Estimation with GANs

Master's Thesis

Presented to the
Department of Economics at the
Rheinische Friedrich-Wilhelms-Universität Bonn

In Partial Fulfillment of the Requirements for the Degree of
Master of Science (M.Sc.)

Supervisor: Prof. Dr. Joachim Freyberger

Submitted in September 2024 by

Marvin Benedikt Riemer

Matriculation Number: 2799234

Contents

List of Figures

List of Tables

1 Introduction

This thesis is based on the paper Kaji, Manresa, and Pouliot (2023) (KMP). My presentation on notation largy follow that paper, I. J. Goodfellow et al. (2014) and, where they relate to the Wasserstein distance, Arjovsky, Chintala, and Bottou (2017). I was inspired to investigate the Wasserstein distance by Athey et al. (2021).

2 Background

2.1 Structural estimation

Consider the problem of estimating the parameters of a structural economic model. Let $Z \sim P(Z)$ be a vector of random noise variables, Θ a parameter space and $\mathcal{X}$ a space of outcomes or observations. Then an economic model can be imagined as a function f that maps a draw from the noise Z and a parameter $\theta \in \Theta$ to an outcome $X \in \mathcal{X}$:

$$X = f(Z, \theta). \tag{1}$$

f might be quite complicated, in particular a system of equations. What makes this equation *structural* is that f is an implementation of a particular economic model together with the assumption that this model captures the true relationships between economic variables, rather than just their statistical relationships. With f therefore given by the presumed model, and Z noise, the goal is of course to find $\hat{\theta}$, a good estimate of the true parameter θ, given observations $X_1, \ldots, X_n$.

$Z \sim P(Z)$ induces a distribution $p(X_1, \ldots, X_n|\theta)$. This suggests the standard approach of maximum likelihood estimation, that is, to find $\hat{\theta}$ such that

$$\hat{\theta}_{MLE} = \arg\max_{\theta \in \Theta} \log p(X_{1,\ldots,n}|\theta). \tag{2}$$

The logarithm here is not strictly necessary; it is, however, usually taken to make the likelihood function more computationally tractable, since it does not affect the position of the maximum. However, for some more sophisticated economic models, it is not easy or even possible to give an analytical expression for the likelihood function.

This motivates further approaches, in particular simulation methods, which attempt to infer θ based on a simulation of the true data. Most notable among these is perhaps the simulated method of moments, which performs inference based on the moments of the simulated data.

The question naturally arises of how to judge whether the simulated distribution comes sufficiently close to the real distribution. This is motivates the idea of introducing a second component that provides "feedback" on this question in the form of a criterion function to be minimized. Updating the estimate $\hat{\theta}$ to minimize this criterion is the central idea behind adversarial estimation, which I will discuss in more detail in section 3.

One idea for defining the criterion is to base it on a classification, more precisely, on the probability that a given data point was drawn from the real data rather than from a simulated distribution. In machine learning, a popular tool for classification are neural networks, which I introduce next.

2.2 Neural networks

A neural network, in the most general sense, is a directed graph that defines how a certain output should be calculated from a given input. There are many different structures (also called *architectures*) of that graph. This thesis only considers so-called *feed-forward neural networs*, for which the graph consists of:

- An input layer with one node for each input variable
- An ordered set of hidden layers. The number of nodes in each is chosen by the researcher. The nodes get incoming edges only from the previous and have outgoing edges only to the successive layer
- An output layer with one node for each output variable

Embedded in any architecture is the fundamental notion of neural networks, which imagines nodes as neurons that calculate inputs and outputs according to their edges.

Definition 1 (Feed-forward neural network)**.** Let $l = 1, \ldots, L$ be an ordered set of layers, with 1 the input and L the output layer. Let I_l be the number of nodes in each layer with $b_{l,1} \ldots, b_{l,I_l}$ and $\sigma_{l,1} \ldots, \sigma_{l,I_l}$ their biases and activation functions, respectively. Let J_l be the number of incoming edges of each layer $l = 2, \cdots, L$, and let each node have at least one incoming edge. For the nodes in the input layer $v_{1,i}$, set their output equal the inputs. For the nodes $v_{l,i}, l \geq 2$, set:

$$v_{l,i} \;=\; \sigma(\sum_{j=1}^{J_l} w_{l,J} \cdot v_{(l-1),i}) \tag{3}$$

and interpret the output of the last nodes $v_{L,i}$ as the output of the network.

Then $\mathcal{N} = (V, E, \Psi)$ is called a neural network with nodes V, edges E and Ψ holds the weights, biases and activation functions.

Note that if the last layer consists only of one node taking values in $[0, 1]$, the network can be interpreted as a classifier suitable to the real-or-fake data problem introduced above.

In practice, neural networks mostly are used for machine learning applications and their parameters *trained* using a gradient descent or other optimization method. In particular, they are sometimes not trained until the optimizer converges, so it is reasonable to include a specification of the training process in the definition.

Definition 2 (Trained feed-forward neural network)**.** Let θ_{train} be a set of hyperparameters specifying the training of a neural network $\mathcal{N}$, including at least:

- A set of initial parameters Ψ_0
- A training algorithm A
- A stopping criterion

Then call $\mathcal{N}(\theta_{\text{train}}) = (V, E, \Psi(\theta_{\text{train}}))$ a feed-forward neural network trained according to θ_{train}.

In practice, Ψ_0 is usually chosen randomly Note that the training algorithm A might itself require the setting of further hyperparameters. So-called *stochastic gradient descent* algorithms are the most common training methods, of which Adam (Kingma and Ba (2017)) is the most popular choice. The stopping criterion might be either the convergence of the training algorithm, or that a certain number of training steps having been performed.

3 Adversarial estimation

The basic idea of adversarial estimation is to employ two auxiliary models, called the *generator* and the *discriminator* (or *critic*). The generator $G : \Theta \times \mathcal{Z} \to \mathcal{X}$ creates simulated data based on a guess $\hat{\theta} \in \Theta$ of the true parameter value θ_0. The discriminator $D : \mathcal{X} \to D \subset \mathbb{R}_{\geq 0}$ returns for each real and generated data point some value. This value is then used to construct the value of an objective function for the optimization of the generator, which I will call *criterion* or *loss*. This loss function can be any divergence or distance between the distributions of the real and simulated data, inlcuding functions that are directy analytically tractable and do not strictly require a separate discriminator to calculate. Assuming however the involvement of a discriminator, the estimation can be viewed as the result of the following minimax game:

$$\hat{\theta}_{adv} = \arg\min_{\theta \in \Theta} \max_{D \in \mathcal{D}} \text{loss}(D(X_i, G(\theta, Z))). \tag{4}$$

Note that this game has a unique Nash-Equilibrium, where $\hat{\theta} = \theta_0$ and $D(x) = 0.5$ for all x. The generator has no incentive to deviate, since …If every data point …

This method is a variant of *Generative Adversarial Networks* (GAN), first proposed by I. J. Goodfellow et al. (2014) (later published as I. Goodfellow et al. (2020)). There, two neural networks take the role of generator and discriminator. In particular, the discriminator is a classifier network that outputs the probability of a data point being a real rather than simulated observation. While have GANs achieved great success in image generation and related tasks, they are not directly suitable for structural estimation. One reason is that the functional form of the generator network is usually very complex, with feed-forward neural networks often being fully connected and activation functions introducing non-linearities. Relatedly, the exact architecture of a neural network is usually not chosen to be economically (or at all) interpretable, but rather as an imprecise "art" based on predictive performance. Therefore, one essential contribution of Kaji, Manresa, and Pouliot (2023) is to impose that the generator has the structure of an economic model. This model being fully specified by θ is what makes adversarial estimation meaningful and the result interpretable. They wouldn't be if θ were a long list of the weights and biases in a multi-layered neural network.

An implementation of 4 looks, generally, like algorithm 1.

