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).

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 ranodm 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 set of parameters $\theta \in \Theta$ to an outcome $X \in \mathcal{X}$:

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

f might be quite complicated, in particular and as suggested by the index notation, a system of equations. What makes the estimation *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.

$Z \sim p(Z)$ induces a distribution $p(X_{1,\dots,n}|\theta)$. This suggests the standard approach of maximum likelihood estimation, that is, to find $\hat{\theta}$ such that

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

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 minimzed. 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 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 are connected by incoming edges only the the previous and by outgoing edges only to the succesive 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 $l = 1, \dots, L$ 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 $b_{l,1} \dots, b_{l,I_l}$ and $\sigma_{l,1} \dots, \sigma_{l,I_l}$ their biases and activation functions, respectively. Let J_l the number of incoming edges of each layer $l = 2, \cdots, L$, and let each node have at least one 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 consist 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 usually used for machine learning applications and their paramters *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 $\mathcal{N}$, inlcuding at least:

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

Then call $\mathcal{N}(\theta_{train}) = (V, E, \Psi(\theta_{train}))$ 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* are the most common training methods, of which Adam (Diederik (2014)) is the most popular choice. The stopping criterion might be either the convergence of the training algorithm, or the a certain number of training steps having been performed.

3 Adversarial estimation

The basic idea of adversarial estimation is to structure the parameter estimation around two auxiliary models, called the *generator* and the *discriminator* (or *critic*). The generator $G : \Theta \times \mathcal{Z} \rightarrow \mathcal{X}$ creates simulated data based on a guess of the true parameter value $\hat{\theta} \in \Theta$. Given the real and the simulated data, the discriminator returns objective function for the generator, which I will call *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. In the classic GAN case involving neural networks, the loss function is the result of the discriminator solving a maximization problem:

