared to each other.

Figure 4.8 summarizes our findings when using one-hot encoding. The dashed orange lines show the learning process of GAIL and AIRL, respectively, using only the basic state representation. In order to predict the behavior of a new customer, the distribution of the new customer is compared to the distribution of each of the ten experts. The element in the one-hot vector that corresponds to the expert who is most similar (in terms of smallest Wasserstein distance between the distributions) to the new customer is set to one. The initial purchase history is taken from the new customer. The red line indicates how the distance to the most similar expert varies during the training. We see that the Wasserstein distance for comparison with experts and new customers decreases faster for GAIL than for AIRL. However, the result after 20000 episodes is essentially the same.

4.2.3 Influence of One-Hot Encoding

It is interesting to investigate how the policy responds to a modification of the initial state. In doing so, we analyze an agent trained with GAIL. For sake of simplicity, we represent purchasing behaviors by purchase ratios, see Section 3.1. In Figure 4.9, the dashed green and orange lines show the purchasing behavior of the agent initialized with purchase history from two different new customers, while the one-hot vector consists of *only zeros*. The solid green and orange lines show the actual behaviors of the two customers. The blue line shows the average behavior of the first ten experts. We see that the agent behaves similarly to the average expert. In other words, when predicting the behavior of a new customer without employing one-hot encoding, we obtain the behavior of the average expert.

Let us also investigate what happens when the initial purchase history contains only zeros, but the one-hot vector represents a specific expert. We consider the

(a) GAIL. **(b)** AIRL.

Figure 4.8: The training process of GAIL and AIRL when extending the state with a one-hot vector. In comparison with the results obtained with the basic state, wee see a clear improvement in the predictions of individual behaviors. Interestingly, the curves for experts and new customers follow each other, meaning that the algorithms perform just as well on new customers as on experts in the training data set.

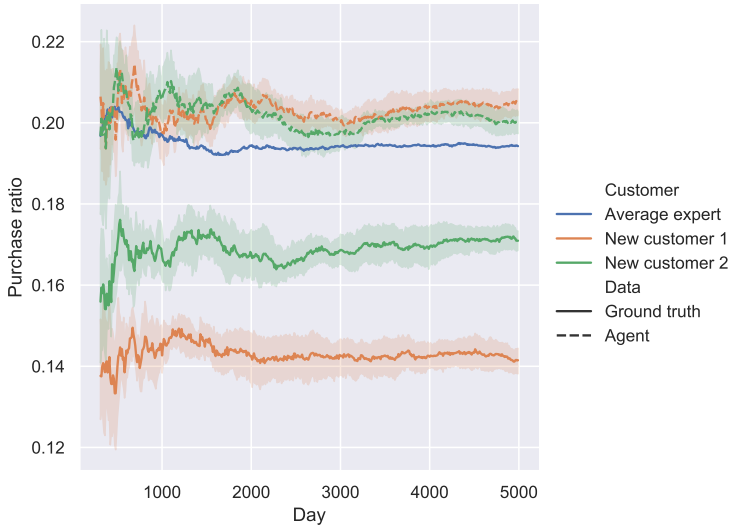


Figure 4.9: Predicted customer behaviors of two new customers when replacing the one-hot vector with the zero vector. The actual behaviors of the customers are included as a reference.

Table 4.1: The ground truth behaviors of the first five experts and the corresponding predictions when the initial purchase history consists of only zeros. The purchase ratios are calculated for sequences of length 5000.

	Purchase ratio	
Expert	Ground truth	Predictions
1	0.173 ± 0.003	0.173 ± 0.003
2	0.142 ± 0.003	0.149 ± 0.002
3	0.139 ± 0.002	0.136 ± 0.002
4	0.219 ± 0.003	0.213 ± 0.003
5	0.167 ± 0.003	0.174 ± 0.002

five first experts. The actual behaviors of these experts are summarized in the column "Ground truth" in Table 4.1. The column "Predictions" shows the predicted behaviors. We conclude that although the initial purchase history consists of only zeros, the agent behaves similarly to the respective expert. Clearly, the policy has learned to map the information in the one-hot vector to a particular expert behavior.

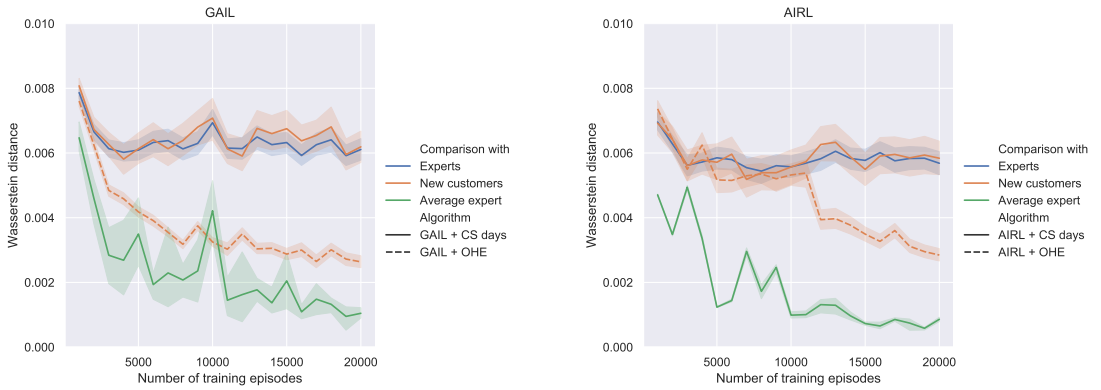
4.2.4 Customer-Specific Days Extension

Using one-hot encoding to better learn individual behaviors is not an optimal solution. First of all, this solution is not scalable to hundreds or thousands of experts. In addition, it is not optimal to compare a new customer to existing experts in order to find the most similar one, which is necessary to predict the behavior of a new customer. Instead, we replace the one-hot vector with 90 customer-specific (CS) days with the intent that the CS days should explain the customer's statistical behavior. For this reason, we reduce the purchase history to 30 days. When training the algorithms using the CS days extension we generate 100 different sequences of CS days.

Figure 4.10 shows the results for GAIL and AIRL, respectively. The results obtained using one-hot encoding are included as a reference. As we see, both GAIL and AIRL predict the average behavior very well, but fail to predict any individual behaviors. The results are comparable to those we obtained using the basic state representation.