Algorithm 1 Adversarial estimation

Set necessary hyperparameters and initial values
Sample real observations $X_i \sim P_0$
Sample noise $Z \sim P_Z$
while Stopping criterion does not hold **do**
    Generate fake observations from the current generator $\widetilde{X} \sim P_\theta$
    if The discriminator requires training: **then**
        Train the discriminator given the fake observations
    end if
    Calculate the criterion: $\text{loss}(D(X_i, G(\theta, Z))$
    Update $\hat{\theta}$
end while

Note that while Kaji, Manresa, and Pouliot (2023) stress the importance of sampling noise only one time, I. J. Goodfellow et al. (2014) and Athey et al. (2021) draw it anew for every training step of the discriminator and generator.

There are various ways to fill in the details of this algorithm. The stopping criterion might be a convergence criterion of the generator's optimization problem, or simply a sufficiently high number of repetitions being reached. A classification discriminator might take various forms, which I discuss below. There are two canonical choices for the loss function, which I discuss afterwards. The updates of the generator can be done with a gradient descent algorithm if it is differentiable or at least smooth enough that calculating numerical gradients will not lead an optimizer astray. Otherwise, they should be performed with a gradient-free optimization procedure, for example, a simplex algorithm.

Algorithm 1 in Kaji, Manresa, and Pouliot (2023) illustrates one way to fill out the details of 1. They use convergence as a stopping criterion, a (not necessarily trained to completion) neural network discriminator, cross-entropy loss, and update the generator using a version of the popular Adam (Kingma and Ba (2017)). Their simulation code shows another way. There, they compare a range of estimators (including neural networks trained to completion) and update the generator using a gradient-free approach.

Now I discuss some of these terms in detail.

3.1 Examples of classifier discriminators

All the following discriminators have in common that for a data point x they return a probability that it is from the real rather than the simulated data. How this probability is then turned into an objective for the generator will be discussed in the next subsection.

Recall the game-theoretic view of adversarial estimation. If the true densities p_0 and $p_\theta(x)$ are known, the discriminator has a best response depending only on these, rather than on the features of a given data point. Kaji, Manresa, and Pouliot (2023) call the discriminator playing this best response the *oracle discriminator*.

Definition 3 (Oracle discriminator)**.** The **oracle discriminator** assigns

$$D_\theta(x) \ := \ \frac{p_0(x)}{p_0(x) + p_\theta(x)} \tag{5}$$

to every $x \in$.

Of course, p_0 and $p_\theta(x)$ are unkown in practice. Nevertheless, this discriminator is useful as a benchmark in simulations and has an interesting theoretical property: If the simulated sample size $m \longrightarrow \infty$, θ_{oracle} approaches θ_{MLE}.

A simple statistical method for classification is logistic regression. In line with the simulation study in Kaji, Manresa, and Pouliot (2023), I consider a version that regresses on some collection of features of the data points and moments of the data.

Definition 4 (Logistic discriminator)**.** Let Λ be a sigmoid function with values in $(0, 1)$, and x^{mom} an $(i + j) \times k$-matrix of features and moments of the data calculated for each data point. Let $(\beta_0, \ldots, \beta_k \in \mathbb{R}^{k+1})$ be coefficients of a logistic regression run with x^{mom} as a regressor and an output vecor Y consisting of 0s and 1s for the simulated and true observations. Then the **logistic discriminator** assigns

$$D(x) \ = \ \Lambda(\beta_0 + \sum_{k=1}^{K} \beta_k x_k^{mom}) \tag{6}$$

to every $x \in$.

Note that this classifier has to be calculated anew after each update of θ. While this calculation will usually be fast on modern computers, the same is not necessarily true of the potentially more powerful neural network discriminator. Therefore, neural networks are often not trained to completion in practice and different training procedures might result in different discriminators.

Definition 5 (Neural network discriminator)**.** Let $\mathcal{N}(\theta_{train}) : \mathcal{X} \to [0, 1]$ be a classifier neural network trained according to θ_{train}. Then the **neural network discriminator** assigns

$$D(x) = \mathcal{N}(\theta_{train})(x) \tag{7}$$

to every $x \in$.

To understand more deeply the loss landscape which a neural network discriminator builds for the generator, we must consider the loss function on which it is trained.

3.2 Generator objectives

3.2.1 Jensen-Shannon divergence

The classical way to turn the probabilities $D(x)$ into an objective for the generator is the following:

Definition 6 (Cross-entropy loss)**.** The empirical cross-entropy loss (CE) is:

$$\frac{1}{n}\sum_{i=1}^{n} \log D\left(X_i\right) + \frac{1}{m}\sum_{i=1}^{m} \log\left(1 - D\left(X_{i,\theta}\right)\right).$$

A discriminator that maximizes the cross-entropy loss thereby calculates the Jensen-Shannon divergence (plus a constant) for the generator to minimize.

Theorem 7 (I. J. Goodfellow et al. (2014)). $\cdots$

A neural network discriminator that is not trained to completion returns an approximation of the Jensen-Shannon divergence. In practice, only such an approximation is often used, for two reasons: First, training a neural network to completion multiple times for every gradient calucation of the generator's optimizer can be very computationally costly, especially given that GANs in practice are often large neural networks and are applied to high-dimensional data sets. Second, an imprecise estimate of the gradient often still leads to convergence.

However, even the Jensen-Shannon divergence calculated by an optimal discriminator has a crucial disadvatage: The divergence is maximal if p_G and p_0 have disjoint support. Therefore, there are regions of the loss landscape where even the optimal CE-discriminator provides a gradient of zero in every direction at every point to the generator. If the generator "ends up" in such a region or the initial guess is there, algorithm 1 is unlikely to converge. Luckily, there are ...

3.2.2 Wasserstein-p distance

Using the so-called Wasserstein-1 distance as criterion for the generator was first proposed by Arjovsky, Chintala, and Bottou (2017).

Definition 8 (Wasserstein-p distance)**.** For two probability distributions $\mathbb{P}_0$ and $\mathbb{P}_G$, let $\Pi(\mathbb{P}_0, \mathbb{P}_G)$ be the set of all joint distributions $\gamma(x, y)$ whose marginals are $\mathbb{P}_r$ and $\mathbb{P}_g$. Then for $p \geq 1$,

$$W_p(\mathbb{P}_0, \mathbb{P}_G) = \inf_{\gamma \in \Gamma(\mathbb{P}_0, \mathbb{P}_G)} \left(\mathbb{E}_{(x,y)\sim\gamma} d(x, y)^p\right)^{1/p}$$

is the Wasserstein-p distance (also Wasserstein p-distance) between $\mathbb{P}_0$ and $\mathbb{P}_G$.

A natural interpretation of this equation comes from the field of optimal transport. It quantifies how much probability mass has to be moved how far in order to transfer $\mathbb{P}_0$ into $\mathbb{P}_G$, or vice versa, assuming that this transport is done optimally. Inspired by this image, the Wasserstein-1 distance is also called *Earth-Mover distance*.

The Wasserstein distances deliver a measure of the distance between two distributions that is strictly monotone even if they are non-overlapping. However, since they require a solution to the optimal transport problem, they can be demanding to calculate, especially in high-dimensional spaces. In the context of GANs, it is natural to consider approximating it using a neural network. To this end, the following fact is helpful:

Theorem 9 (Kantorovich-Rubinstein duality).

$$W_1(\mathbb{P}_0, \mathbb{P}_G) = \sup_{\|f\|_L \leq 1} \mathbb{E}_{x\sim\mathbb{P}_0}[f(x)] - \mathbb{E}_{x\sim\mathbb{P}_G}[f(x)] \tag{8}$$

This dual representation of the Wasserstein-1 distance paves the way to approximating it using a neural network. The network estimates the function f and is often called critic instead of discriminator since it does not return a probability anymore. Unfortunately, it is not trivial to regularize a neural network to obey the Lipschitz constraint. Arjovsky, Chintala, and Bottou (2017) clamp the weights of the neural network to lie in a compact space. They themselves describe this approach as "clearly terrible", since there is no principled way to chose the clipping parameter and setting it too big or too small comes with difficult trade-offs.

Gulrajani et al. (2017) propose to penalize the norm of the gradient of the critic with respect to its input. This solution has been more widely accepted and is also used by Athey et al. (2021).