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

Note that this game has a clear Nash-Equilibrium

This method is a variant of “Generative Adversarial Networks”, 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 and instead of estimating a parameter vector, noise is transformed into some output, such as an image. While 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 nodes being fully connected and activation functions being used. 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. It 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
while Stopping criterion does not hold do
    Generate fake observations $\widetilde{X}$ from the current generator
    Train the discriminator given the fake observations
    Calculate the loss
    Update $\hat{\theta}$
end while
```

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. The 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.

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 (Diederik (2014)). 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 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 unique Nash equilibrium from .... If the true densities p_0 and $p_\theta(x)$ are known, we get 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 $x \in$.

Kaji, Manresa, and Pouliot (2023) call this the *oracle discriminator*. Of course, p_0 and $p_\theta(x)$ are unkown in practice. Also, this discriminator is only optimal in the Nash equilibrium as off-equilibrium, it neglects to update from the prior proabilities p_0 and $p_\theta(x)$. Nevertheless, it 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 we different training procedures might result in different discriminators.

Definition 5 (Neural network discriminator)**.** Let $\mathcal{N}(\theta_{train}) : \mathcal{X} \longrightarrow [0, 1]$ be a classifier neural networked 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, the 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 $\ldots$

3.2.2 Wasserstein-p distance

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

Definition 8 (Wasserstein-p distance)**.** For two probability distribution $\mathbb{P}_0$ 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 $p \geq 1$,

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

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).

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

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 result. 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 Kaji, Manresa, and Pouliot (2023)

The first states that for some reasonable conditions on the oracle and estimated loss, the adversarial estimator is indeed consistent.

The second states that it convergeges at a rate of $O_P^*(n^{-1/2})$ and requires the following assumptions: First, that the generator is parametric and well-behaved in the parameters. The second condition, that m diverges faster than n, can easily be guaranteed by the researcher.