4.3 MMCT-GAIL

In the standard way of training the algorithms, the experts are assumed to behave according to an underlying policy π^E. In Section 3.5, we used self-driving cars as a concrete example when this assumption is met. If the self-driving car is trained on driving data from multiple drivers, it would be a good outcome if the car learned to drive as the average driver. However, learning individual customer behaviors is a different problem that potentially needs to be solved with a different approach. Therefore, we proposed matched mixture component trained GAIL (MMCT-GAIL) in Section 3.5.



(a) GAIL. **(b)** AIRL.

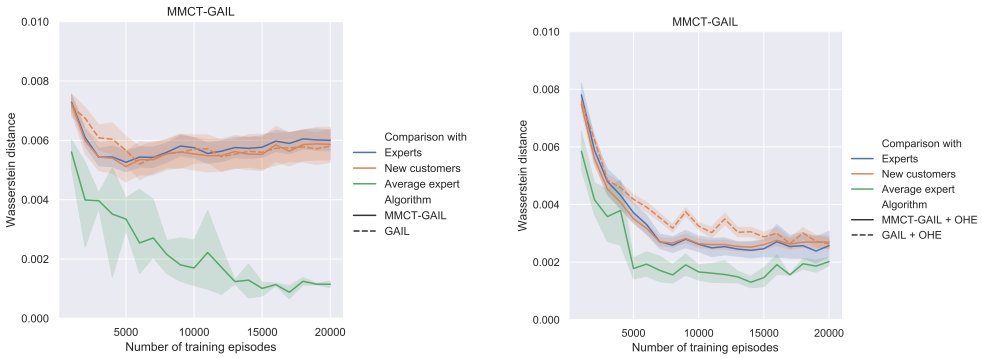
Figure 4.10: Training progress for GAIL and AIRL when extending the basic state with CS days. The results obtained using one-hot encoding are included as a reference. We see that neither GAIL nor AIRL learn individual behaviors.

Figure 4.11a shows the performance of MMCT-GAIL using each of the three state representations as discussed in Section 3.3. The results obtained with GAIL in Section 4.2 are included as references. Using the basic state representation, we see that MMCT-GAIL predicts the average expert behavior but fails in predicting individual behaviors. The performance when evaluating MMCT-GAIL on new customers is similar to the performance of GAIL.

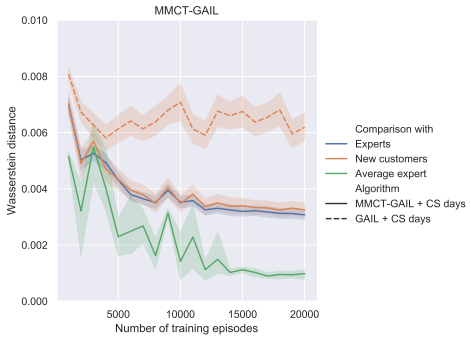
Extending the state with a one-hot vector, we see that MMCT-GAIL accurately predicts individual behaviors. Interestingly, the results for experts and new customers are almost identical. Note that MMCT-GAIL is preferable to GAIL when predicting new customers during the initial stage of the training process. In the long run, it seems like GAIL performs similarly to MMCT-GAIL, but MMCT-GAIL learns individual behaviors faster than GAIL. Thus, we argue that MMCT-GAIL improves the overall learning process.

Perhaps most interesting is that MMCT-GAIL, when using CS days instead of one-hot encoding, still learns individual behaviors. After 20000 episodes, the average Wasserstein distance between the agent and new customers is clearly below 0.004. This should be compared to the reference, i.e., the traditional way of training GAIL, which exceeds 0.006. This result is important since it allows CS days to be used instead of one-hot encoding. Using CS days is preferable in a real-world scenario, because the behavior of a new customer can be simulated directly, without having to compare the new customer to experts in the training data set.

As a final comparison, the performance of the three algorithms when evaluating on new customers is summarized in Figure 4.12. The different state representations are marked as blue (basic), orange (one-hot encoding) and green (CS days). We see that MMCT-GAIL is superior to both GAIL and AIRL in terms of learning individual customer behaviors when using CS days. Using one-hot encoding, MMCT-GAIL



(a) MMCT-GAIL using basic state. **(b)** MMCT-GAIL using OHE.



(c) MMCT-GAIL using CS days.

Figure 4.11: A summary of the results obtained with MMCT-GAIL using different state representations. The results obtained with GAIL are used as references. **(a)**: Using a basic state representation, MMCT-GAIL fails in learning any individual behaviors, but the distance to the average expert steadily decreases during the training process. **(b)**: Using dummy variables, MMCT-GAIL successfully learns individual behaviors. Note in particular that the performance is better than the reference. **(c)**: MMCT-GAIL also learns individual behaviors when the dummy variables are replaced with CS days. This result is especially interesting since GAIL fails in doing so.

(a) GAIL. **(b)** AIRL. **(c)** MMCT-GAIL.

Figure 4.12: A summary of the results obtained with different state representations: the basic one (blue lines), the one-hot encoding extension (orange lines) and the CS days extension (green lines). Only the performance on new customers is considered.

learns individual behaviors faster than the other algorithms. However, it should be pointed out that the algorithm itself is not a quick fix to the problem of learning individual behaviors. Using the basic state, the algorithm only learns an average behavior. The choice of state representation is therefore crucial.

4.3.1 Influence of CS days

The intuition behind a state that consists of both CS days and purchase history is that the CS days should capture the customer's statistical purchasing behavior whilst the purchase history keeps track of recent actions. The policy network should learn to utilize both these parts to make accurate predictions of individual customer behaviors. In Figure 4.13, we evaluate the training process for MMCT-GAIL on new customer data when varying the length of each part.

In the left panel, the length of the purchase history is set to 30, 60 and 90 days, respectively, while keeping the number of CS days fixed (90 days). Even though the differences are quite small, we see that the length of the purchase history can advantageously be reduced to 30 days. It is not impossible that this number can be even smaller. Recall from Section 3.2 that the auto-correlation of the time-series indicates that only the decisions from the last week affect today's purchase decision.

In the right panel, the number of CS days is varied while keeping the purchase history fixed (30 days). It seems like six months is the best choice, followed by two and three months, respectively. Clearly, 30 CS days is too little. One can imagine that a sequence of 30 days does not capture the statistical behavior of a customer. Interestingly, is seems like 365 CS days is too much. However, it should be pointed out that the network architectures were the same during all evaluations. When using 365 CS days, the state consists of almost 400 elements, which potentially requires larger networks.