Of course, it is also possible to approximate the Wasserstein-1 distance without training a neural network. Since the Wasserstein distance is the solution to an optimal transport problem, it can be derived using a "Pseudo-auction algorithm". For differentiability, this algorithm can than be approximated using a smoothed "soft-auction" algorithm. The divergence resulting from this algorithm is called the *Sinkhorn divergence.*

3.3 Theoretical properties

Kaji, Manresa, and Pouliot (2023) contains three main theoretical results. In line with the focus of their paper, they are stated for the case with the (estimated) Jensen-Shannon distance as a criterion.

3.3.1 Theorems in I. J. Goodfellow et al. (2014)

3.3.2 Theorems in Kaji, Manresa, and Pouliot (2023)

The first states that for some reasonable conditions on the true and estimated criterion functions, which discuss below, the adversarial estimator is indeed consistent.

Theorem 10 (Theorem 1, KMP). *Suppose*

(1) *For every open* $G \in \Theta : inf_{\theta \notin G} \mathbb{M}_\theta(D_\theta) > \mathbb{M}_{\theta_0}(D_{\theta_0})$,

(2) $\{\log D_\theta : \theta \in \Theta\}$ *and* $\{\log D_\theta \circ T_\theta : \theta \in \Theta\}$ *are* P_0- *and* P_Z-*Glivenko-Cantelli, respectively,*

(3) $\sup_{\theta \in \Theta} \|\mathbb{M}_\theta(\hat{D}_\theta) - \mathbb{M}_\theta(D_\theta)\| \to 0$ *in probability, and*

(4) $\hat{\theta}$ *satisfies* $\mathbb{M}_{\hat{\theta}}(\hat{D}_{\hat{\theta}}) \leq \inf_{\theta \in \Theta} \mathbb{M}_\theta(\hat{D}_{\hat{\theta}}) + o^*_P(1) \to 0.$

Then $h(\hat{\theta}, \theta_0) \to 0.$

Condition 1 is that the optimum θ_0 is identified by the true criterion. Condition 2 is that the generator and discriminator terms of the cross-entropy criterion converge to the true distributions P_0 and P_G, respectively. Condition 3 is that the estimated criterion converges uniformly to the true criterion in the entire parameter space. Condition 4 is that the estimated criterions of estimates $\hat{\theta}$ converge to the estimands θ at rate 1 in outer probability.

The second theorem states that the estimator converges at a rate of $O^*_P(n^{-1/2})$ and requires multiple assumption. The first requires, roughly speaking, that the generator is parametric, well-behaved in the parameters, can "stably" be inverted, and that the inverted generator is also parametric and well-behaved in the parameters.

Assumption 1 (A1, KMP). *(1)* *Θ is (a subset of) a Euclidean space*

(2) *p_θ is differentiable in θ at every $\theta \in \Theta$ for every $x \in \mathcal{X}$ with the derivative continuous in both x and θ;*

(3) *The maximum eigenvalue of the Fisher information $I_\theta = P_\theta \dot{\ell}_\theta \dot{\theta}_\theta^\top$ is bounded uniformly in $\theta \in \Theta$*

(4) *the minimum eigenvalue of I_θ is bounded away from 0 uniformly in $\theta \in \Theta$.*

The same is assumed for the "inverted" structural model $\tilde{\mathcal{P}}_\theta = \{((p_0/p_\theta) \circ T_\theta)\, p_Z : \theta \in \Theta\}$.

The second condition is on the growing synthetic sample size, and, barring computational constraints, can easily be guaranteed by the researcher.

Assumption 2 (A2, KMP). $n/m \to 0$

The third assumption has two parts: The first is that the estimated criterion converges to its minimum for the true θ at rate $o_P^*(n^-1)$. The second part, which the authors call "orthogonality", is that the derivative of the estimated loss converges to that of the oracle. It can be empirically checked by plotting cross-sections of the loss around the original value, as the authors and I do in the simulation part.

Assumption 3 (A3, KMP). *There exists a sequence of open balls $G_n := \{\theta \in \Theta : h(\theta, \theta_0) < \eta_n\}$ such that*

(1) $\eta_n \sqrt{n} \to \infty, \mathbb{M}_{\hat{\theta}}\left(\hat{D}_{\hat{\theta}}\right) \leq \inf_{\theta \in G_n} \mathbb{M}_\theta\left(\hat{D}_\theta\right) + o_P^*\left(n^{-1}\right)$*, and*

(2) $\inf_{\theta \in G_n}\left[\mathbb{M}_{\hat{\theta}}\left(\hat{D}_{\hat{\theta}}\right) - \mathbb{M}_\theta\left(\hat{D}_\theta\right)\right] - \left[\mathbb{M}_{\hat{\theta}}\left(D_{\hat{\theta}}\right) - \mathbb{M}_\theta\left(D_\theta\right)\right] = o_P^*\left(n^{-1}\right)$..

The fourth assumption imposes two requirements to aid identification: That the true loss has approximately quadratic curvature near the optimum and that P_θ and P_{θ_0} overlap.

Assumption 4 (A4, KMP). *(1)* *There exists open $G \subset \Theta \subset \mathbb{R}^k$ containing θ_0 in which $M_\theta\left(D_\theta\right) - M_{\theta_0}\left(D_{\theta_0}\right) \gtrsim h\left(\theta, \theta_0\right)^2$.*

(2) $h\left(\theta, \theta_0\right)^2 = O\left(\int D_{\theta_0}\left(\sqrt{p_{\theta_0}} - \sqrt{p_\theta}\right)^2\right)$ *as $\theta \to \theta_0$.*

Together, these yield:

Theorem 11 (Theorem 2, KMP). *Under Assumptions 1 to* 4, $h\left(\hat{\theta}, \theta_0\right) = O_P^*\left(n^{-1/2}\right)$.

Regarding the efficiency of the estimator, consider this fifth assumption, that imposes the twice differentiability of the likelihood of the generator.

Assumption 5 (A5, KMP). *(1) The parameter space Θ is (a subset of) a Euclidean space $\mathbb{R}^k$.*

(2) The structural model $\{P_\theta : \theta \in \Theta\}$ has a likelihood that is twice differentiable in θ at θ_0 for every $x \in \mathcal{X}$ with the derivatives continuous in both x and θ.

(3) The Fisher information matrix $I_{\theta_0} := P_{\theta_0}\dot{\ell}_{\theta_0}\dot{\ell}_{\theta_0}^\top = -P_{\theta_0}\ddot{e}_{\theta_0}$ and the matrix $\tilde{I}_{\theta_0} := 2P_{\theta_0}\left(D_{\theta_0}\dot{e}_{\theta_0}\dot{\ell}_{\theta_0}^\top + \left(\ddot{\ell}_{\theta_0} + \ \dot{\ell}_{\theta_0}\dot{\theta}_{\theta_0}^\top\right)\log\left(1 - D_{\theta_0}\right)\right)$ are positive definite.

(4) T_θ is continuously differentiable in θ for every $x \in \mathcal{X}$ and P_0 has a likelihood that is continuously differentiable in x.

Point 3 requires a quadratic shape of the criterion, similar to Assumption 4 above.

Now Kaji, Manresa, and Pouliot (2023) arrive at their third theorem. It states the asymptotic distribution towards which the adversarial estimator weakly converges.

Theorem 12 (Theorm 3, KMP). *Under the conclusion of Theorem 2 and Assumptions 2, 3, and 5,*

$$\sqrt{n}\left(\hat{\theta} - \theta_0\right) \ = \ 2\tilde{I}_{\theta_0}^{-1}\sqrt{n}\left[\mathbb{P}_0\left(1 - D_{\theta_0}\right)\dot{\ell}_{\theta_0} - \mathbb{P}_{\theta_0}D_{\theta_0}\dot{\ell}_{\theta_0} - \tilde{\mathbb{P}}_0\tau_n\right] + o_P^*(1) \ \rightsquigarrow \ N\left(0, \tilde{I}_{\theta_0}^{-1}V\tilde{I}_{\theta_0}^{-1}\right) \tag{9}$$

where $V := \lim_{n\to\infty} 4P_{\theta_0}D_{\theta_0}\left(1 - D_{\theta_0}\right)\dot{\ell}_{\theta_0}\dot{\ell}_{\theta_0}^\top$.