(a) Length of purchase history. **(b)** Number of CS days.

Figure 4.13: The influence of the length of the purchase history as well as the number of CS days when predicting the behavior of new customers.

4.4 Discussion

In the above sections, we have presented the major findings of this work. First, we showed the results obtained with standard implementations of GAIL and AIRL. Then, we evaluated the performance of MMCT-GAIL, which we proposed as a novel method for learning individual customer behaviors. In this section, we discuss some interesting aspects of our results. We start with a discussion of our choice of evaluation metric.

4.4.1 Evaluation Metric

We have evaluated the results using the proposed method for converting binary time-series into distributions. Ideally, this method should transfer different time-series into equally different distributions. If information is lost during the conversion, it is more difficult to draw certain conclusions about the results. We strongly believe that we have developed a robust evaluation method, but no method is of course perfect. In a future work, it would be interesting to study different evaluation methods in more detail.

A comment should also be made on using the Wasserstein metric to measure the distance between distributions. The Wasserstein distance is dependent on how the bars are arranged. For example, by changing the order of the bars in Figure 3.7, we would obtain a different Wasserstein distance between the distributions. We have chosen to order the bars according to increasing number of weekly purchases, and we argue that this choice is the natural way of arranging the bars. However, different approaches, e.g., summing the absolute difference between each pair of bars, would

be possible.

4.4.2 Algorithms

In Section 4.2, we compared the standard implementations of GAIL and AIRL. In general, the algorithms show similar performance for all state representations. In comparison with GAIL, AIRL has two major advantages: It is robust to changes in the dynamics and it recovers the true reward function. However, in our case, the true reward function is difficult to interpret and therefore of minor interest. Furthermore, the customers in our synthetic data set have a constant behavior over time, i.e., the dynamics is constant. By also taking into account that AIRL requires approximately twice the run time of GAIL, we argue that GAIL, in our case, is the better choice.

As we have already pointed out, MMCT-GAIL is preferable to GAIL and AIRL in terms of learning individual customer behaviors. Both GAIL and MMCT-GAIL aim to minimize the Wasserstein distance between the generating distribution and the true distribution. However, while the discriminator in GAIL is fed with randomly selected samples, the discriminator in MMCT-GAIL receives matched samples. That is, the discriminator in MMCT-GAIL minimizes the Wasserstein distance between matched mixture components of the expert policy.

4.4.3 State Representations

Although MMCT-GAIL outperforms GAIL and AIRL in our experiments, it is important to note that the algorithm itself is not a quick fix to the problem of learning individual behaviors. The choice of state representation is critical. For instance, using the basic state, which only consists of historical purchases, all algorithms fail in learning individual behaviors.

The results obtained with the basic state could be interpreted as the algorithms suffering from mode collapse – a well-known problem in GANs as discussed in Section 2.1.2. However, there are differences between AIL algorithms and GANs. For instance, in order to predict the behavior of a particular customer, we initialize the agent into a state that is formed by purchase history from that specific customer. In GANs, there is no such thing as an initial state. Instead, data is sampled by feeding noise through the generator network. In addition, we use the WGAN-GP objective when training GAIL and MMCT-GAIL. As mentioned before, WGANs have a lower tendency to suffer from mode collapse. Therefore, we cannot say with certainty that we observe mode collapse.

One-hot encoding was introduced to prove that the algorithms could learn individual behaviors. We expected the algorithms to overfit to the experts and thus not generalize to new customers. However, we found that the predictions were just as good for new customers as for experts. For instance, review Figure 4.8, where the blue and orange lines follow each other. We also see that the red line, which relates

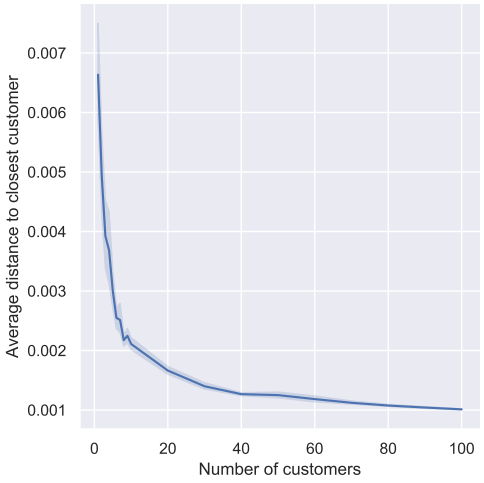


Figure 4.14: The average Wasserstein distance between 50 new customers and the closest expert in the expert data set as a function of the number of experts included in the expert data set.

the agent – when initialized into a state from a new customer – to the expert who is most similar to the new customer, follows the blue and orange ditto. If overfitting would have been the case, the red line should lie below the orange one.

Figure 4.14 puts this result into a different perspective. The figure shows the average Wasserstein distance between a new customer and the closest expert as a function of the number of experts. Clearly, a new customer is more and more similar to the closest expert as the number of experts increases. It seems like ten experts span the space of customer behaviors quite well. Using ten experts, it is likely that a new customer behaves similarly to one of the experts. It is therefore difficult to determine whether the agent follows the behavior of the new customer or simply imitates the closest expert. That is, it is hard to assess whether or not overfitting occurs.

Possibly the most interesting part of our work is that MMCT-GAIL learns individual purchasing behaviors when the one-hot vector is replaced with CS days. As we found out in Section 4.2, the CS days extension does not allow GAIL and AIRL to learn individual behaviors. It should be pointed out that we have limited ourselves to only evaluating the algorithms with states that consist of 90 CS days and 30 days of purchase history. This choice works for MMCT-GAIL, but it not impossible that other settings would improve the performance of GAIL and AIRL. However, it is desirable to reduce the amount of data needed to form a state. Therefore, we argue that MMCT-GAIL is superior to GAIL and AIRL regardless of the performance of GAIL and AIRL using larger states.

4.4.4 Final Comments

During our work, we have experimented with different state representations that have not been presented in this thesis. For example, we tried to represent the purchase history as the number of days between consecutive purchases, but we found that this representation impaired learning. However, there could be other state representations that work even better than the ones presented in this thesis.

As an experiment when using one-hot encoding, we only let the generator network, i.e., the parameterized policy, see the one-hot vector. The one-hot vector was omitted from the state when feeding state-action pairs to the discriminator. The intuition behind this was that the one-hot vector would prevent the discriminator from effectively minimizing the distance between the distributions of state-action pairs. However, this intuition proved to be incorrect, and we found that the one-hot vector must be provided to both the generator and the discriminator in order to obtain a good result.