Efficiency requires correct specification of the generator model.

Assumption 6 (A6, KMP). *The synthetic model $\{P_\theta : \theta \in \Theta\}$ is correctly specified, that is, $P_{\theta_0} = P_0$ and $D_{\theta_0} \equiv 1/2$.*

This yields

Theorem 13 (Corollary 4, KMP). *Under the conclusion of Theorem 3 and Assumption 6,* $\sqrt{n}\left(\hat{\theta} - \theta_0\right) = I_{\theta_0}^{-1}\sqrt{n}\left(\mathbb{P}_0 - \mathbb{P}_{\theta_0}\right)\dot{\ell}_{\theta_0} + o_P^*(1) \rightsquigarrow N\left(0, I_{\theta_0}^{-1}\right)$.

3.3.3 Applicability to simulations

Kaji, Manresa, and Pouliot (2023) partially discuss the applicability of the theoretical results to their simulation study. It seems plausible that the requirements of Theorem 10 are fulfilled by at least one their discriminators, and indeed their results show convergence and amicable loss landscapes. The discrete choice nature of the Roy model (which will be introduced below) seems problematic for the results on convergence and efficieny, which depend on the generator.

First, it is not obvious that it can be inverted, as required by Assumption 1. Second, because d_1 and d_2 are (crucial parts) of the observation, the generator is not differentiable and therefore not twice differentiable along x, as required by Assumption 5.

Regarding my own simulations, of course the unchanged Roy model generator will preserve these difficulties. Focusing on the Sinkhorn approximation of the Wasserstein discriminator, it is beyond the scope of this thesis to check whether the results above can be transfered to this case. However, consider the simple example of learning only a location parameter. The population Wasserstein-1 distance will not show quadratic curvature near the optimum, violating Assumptions 4 and 5. Therefore, the Wasserstein-2 loss seems more promising to uphold the theoretical results.

4 Simulation study

The authors simulate the estimation of the Roy model, a discrete choice model which has intractable likelihood for certain parameter values.

4.1 The Roy model

The Roy model models a set of agents choosing which sector to work in in each of two time periods. At the start of the game, nature determines the (natural logarithms of the) wages offered to each agent, by the following formulas:

$$\log w_{i1s} = \mu_s + \varepsilon_{i1s} \tag{10}$$

$$\log w_{i2s} = \mu_s + \gamma \mathbb{1}_{d_{i1}=s} + \varepsilon_{i2s} \tag{11}$$

where the noise of the offered wages is distributed as follows:

$$\begin{bmatrix} \varepsilon_{i11} \\ \varepsilon_{i12} \\ \varepsilon_{i21} \\ \varepsilon_{i22} \end{bmatrix} \sim N\left(\begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} \sigma_1^2 & \rho_s\sigma_1\sigma_2 & \rho_t\sigma_1^2 & \rho_s\rho_t\sigma_1\sigma_2 \\ \rho_s\sigma_1\sigma_2 & \sigma_2^2 & \rho_s\rho_t\sigma_1\sigma_2 & \rho_t\sigma_2^2 \\ \rho_t\sigma_1^2 & \rho_s\rho_t\sigma_1\sigma_2 & \sigma_1^2 & \rho_s\sigma_1\sigma_2 \\ \rho_s\rho_t\sigma_1\sigma_2 & \rho_t\sigma_2^2 & \rho_s\sigma_1\sigma_2 & \sigma_2^2 \end{bmatrix}\right). \tag{12}$$

In the first time period, each agent i observes the wages $\log w_{i11}$ and $\log w_{i12}$ offered to them in the two sectors. Knowing their own discount factor β and the parameters γ_1 and γ_2, they solve the dynamic progamming problem and pick a sector for the first period. In the second period, they the wages $\log w_{i21}$ and $\log w_{i12}$ are revealed to them and they pick a sector.

The researcher observes the realized wages $\log w_1$ and $\log w_2$ as well as the corresponding sector choices $d_1, d_2 \in 1, 2$. Broadly speaking, the parameters μ_1, μ_2, and to a lesser degree γ_1

and γ_2, are location parameters of the distributions of offered wages, while σ_1, σ_2, ρ_s and ρ_t determine the shape and correlations (shapes of the joint distributions).

Note the selection effects resulting from only realized wages being observed. In particular, if μ_i is sufficiently below μ_j, sector i will never be picked and μ_i will not be identified.

If $\rho_t = 0$, a likelihood function for observations from the Roy model is available.

4.2 General simulation structure

First, I reproduce parts of the authors' simulation in the scientific Python stack, more precisely, using the packages `numpy`, `scipy`, and `scikit-learn` (Harris et al. (2020), Virtanen et al. (2020), and Pedregosa et al. (2011), respectively).

I implement the Sinkhorn divergence as an approximation of the Wasserstein distance using the `SampleLoss` function from the `GeomLoss` (Feydy et al. (2019)) package. It is also optimized for running on GPUs by virtue of being built on `KeOps` (Charlier et al. (2021)). `GeomLoss` is built on `PyTorch` (Ansel et al. (2024)), which I therefore integrate into the scientific Python stack for my simulation.

For my scientific Python code, I use the `mp` module from Python's standard library to parallelize simulation runs on an HPC cluster (cf. A). I generate plots using `Matplotlib` (Hunter (2007)). My code is available somewhere.

The replication package of Kaji, Manresa, and Pouliot (2023) can be downloaded from the journal website. It contains the authors' simulation code, written in Matlab. As the authors state in the readme file, the simulations for the Roy model are contained in the files `main_roy.m` (Figures 6, 7) and `main_case.m` (Figures 8, 9, and Table I). They draw on functions in other files to simulate data and calculate losses.

Both main files share a general structure: after setting parameters of the simulation itself (for example sample sizes, number of simulation runs) and the Roy model, the values of loss functions are calculated along a linear grid and then rendered to created Figures 6 and 8. The grid describes seven- or eight-dimensional "cross-hairs", where one parameters is varied while the others remain fixed at the true value. Thereafter, real and fake observations are generated and an initial guess in generated. Then, the Roy model is estimated using multiple methods, which are implemented as constrained minimizations.The constraints are bounds on the parameters of the Roy model, on which the authors do not futher elaborate, but which are likely added for computational efficiency. Where necessary, an additional nonlinear constraint enforces that the guesses of the minimizer stay within the support of the Roy model.

Algorithm 2 Initial guess in main_roy.m

Input: True parameter vector θ, lower and upper bounds L, U, the support of the Roy model $\mathcal{S}$

while $\theta_{0,p} \notin \mathcal{S}$, for each parameter p of θ to be estimated: **do**

  Sample noise $u_p \sim \mathcal{N}(0, 0.2)$

  Set $\theta_{0,p} := \theta_p + u_p$

  Clip $\theta_{0,p} := min(max(\theta_{0,p}, L_p), U_p)$

end while

In `main_roy.m`, the authors plot cross-sections of the loss landscapes generated by the Oracle and neural network discriminator as well as the loss implied by MLE. Then, they generate the initial guess θ_0 using algorithm 2. Following simple maximum likelihood estimation with θ_0 as an initial value, they perform adversarial estimation with:

- the Oracle discriminator, using the result of MLE as initial value
- the Logistic discriminator, using the result of the previous step as initial value
- the neural network discriminator, again using the result of the Oracle step as initial value

They also estimate the Roy model using Indirect Inference and optimally-weighted SMM, but don't use the results.

`Main_case.m` simulates the untractable likelihood case. Therefore, an initial guess θ_0 is drawn similarly algorithm 2, but the first draw is used. Logistic regression is then perfomed using θ_0 as an initial value, and the result used to initialized adversarial estimation. Again, loss-curves for the neural network and logistic discriminator are plotted, including for ρ_t. To study the properties of the adversarial estimators, Kaji, Manresa, and Pouliot (2023) then perform the bootstrap, sampling with replacement from the noise and the true observations independently. They perform estimation with the logistic discriminator using the previous logistic estimate as initial value. The result of this serves as the initial value for estimation with the neural network discriminator, as well as for the Indirect Inference and optimally-weighted SMM estimators (but the authors don't report the Indirect Inference results in the paper).

4.3 Implementation details

4.3.1 Initial values

As mentioned above, Kaji, Manresa, and Pouliot (2023) use results of estimation procedures as initial guesses for other estimation procedures. This is unproblematic in the sense that an estimator will hopefully converege to the optimum independent of its starting value and therefore the final distribution of estimates should be largely independent of the initializations. However, about their Figure 7, they write:"The resulting estimators are comparable with MLE", referring to the Oracle and the neural network discriminators. This is unsurprising, perhaps even disappointing, given that the theoretically optimal Oracle estimator is initialized with the result of MLE and the neural network estimator with the result of the Oracle estimation.

This issue is less pronounced for their Figure 9, because the setting of initial values does not involve the impossible-in-practice Oracle discriminator. However, it is still not surprsing that the neural network estimators results are "comparable" to the logistic estimator with which it is initialized.

In the `README.pdf` of the replication package, Kaji, Manresa, and Pouliot (2023) propose to start adversarial the adversarial esitmation by pre-estimation with a logistic discriminator. Their Figures 7, 9, and table I should perhaps rather be seen as indicating neural network adversarial estimation does not bring large improvements after this first step in the Roy model case.

To achieve a proper comparision, I reproduce their choice of initial values in my simulation of the Wasserstein discriminator. I will leave out the pre-estimation in my simulation of initial values drawn from wider intervals, because then I'll be interested in the more pure properties of the different criterion functions.

4.3.2 Discriminators

I reproduce the MLE and Oracle discriminator by translating the `logroypdf.m` function that Kaji, Manresa, and Pouliot (2023) have provided for the case with ρ_t known to be zero. Kaji, Manresa, and Pouliot (2023) use two logistic regression discriminators. The first in `main_case.m`, is:

$$D_{\text{loss1}}(x) \ : \ \Lambda(\boldsymbol{\beta}^\top \boldsymbol{x}^{\boldsymbol{mom}}) \tag{13}$$

with $\boldsymbol{\beta} = (\beta_0, \ldots, \beta_7)$ and $\boldsymbol{x}^{\boldsymbol{mom}} = (1, \log w_1, d_1, \log w_2, d_2, (\log w_1)^2, (\log w_2)^2)$. As illustrated in Figure 8 of the working paper version (compare also section 4.4), ρ_t is not identified using this discriminator. Therefore, they add the cross-moment between $(\log w_1)$ and $(\log w_2)$ in `main_case.m`:

$$D_{\text{loss2}}(x) \ : \ \Lambda(\boldsymbol{\beta}^\top \boldsymbol{x}^{\boldsymbol{mom}}) \tag{14}$$

with $\boldsymbol{\beta} = (\beta_0, \ldots, \beta_8)$ and $\boldsymbol{x}^{\boldsymbol{mom}} = (1, \log w_1, d_1, \log w_2, d_2, (\log w_1)^2, (\log w_2)^2), \log w_1 \cdot \log w_2$. I employ `sklearn.linear_model.LogisticRegression` for both logistic regression discriminators.

The authors' code for the neural network discriminator is in `NND.m`. It uses Matlab's `patternnet` and `train`. The scientific Python stack comes with limited support for neural networks, but I can sufficiently approximate the authors' discriminator using `sklearn.neural_network.MLPClassifier`.

Following the authors, I create a net with 1 hidden layer containing 10 nodes, followed by the tanh activation function. Inspecting sklearn's source code reveals that a logistic output activation function is automatically set. The authors train their network with a conjugate gradient descent algorithm. Because this is not available to train `MLPClassifier`, I use the Adam algorithm (Kingma and Ba (2017)).

`MLPClassifier`'s default convergence criteria cause my code to raise warnings about non-convergence of the discriminator nets. This is not completely mitigated even by setting `max_iter` (the maximum number of iterations of the optimizer) to 2000 (10 times the default value), at the cost of a longer runtime. Nevertheless, the networks converge well enough under the default settings. Leaving `max_iter` at 200, but increasing `tol`, the tolerance of the convergence criterium, five- or tenfold mitigates the warnings but results in flatter and less smooth loss functions. Therefore, I leave the default settings and accept the warnings.

The authors also set the normalization and regularization parameters of `patternnet`. Since these are handled differently in `MLPClassifier`, I do not translate this adaption.

Figures 1 and 2 in section 4.4 suggest that these modifcations do not perceptively alter the loss landscape.

4.3.3 Generator

Both true and simulated observations from the roy model are generated by drawing uniform noise, transforming it into the multinormally distributed shocks $(\varepsilon_{i11}, \varepsilon_{i12}, \varepsilon_{i21}, \varepsilon_{i22})$ and then, based on some given θ, calculating the decisions of the agents. The authors provide an option to smooth the observations, but do not use it, since the loss crosssections (cf. section 4.4) look sufficiently smooth and the estimations work nevertheless.

For the outer optimization loop that trains the generator, the authors use the third-party `fminsearchcon` function (D'Errico (2024)). This is a wrapper function that adds support for bounds and nonlinear constraints to Matlab's built-in `fminsearch`, which employs the Nelder-Mead simplex algorithm (Lagarias et al. (1998)) to minimize a function without computing gradients. I employ `scipy.optimize.minimize`, which natively supports the Nelder-Mead algorithm with bounds and nonlinear constraints. I set an option to perform a version of the Nelder-Mead algorithm that's adapted to the dimensionality of the problem (Gao and Han (2012)), which shows improved convergence in my simulation.

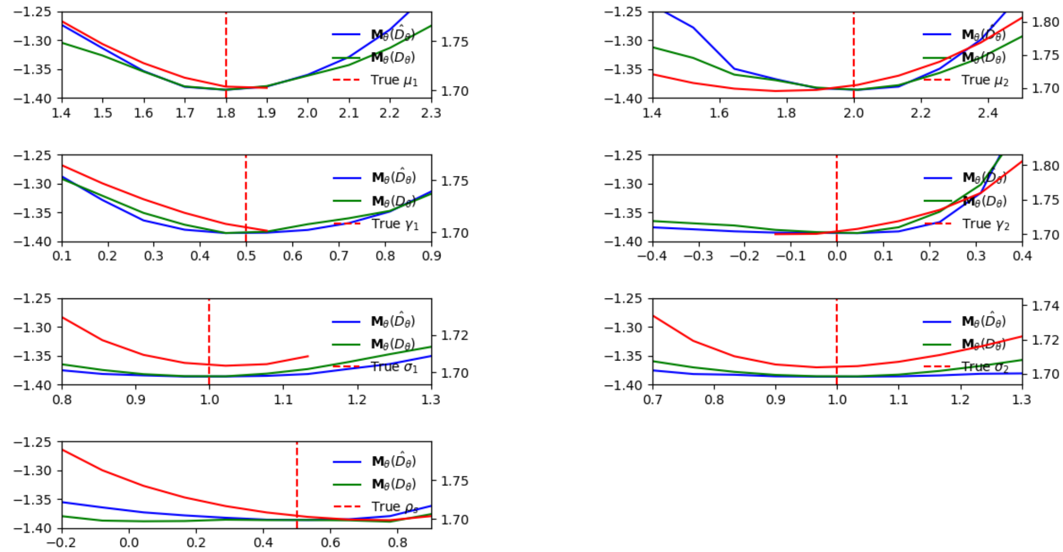

Figure 1. Replication of Figure 6 in Kaji, Manresa, and Pouliot (2023)