5
Conclusion

In this thesis we have evaluated the possibility to learn and predict realistic customer behaviors using two different imitation learning algorithms, namely GAIL and AIRL. In particular, the algorithms were evaluated on synthetic data in the form of time-series of daily purchases of consumer goods. In order to analyze the performance of the algorithms we developed a method for converting time-series into distributions, which can be compared more easily. Moreover, we proposed matched mixture component trained GAIL (MMCT-GAIL) as an approach to the specific problem of predicting individual customer behaviors.

We conclude that the performance is highly dependent on the choice of state representation. When using a basic state that consists of only purchase history, all algorithms fail in learning individual customer behaviors. When extending the state with a one-hot vector, the performance of all algorithms significantly increases. Most interestingly, MMCT-GAIL is the only algorithm that accurately predicts individual customer behaviors when replacing the one-hot vector with CS days. CS days are intended to generalize better to new customers. In addition, our results show that MMCT-GAIL speeds up learning compared to GAIL and AIRL.

We argue that MMCT-GAIL is a promising algorithm to the specific problem of predicting individual customer behaviors. However, we found that a rather small number of experts spans the space of customer behaviors. A new customer is therefore likely to be similar to one of the experts, which makes it difficult to assess whether or not the algorithms overfit to the expert data.

5.1 Future Work

According to our results, GAIL and AIRL show similar performance. In contrast to GAIL, AIRL estimates the true reward function and is intended to be more robust to changes in the dynamics. However, in our work, the reward function is difficult to interpret and the behavior of the customers is constant over time. As a future work, it would be interesting to construct the synthetic data in a way that enables interpretation of the learned reward function. It would also be interesting to analyze the robustness of AIRL, when trained in a dynamic environment.

In this work, we have focused mainly on analyzing customer behavior in terms of purchase events. That is, we have not focused on analyzing the customers' purchase

amounts. We briefly investigated this extended problem using MMCT-GAIL, but no satisfying results were obtained. It would be of great interest to further analyze purchase amounts.

Finally, it would be interesting to theoretically define the problem that MMCT-GAIL solves. It is perhaps possible to extend the problem formulation of GAIL. Moreover, it would also be interesting to apply the MMCT framework to other AIL algorithms.

Bibliography

[1] Volodymyr Mnih et al. "Playing atari with deep reinforcement learning". In: *NIPS deep learning workshop*. 2013.

[2] John Moody and Matthew Saffell. "Learning to trade via direct reinforcement". In: *IEEE transactions on neural networks* 12.4 (2001), pp. 875–889.

[3] Mariya Popova, Olexandr Isayev, and Alexander Tropsha. "Deep reinforcement learning for de novo drug design". In: *Science advances* 4.7 (2018), eaap7885.

[4] Jing-Cheng Shi et al. "Virtual-Taobao: Virtualizing real-world online retail environment for reinforcement learning". In: *AAAI conference on artificial intelligence*. 2019.

[5] Jonathan Ho and Stefano Ermon. "Generative adversarial imitation learning". In: *Advances in neural information processing systems*. 2016.

[6] Justin Fu, Katie Luo, and Sergey Levine. "Learning robust rewards with adversarial inverse reinforcement learning". In: *International conference on learning representations*. 2018.

[7] Huang Xiao et al. "Wasserstein adversarial imitation learning". In: *arXiv preprint arXiv:1906.08113* (2019).

[8] Xue Bin Peng et al. "Variational discriminator bottleneck: Improving imitation learning, inverse RL, and GANs by constraining information flow". In: *International conference on learning representations*. 2019.

[9] Lantao Yu, Jiaming Song, and Stefano Ermon. "Multi-agent adversarial inverse reinforcement learning". In: *International conference on machine learning*. 2019.

[10] Karol Hausman et al. "Multi-modal imitation learning from unstructured demonstrations using generative adversarial nets". In: *Advances in neural information processing systems*. 2017.

[11] Feng Yu et al. "A dynamic recurrent model for next basket recommendation". In: *International ACM SIGIR conference on research and development in information retrieval*. 2016.

[12] Yu Zhu et al. "What to do next: Modeling user behaviors by time-LSTM." In: *International joint conferences on artificial intelligence*. 2017.

[13] Lichung Jen, Chien-Heng Chou, and Greg M Allenby. "A Bayesian approach to modeling purchase frequency". In: *Marketing letters* 14.1 (2003), pp. 5–20.

[14] George EP Box et al. *Time series analysis: Forecasting and control*. John Wiley & Sons, 2015.

[15] Warren S McCulloch and Walter Pitts. “A logical calculus of the ideas immanent in nervous activity”. In: *The bulletin of mathematical biophysics* 5.4 (1943), pp. 115–133.
[16] George Cybenko. “Approximation by superpositions of a sigmoidal function”. In: *Mathematics of control, signals and systems* 2.4 (1989), pp. 303–314.
[17] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.
[18] Sebastian Ruder. “An overview of gradient descent optimization algorithms”. In: *arXiv:1609.04747* (2016).
[19] Diederik P Kingma and Jimmy Ba. “Adam: A method for stochastic optimization”. In: *International conference on learning representations*. 2015.
[20] Ian Goodfellow et al. “Generative adversarial nets”. In: *Advances in neural information processing systems*. 2014.
[21] Tim Salimans et al. “Improved techniques for training GANs”. In: *Advances in neural information processing systems*. 2016.
[22] John F Nash et al. “Equilibrium points in n-person games”. In: *Proceedings of the national academy of sciences* 36.1 (1950), pp. 48–49.
[23] Martin Arjovsky, Soumith Chintala, and Léon Bottou. “Wasserstein GAN”. In: *International conference on machine learning*. 2017.
[24] Cédric Villani. *Optimal transport: Old and new*. Vol. 338. Springer Science & Business Media, 2008.
[25] Ishaan Gulrajani et al. “Improved training of Wasserstein GANs”. In: *Advances in neural information processing systems*. 2017.
[26] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.
[27] John Schulman et al. “Proximal policy optimization algorithms”. In: *arXiv preprint arXiv:1707.06347* (2017).
[28] Sham Kakade and John Langford. “Approximately optimal approximate reinforcement learning”. In: *International conference on machine learning*. 2002.
[29] Sham Machandranath Kakade et al. “On the sample complexity of reinforcement learning”. PhD thesis. University of London London, England, 2003.
[30] John Schulman et al. “Trust region policy optimization”. In: *International conference on machine learning*. 2015.
[31] Joshua Achiam et al. “Constrained policy optimization”. In: *International conference on machine learning*. 2017.
[32] Stephen Wright and Jorge Nocedal. “Numerical optimization”. In: *Springer Science* 35.67-68 (1999), p. 7.
[33] Volodymyr Mnih et al. “Asynchronous methods for deep reinforcement learning”. In: *International conference on machine learning*. 2016.
[34] John Schulman et al. “High-dimensional continuous control using generalized advantage estimation”. In: *arXiv preprint arXiv:1506.02438* (2015).
[35] Dean A Pomerleau. “Alvinn: An autonomous land vehicle in a neural network”. In: *Advances in neural information processing systems*. 1989.
[36] Takayuki Osa et al. “An algorithmic perspective on imitation learning”. In: *Foundations and trends in robotics*. 2018.