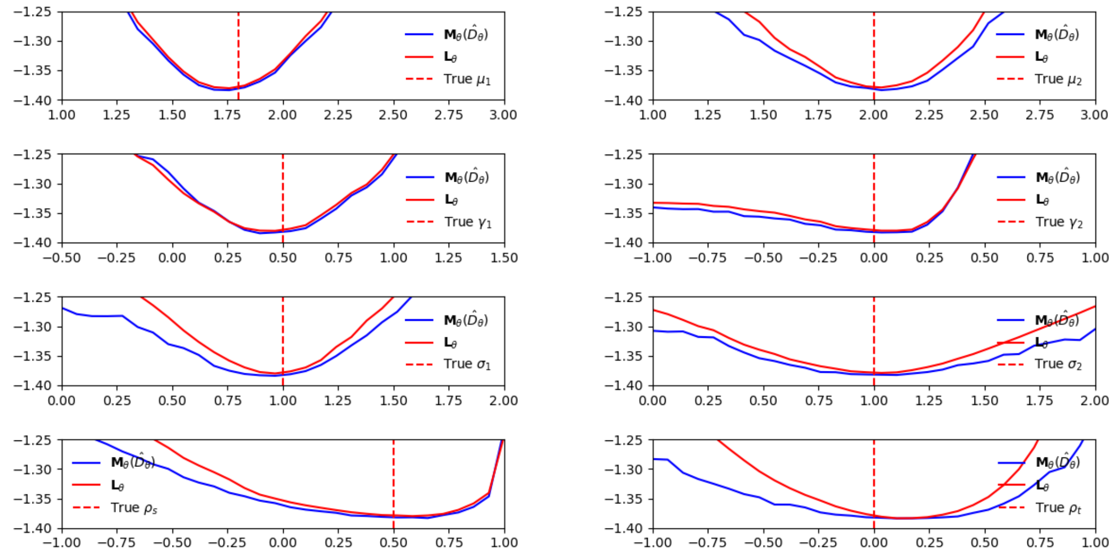

Figure 2. Replication of Figure 8 in Kaji, Manresa, and Pouliot (2023)

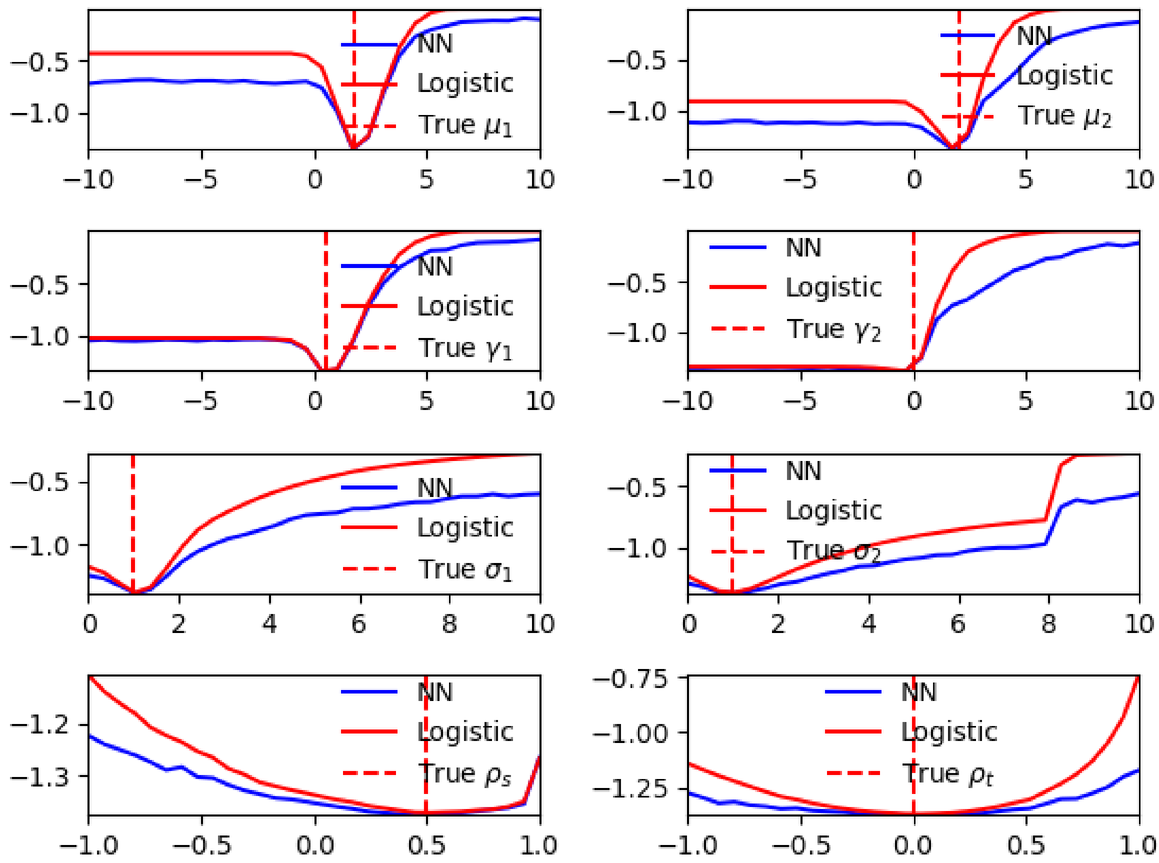

Figure 3. Losses cross-sections plotted over wider intervals

4.4 Cross-sections of the loss landscape

Figures 1 and 2 match Figures 6 and 8 in Kaji, Manresa, and Pouliot (2023), confirming that my replication in the scientific Python stack, including of the neural network discriminator, is sufficiently close to the original.

Figure 3 shows a variant of figure 2 plotted over wider intervals. It is striking that for some parameters, the loss is flat when moving too far away from the optimal value. Partly, this can be explained by the discrete choice nature of the Roy model. Consider for example μ_1. If it becomes too small relative to μ_2 while γ_1 and γ_2 are held constant, the agents stop choosing sector 1. Since only their chosen sectors and wages are observed, in such cases there are no observations that help to narrow down the value of μ_1 (except to bound it from above). Similar arguments explain the flatness towards the tail of the cross-hairs for all four location parameters μ_1, μ_2, γ_1, and γ_2.

Recall from section 3.2.1 that there is another reason for the loss-function to become flat, at at least for the neural network estimator with cross-entropy loss. Namely, the constant Jensen-Shannon divergence for disjoint distributions. To isolate the effect, I rotate part of the cross-hairs