[37] Brian D. Ziebart et al. “Maximum entropy inverse reinforcement learning”. In: *AAAI conference on artificial intelligence*. 2008.

[38] Pieter Abbeel and Andrew Y Ng. “Apprenticeship learning via inverse reinforcement learning”. In: *International conference on machine learning*. 2004.

[39] Edwin T Jaynes. “Information theory and statistical mechanics”. In: *Physical review* 106.4 (1957), p. 620.

[40] Peter Harremoës and Flemming Topsøe. “Maximum entropy fundamentals”. In: *Entropy* 3.3 (2001), pp. 191–226.

[41] Brian D Ziebart, J Andrew Bagnell, and Anind K Dey. “Modeling interaction via the principle of maximum causal entropy”. In: *International conference on machine learning*. 2010.

[42] Chelsea Finn, Sergey Levine, and Pieter Abbeel. “Guided cost learning: Deep inverse optimal control via policy optimization”. In: *International conference on machine learning*. 2016.

[43] Umar Syed, Michael Bowling, and Robert E Schapire. “Apprenticeship learning using linear programming”. In: *International conference on machine learning*. 2008.

[44] Chelsea Finn et al. “A connection between generative adversarial networks, inverse reinforcement learning, and energy-based models”. In: *NIPS 2016 workshop on adversarial training*. 2016.

[45] Lantao Yu et al. “SeqGAN: Sequence generative adversarial nets with policy gradient”. In: *AAAI conference on artificial intelligence*. 2017.

[46] Yusuke Nakate. ”deepirl-chainer”, GitHub repository. 2019. URL: `https://github.com/uidilr/deepirl_chainer`.

A

Appendix A

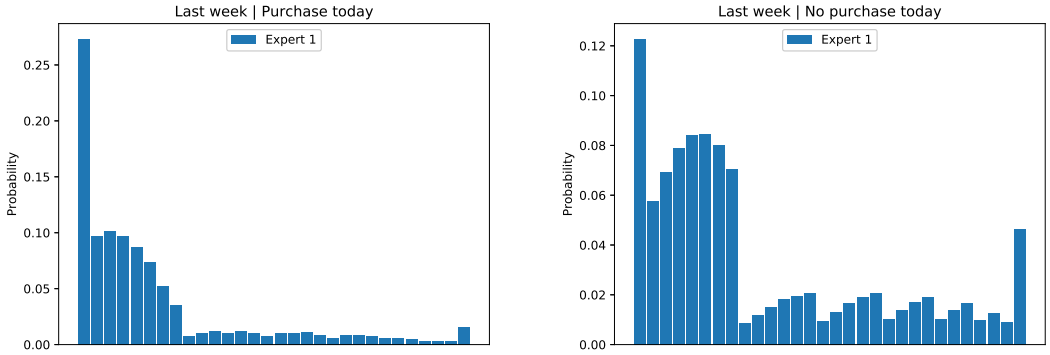
Table A.1: Hyper-parameters used in the implementations of the AIL algorithms.

Model	Parameter	Value
Generator	Size hidden layers	(64, 64)
	Adam learning rate	3e-4
	Adam epsilon	0.1
	Discount factor	0.99
	Lambda return factor	0.95
	Weight coefficient for loss of value function	1.0
	Weight coefficient for entropy bonus	0.01
	Epochs	10
	Epsilon for clipping of likelihood ratio to update policy	0.2
	Entropy coefficient	0.01
	Weight decay	0.0
	Minibatch size	73
Discriminator	Size hidden layers	(64, 64)
	Adam learning rate	1e-5
	Adam epsilon	1e-5
	Minibatch size	73
	Epochs	10

B

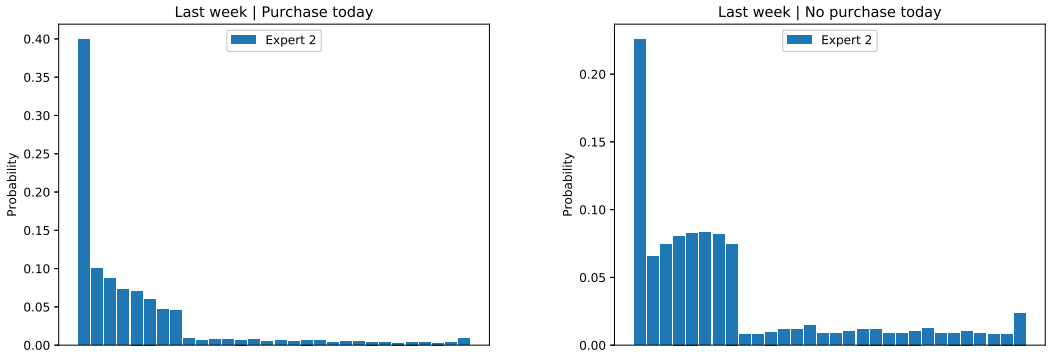
Appendix B

B.1 Expert Behaviors



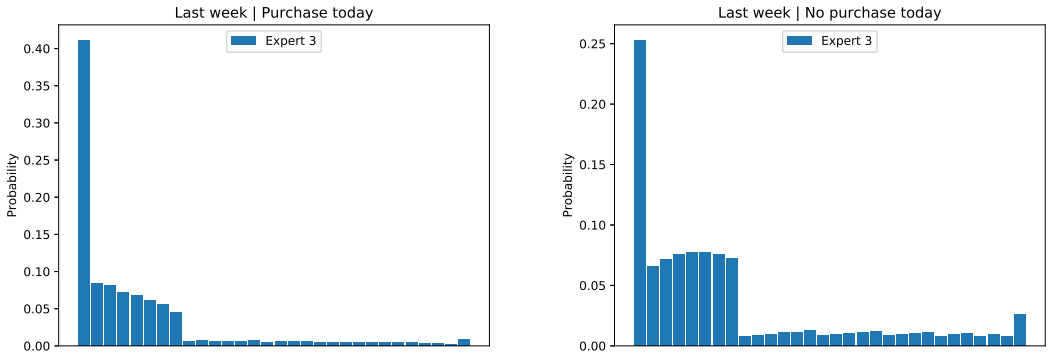
(a) Last week given purchase today. **(b)** Last week given no purchase today.

Figure B.1: The behavior of the first expert in the training data set.



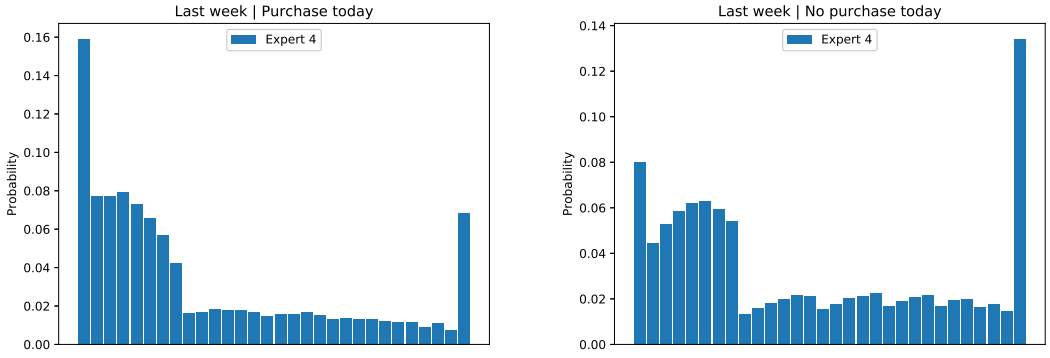
(a) Last week given purchase today. **(b)** Last week given no purchase today.

Figure B.2: The behavior of the second expert in the training data set.



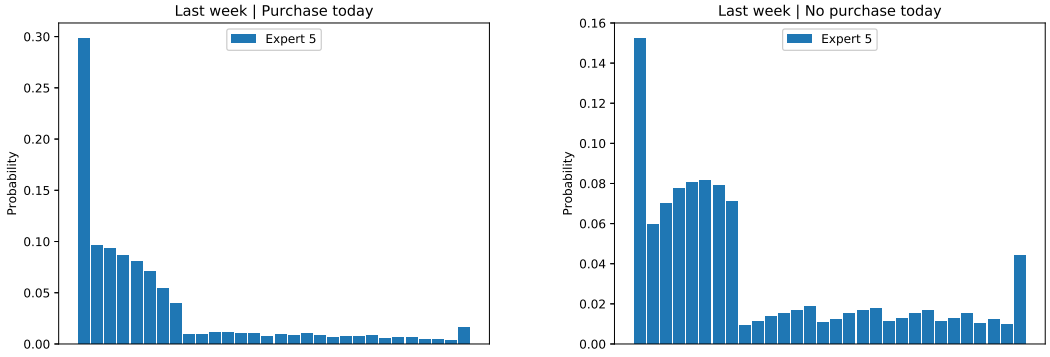
(a) Last week given purchase today. **(b)** Last week given no purchase today.

Figure B.3: The behavior of the third expert in the training data set.



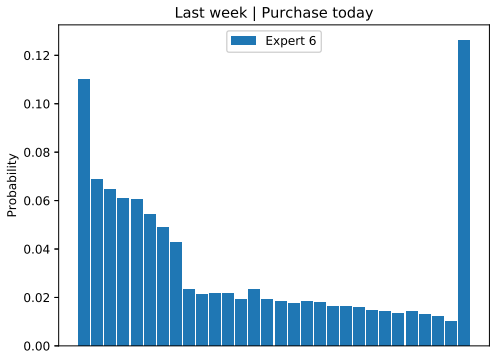
(a) Last week given purchase today. **(b)** Last week given no purchase today.

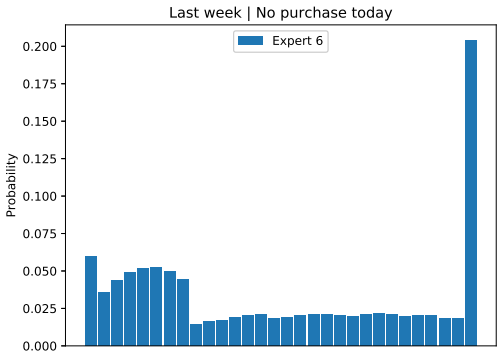
Figure B.4: The behavior of the fourth expert in the training data set.



(a) Last week given purchase today. **(b)** Last week given no purchase today.

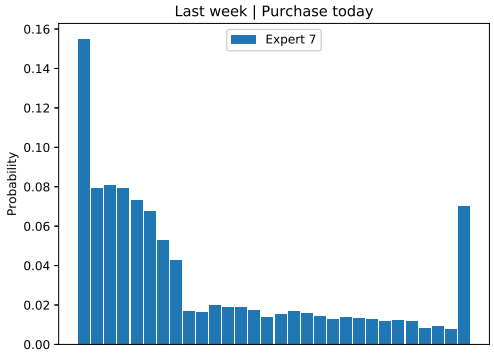
Figure B.5: The behavior of the fifth expert in the training data set.

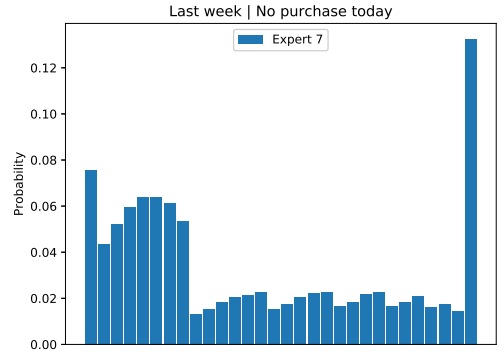




(a) Last week given purchase today. **(b)** Last week given no purchase today.