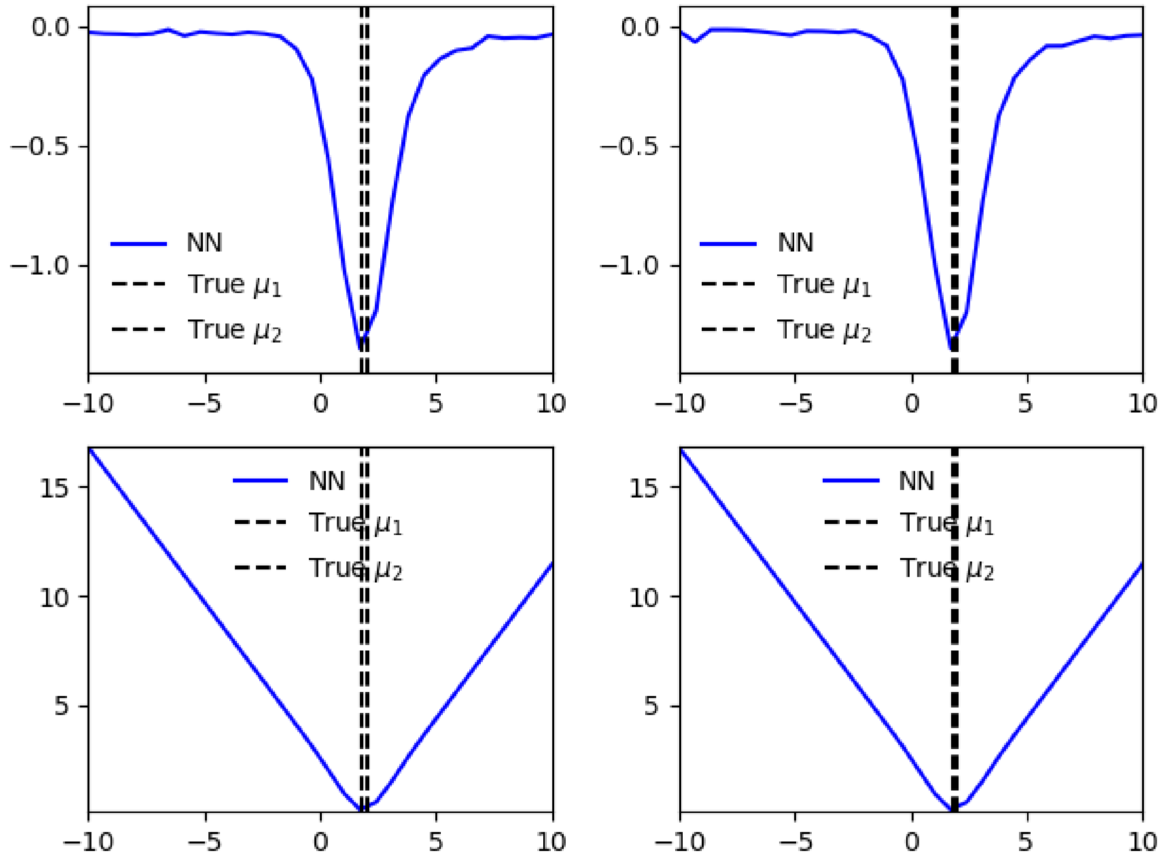

Figure 4. Diagonal loss cross-sections. Rows: Jensen-Shannon divergence, Wasserstein-1. Columns: $\text{diag}_{1,2}$, $\text{diag}_{1,-2}$.

Table 1. Parameter estimates

	μ_1	μ_2	γ_1	γ_2	σ_1	σ_2	ρ_s	ρ_t
Wasserstein-1-discriminator	1.81	1.91	0.58	0.06	0.92	1.01	0.43	0.06
	(0.14)	(0.12)	(0.08)	(0.02)	(0.04)	(0.10)	(0.12)	(0.03)
W1 uniform init	-0.97	-0.74	-1.31	-2.36	1.78	2.05	-0.10	-0.05
	(4.48)	(4.19)	(4.72)	(4.95)	(1.88)	(1.99)	(0.67)	(0.53)

to look at the loss along the diagonal $\text{diag}_{1,2} = \{(\mu_1, \mu_2) = (m, m); m \in \mathbb{R}\}$. This way, the fake distribution can become disjoint from the real distribution without affecting the agents' choices. [1]

Figure 4 shows the results: In the top row, a neural network discriminator approximates the JS divergence. In the bottom row, the Wasserstein-1 distance as approximated by the Sinkhorn divergence is plotted. The left column shows clearly that the Jensen-Shannon divergence becomes constant when m is small or big enough that the distributions become disjoint, while the Wasserstein-1 distance provides constant gradients. The right column plots the diagonal $\text{diag}_{1,-2} = \{(\mu_1, -\mu_2) = (m, m); m \in \mathbb{R}\}$: disjointness is also realized in this case for large values of m.

4.5 Estimation

I reproduce the simulations in section 3.2.2 of Kaji, Manresa, and Pouliot (2023) using the approximate Wasserstein-1 estimator implemented in `geomloss.SamplesLoss` and 200 bootstrap samples. The first row of 1 shows the results. The point estmimates are always at least as close to the true parameter values as those reported in Table I of Kaji, Manresa, and Pouliot (2023). The standard errors are also comparable or tighter except for μ_1 and μ_2.

Figure 5 visualizes the distribution of estimates.

The Wasserstein-1 estimator should have an advantage when starting far away from the true value. To verify this, I run a simulation that draws initial values over a uniform distribution from a wider set of parameter bounds. The results are summarized in the second row of table 1 and plotted in figure 6. The numerical comparision shows that they are much less precise than the pre-estimated estimates. However, checking the histograms reveals that the mode of the estimate is clearly distinct close to the true value for the location parameters and the variance. There is also a mode, albeit less clear, near the true ρ_t, but ρ_s does not get estimated at all. In