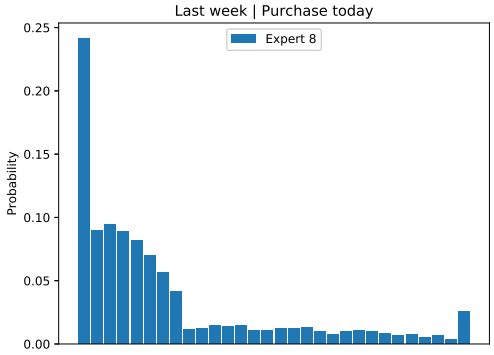
Figure B.6: The behavior of the sixth expert in the training data set.

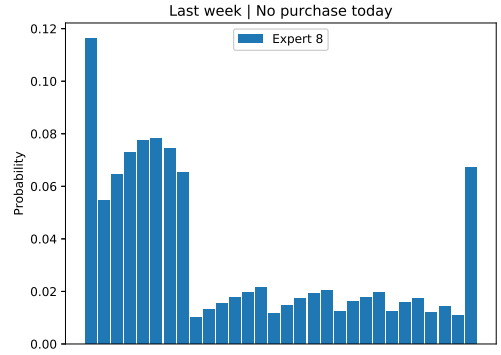




(a) Last week given purchase today. **(b)** Last week given no purchase today.

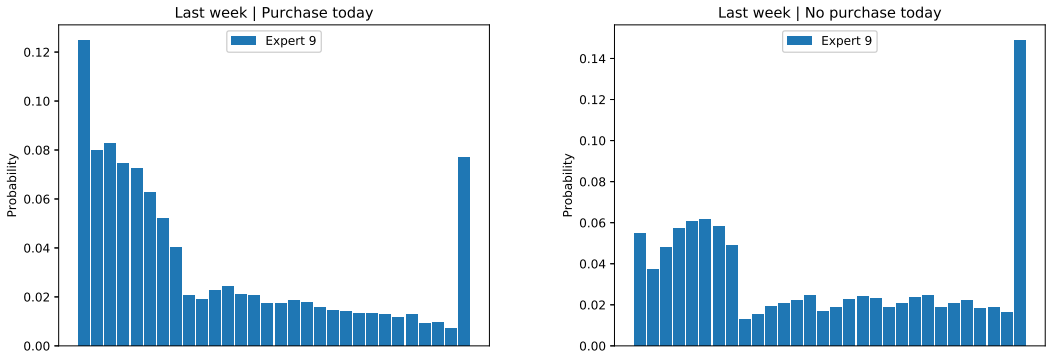
Figure B.7: The behavior of the seventh expert in the training data set.





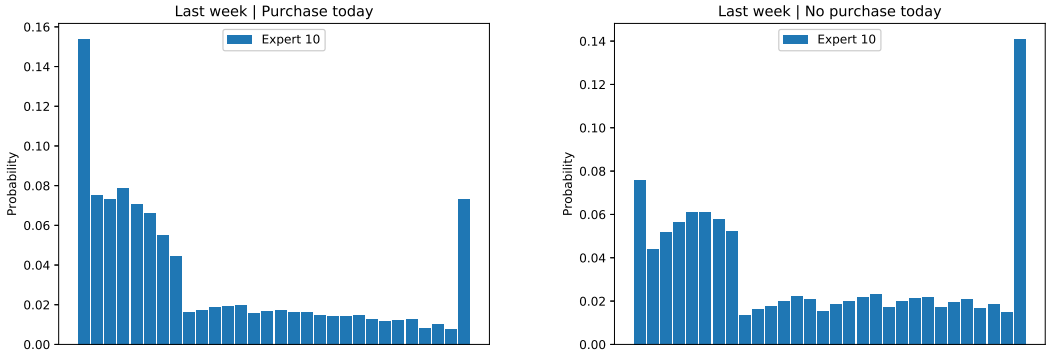
(a) Last week given purchase today. **(b)** Last week given no purchase today.

Figure B.8: The behavior of the eight expert in the training data set.



(a) Last week given purchase today. **(b)** Last week given no purchase today.

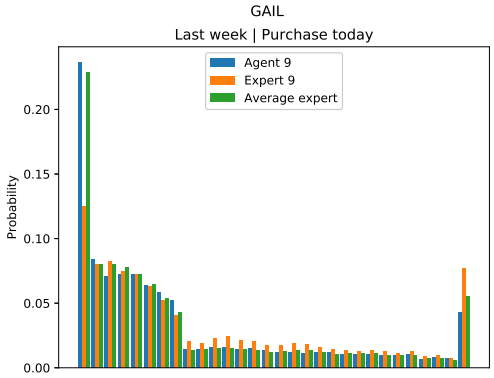
Figure B.9: The behavior of the ninth expert in the training data set.



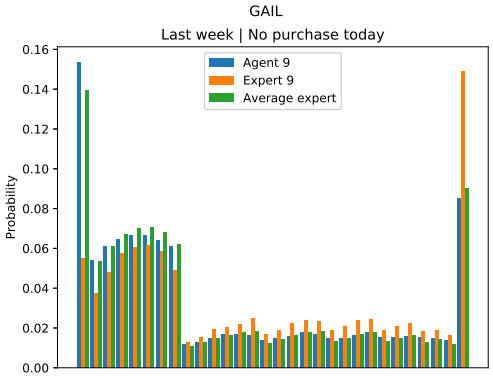
(a) Last week given purchase today. **(b)** Last week given no purchase today.

Figure B.10: The behavior of the tenth expert in the training data set.

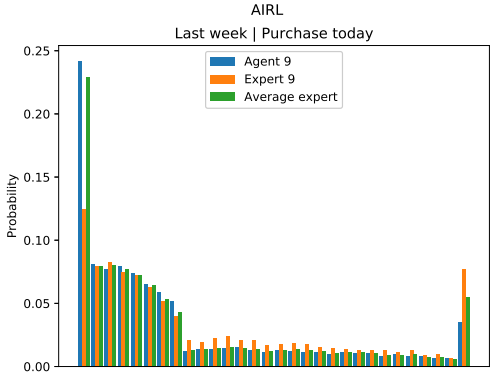
B.2 Supplementary Figures



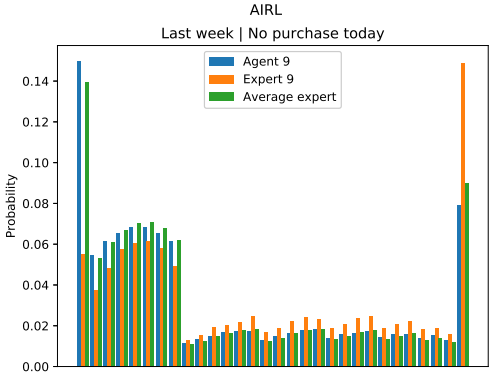
(a) GAIL.



(b) GAIL.

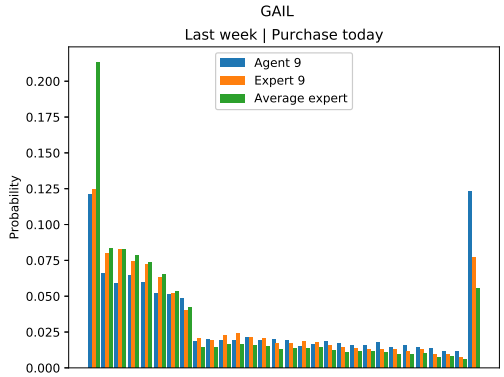


(c) AIRL.

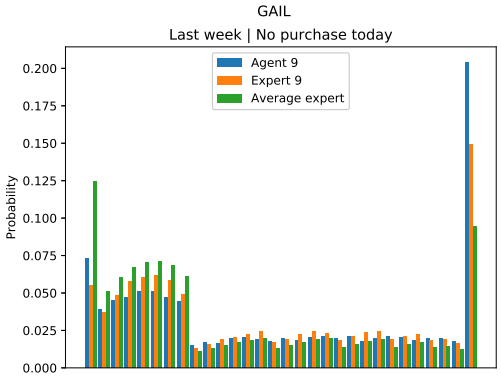


(d) AIRL.

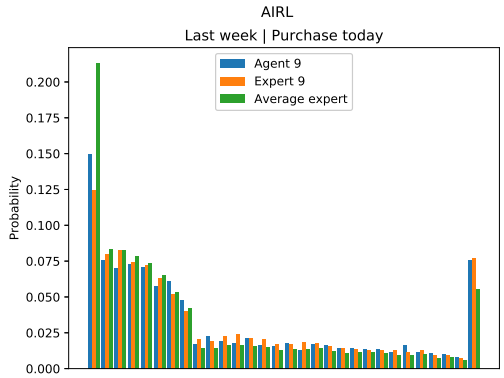
Figure B.11: A comparison between the agent (blue bars) and the ninth expert (orange bars) using a basic state representation.



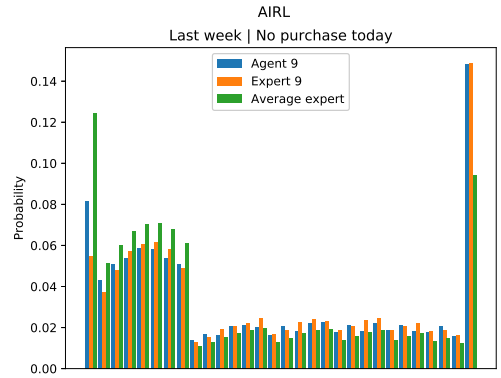
(a) GAIL.



(b) GAIL.

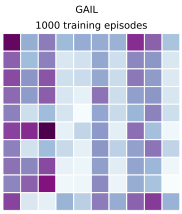


(c) AIRL.

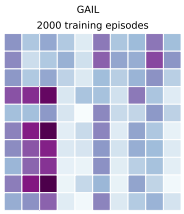


(d) AIRL.

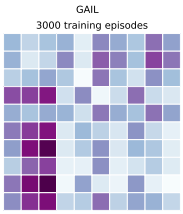
Figure B.12: A comparison between the agent (blue bars) and the ninth expert (orange bars) using one-hot encoding.



(a) 1000 episodes.

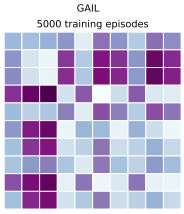


(b) 2000 episodes.

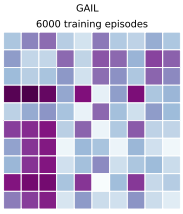


(c) 3000 episodes.

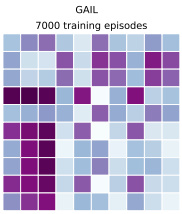
(d) 4000 episodes.



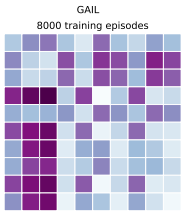
(e) 5000 episodes.



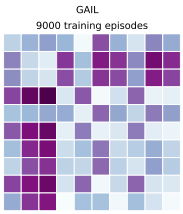
(f) 6000 episodes.



(g) 7000 episodes.

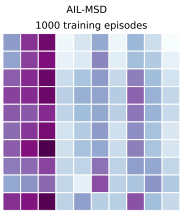


(h) 6000 episodes.

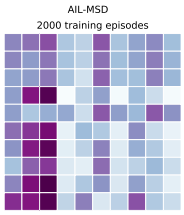


(i) 9000 episodes.

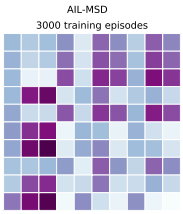
Figure B.13: An overview of the training process for GAIL using one-hot encoding. The performance is evaluated on new customers.



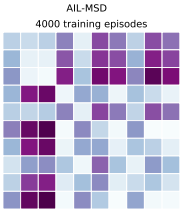
(a) 1000 episodes.



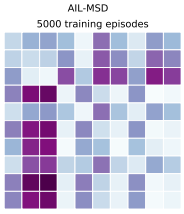
(b) 2000 episodes.



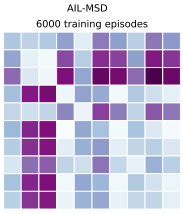
(c) 3000 episodes.



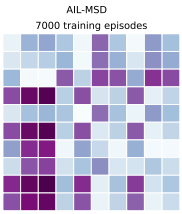
(d) 4000 episodes.



(e) 5000 episodes.

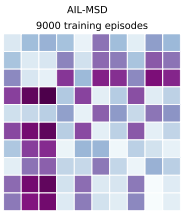


(f) 6000 episodes.



(g) 7000 episodes.

(h) 6000 episodes.



(i) 9000 episodes.

Figure B.14: An overview of the training process for MMCT-GAIL using one-hot encoding. The performance is evaluated on new customers..