1. There is a small distortion because $\mu_1 = 1.8 \neq 2.0 = \mu_2$.

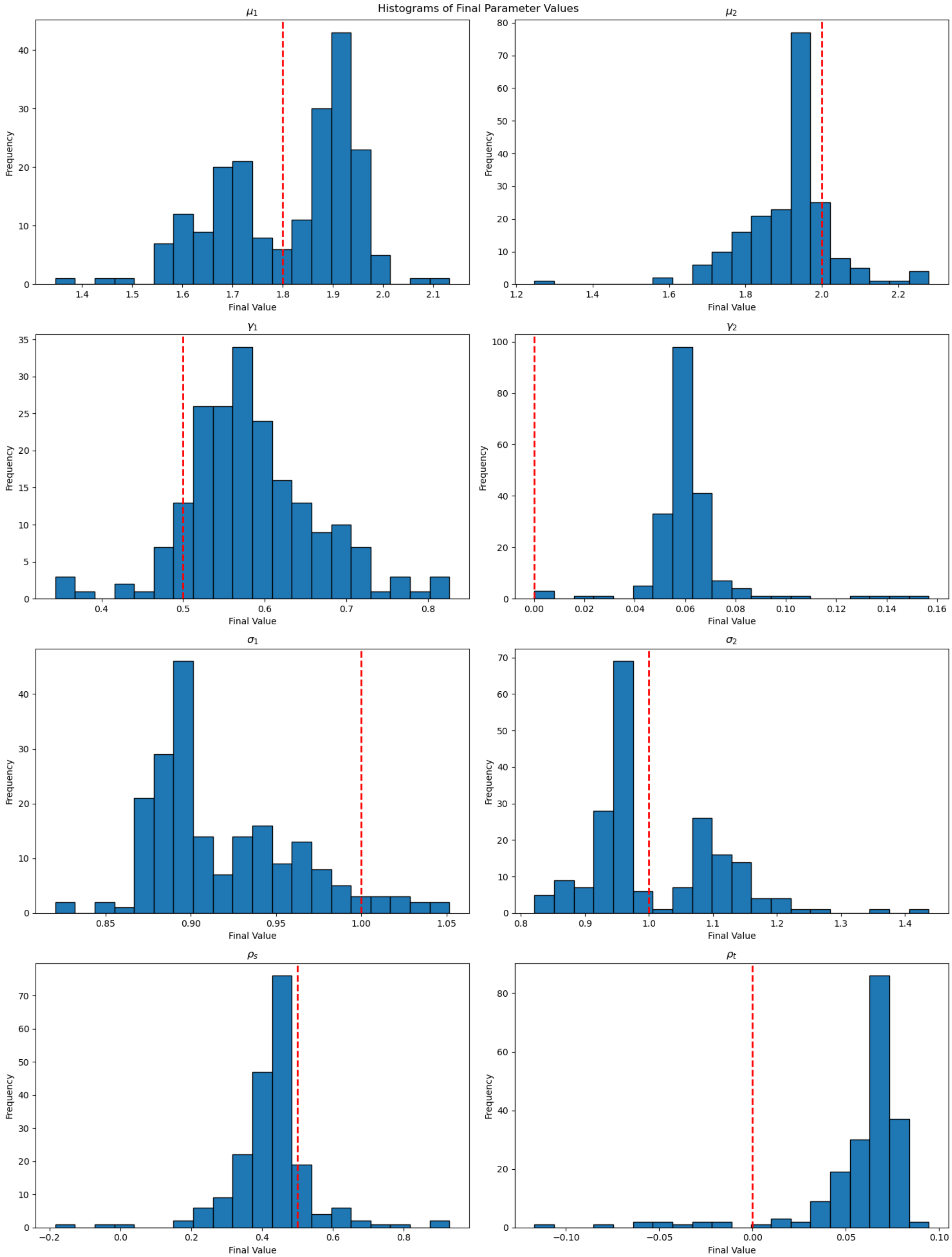

Figure 5. Results of 200 bootstrap estimations with Wasserstein-1 loss

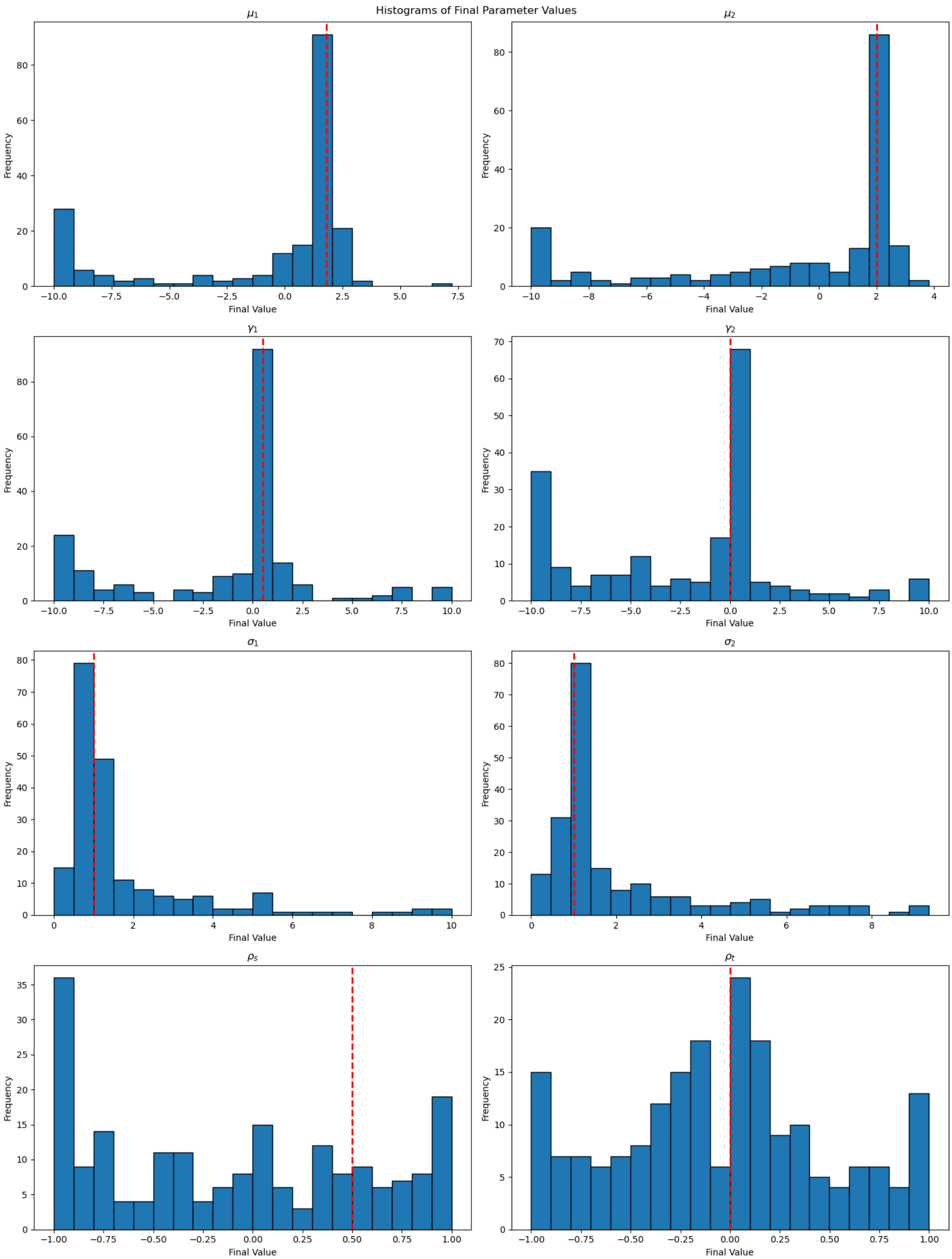

Figure 6. Results of 200 estimations with Wasserstein-1 loss and initial values that are uniform over broad intervals

all cases, the distribution of estimates is quite wide, and in some cases, a significant amount is at the bounds of the considered intervals.

Comparision to authors' methods to follow…

5 Conclusion

This section concludes.

Appendix A Acknowledgement of system use

The author gratefully acknowledges the granted access to the Marvin cluster hosted by the University of Bonn.

References

Ansel, Jason, Edward Yang, Horace He, Natalia Gimelshein, Animesh Jain, Michael Voznesensky, Bin Bao, et al. 2024. "PyTorch 2: Faster Machine Learning Through Dynamic Python Bytecode Transformation and Graph Compilation." In *29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2 (ASPLOS '24).* ACM. https://doi.org/10.1145/3620665.3640366. [12]

Arjovsky, Martin, Soumith Chintala, and Léon Bottou. 2017. *Wasserstein GAN.* arXiv: 1701.07875 `[stat.ML]`. [1, 7]

Athey, Susan, Guido W Imbens, Jonas Metzger, and Evan Munro. 2021. "Using Wasserstein Generative Adversarial Networks for the design of Monte Carlo simulations." *Journal of Econometrics,* 105076. [1, 4, 8]

Charlier, Benjamin, Jean Feydy, Joan Alexis Glaunès, François-David Collin, and Ghislain Durif. 2021. "Kernel Operations on the GPU, with Autodiff, without Memory Overflows." *Journal of Machine Learning Research* 22 (74): 1–6. http://jmlr.org/papers/v22/20-275.html. [12]

D'Errico, John. 2024. *fminsearchbnd, fminsearchcon.* https://www.mathworks.com/matlabcentral/fileexchange/8277-fminsearchbnd-fminsearchcon. MATLAB Central File Exchange. Accessed September 12, 2024. [15]

Feydy, Jean, Thibault Séjourné, François-Xavier Vialard, Shun-ichi Amari, Alain Trouve, and Gabriel Peyré. 2019. "Interpolating between Optimal Transport and MMD using Sinkhorn Divergences." In *The 22nd International Conference on Artificial Intelligence and Statistics,* 2681–90. [12]

Gao, Fuchang, and Lixing Han. 2012. "Implementing the Nelder-Mead simplex algorithm with adaptive parameters." *Computational Optimization and Applications* 51 (1): 259–77. [15]

Goodfellow, Ian, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2020. "Generative adversarial networks." *Communications of the ACM* 63 (11): 139–44. [4]

Goodfellow, Ian J., Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. *Generative Adversarial Networks.* eprint: arXiv:1406.2661. [1, 4, 6, 8]

Gulrajani, Ishaan, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron Courville. 2017. *Improved Training of Wasserstein GANs.* arXiv: 1704.00028 `[cs.LG]`. [8]

Harris, Charles R., K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, et al. 2020. "Array programming with NumPy." *Nature* 585 (7825): 357–62. https://doi.org/10.1038/s41586-020-2649-2. [12]

Hunter, J. D. 2007. "Matplotlib: A 2D graphics environment." *Computing in Science & Engineering* 9 (3): 90–95. https://doi.org/10.1109/MCSE.2007.55. [12]

Kaji, Tetsuya, Elena Manresa, and Guillaume Pouliot. 2023. "An adversarial approach to structural estimation." *Econometrica* 91 (6): 2041–63. [1, 4, 5, 8, 10, 12–14, 16, 17, 19]

Kingma, Diederik P., and Jimmy Ba. 2017. *Adam: A Method for Stochastic Optimization.* arXiv: 1412.6980 [cs.LG]. [3, 5, 15]

Lagarias, Jeffrey C, James A Reeds, Margaret H Wright, and Paul E Wright. 1998. "Convergence properties of the Nelder–Mead simplex method in low dimensions." *SIAM Journal on optimization* 9 (1): 112–47. [15]

Pedregosa, F., G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, et al. 2011. "Scikit-learn: Machine Learning in Python." *Journal of Machine Learning Research* 12: 2825–30. [12]

Virtanen, Pauli, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, et al. 2020. "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python." *Nature Methods* 17: 261–72. https://doi.org/10.1038/s41592-019-0686-2. [12]

Selbstständigkeitserklärung

Ich versichere hiermit, dass ich die vorstehende Masterarbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe, dass die vorgelegte Arbeit noch an keiner anderen Hochschule zur Prüfung vorgelegt wurde und dass sie weder ganz noch in Teilen bereits veröffentlicht wurde. Wörtliche Zitate und Stellen, die anderen Werken dem Sinn nach entnommen sind, habe ich in jedem einzelnen Fall kenntlich gemacht.

28. September 2024

Marvin Benedikt Riemer