The third assumption has two parts: The first is that the estimted 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.

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.

With the conclusion of this, again requiring Assumptions 2 and 3, and a fifth assumption on the (twice) differentiability of the structural model, Kaji, Manresa, and Pouliot (2023) arrive at their third theorem. It states the asymptotic distribution towards which the adversarial estimator weakly converges.

In fact, assuming correct specification, a corollary states that the generator is efficient.

3.3.2 Applicability to simulations

4 Simulation

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 chosing 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:

$$\log w_{i1s} \;=\; \mu_s + \varepsilon_{i1s} \log w_{i2s} \;=\; \mu_s + \gamma \nVdash d_{i1} \;=\; s + \varepsilon_{i2s} \tag{9}$$

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

$$\begin{bmatrix} \varepsilon_{i11} \\ \varepsilon_{i12} \\ \varepsilon_{i21} \\ \varepsilon_{i22} \end{bmatrix} \sim N\left(\begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} \sigma_1^2 & \rho_s\sigma_1\sigma_2 & \rho_t\sigma_1^2 & \rho_s\rho_t\sigma_1\sigma_2 \\ \rho_s\sigma_1\sigma_2 & \sigma_2^2 & \rho_s\rho_t\sigma_1\sigma_2 & \rho_t\sigma_2^2 \\ \rho_t\sigma_1^2 & \rho_s\rho_t\sigma_1\sigma_2 & \sigma_1^2 & \rho_s\sigma_1\sigma_2 \\ \rho_s\rho_t\sigma_1\sigma_2 & \rho_t\sigma_2^2 & \rho_s\sigma_1\sigma_2 & \sigma_2^2 \end{bmatrix} \right). \tag{10}$$

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 $\log w_2$ and 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 corrrelations (shapes of the joint distributions).

Note the selection effects resulting from only realized wages being observed. In particular, if μ_i is suffiently 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). My code is available

I program another simulation using `PyTorch` (Ansel et al. (2024)), a popular and highly developed neural network library for Python. Among the advantages of Pytorch is that it offers support for training neural networks on GPUs Additionally, the library `GeomLoss` (Feydy et al. (2019)) builds on PyTorch. Its `SampleLoss` function provides the Sinkhorn approximations of the Wasserstein-1 and -2 distances. It is also optimized for running on GPUs by virtue of being built on `KeOps` (Charlier et al. (2021)).

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)).

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 (e.g. 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 intial guess in generated. Then, the the Roy model is estimated using multiple methods, which are implemented as a 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 $\theta$, 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 $\theta$ 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 intial guess θ_0 using 2. Following simple maximum likelihood estimation with θ_0 as an intial value, they perform adversarial estimation with:

- the Oracle discriminator, using the result of MLE as intial 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 intial value

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

`Main_case.m` simulates 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 intial value, and the result used to intialized 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 initial value for estimation with the neural network discriminator, as well as for the Indirect Inference and optimally-weighted SMM estimators.

4.3 Implementation details

4.3.1 Discriminators

Kaji, Manresa, and Pouliot (2023) use two

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. For the logistic regression discriminators, I employ `sklearn.linear_model.LogisticRegression`.

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. Because the conjugate-gradient descent algorithm is not available to train `MLPClassifier`, I use the Adam algorithm (Diederik (2014)). It is popular for training neural networks and achieves comparable results in my case.

`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.

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

My simulations show that these modifications do not significantly alter the shape of the loss curves.

4.3.2 Generator

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

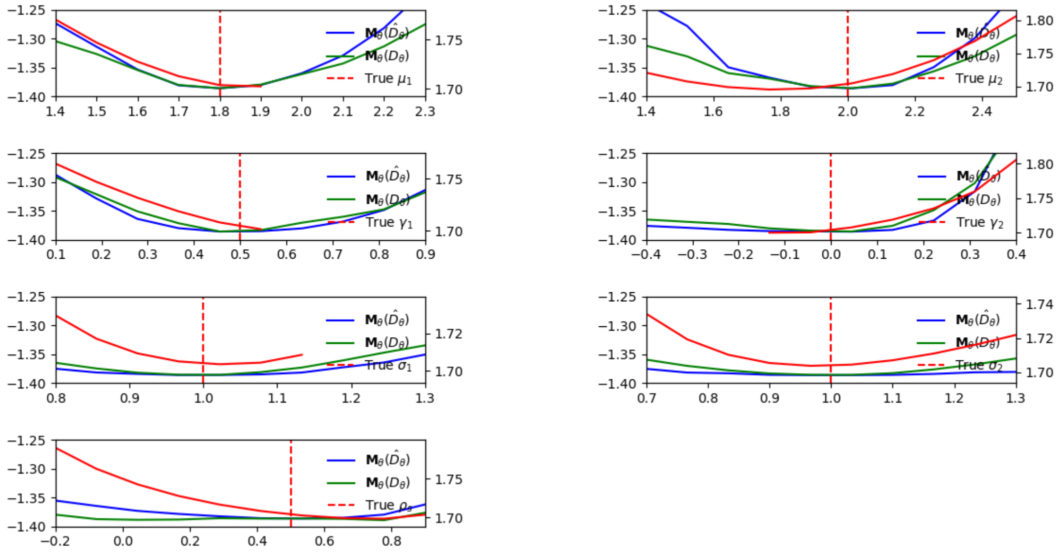

Figure 1. Replication of Figure 6 in Kaji, Manresa, and Pouliot (2023)

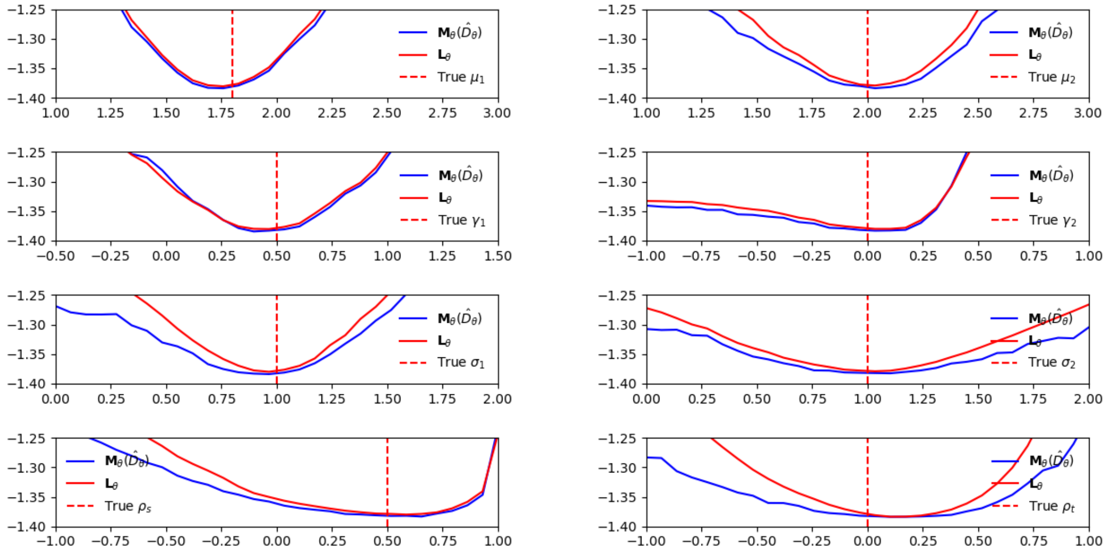

Figure 2. Replication of Figure 8 in Kaji, Manresa, and Pouliot (2023)

Nelder-Mead algorithm that's adapted to the dimensionality of the problem (Gao and Han (2012)), which shows improved convergence in my simulation.

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 2 plotted over wider intervals. It is striking that for some parameters, the loss is flat when moving to far away from the optimal value. Partly, this can

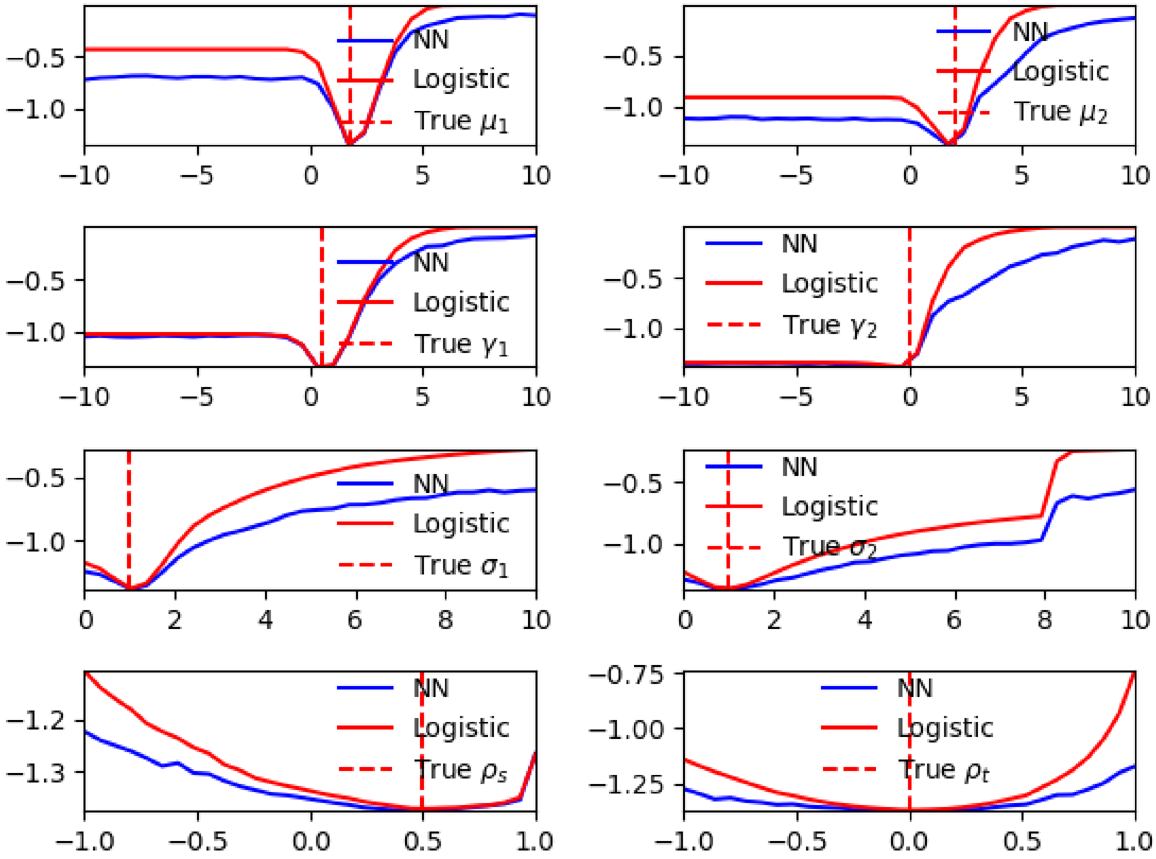

Figure 3. Losses cross-sections plotted over wider intervals

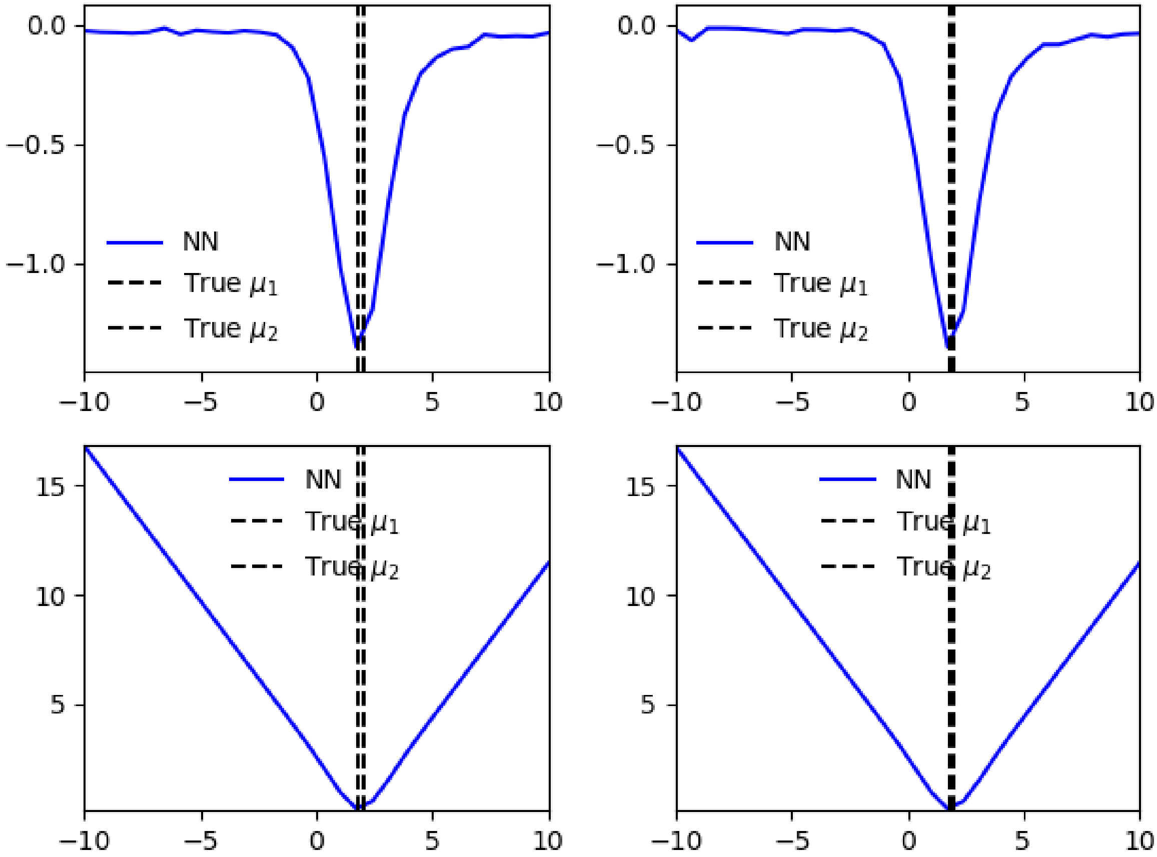

Figure 4. Diagonal loss cross-sections. Rows: Jensen-Shannon divergence, Wasserstein-1. Columns: (μ_1, μ_2), $(\mu_1, -\mu_2)$.

be explained by the discrete choice nature of the Roy model. Consider for example μ_1. If it becomes to small relative to mu_2 while γ_1 and γ_2 are held constant, the agents stop chosing sector 1. Since only their chosen sectors and wages are observed, in such cases there are no observations that help to narrow down the vale 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 to look at the loss along the diagonal $\{(\mu_1, \mu_2) = (m, m); m \in \mathbb{R}\}$. This way, the fake distribution can become disjoint from the real distribution without affecting the agents' choices.

[1]

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

Table 1. Parameter estimates

	μ_1	μ_2	γ_1	γ_2	σ_1
Wasserstein-1-discriminator	1.81 (0.14)	1.91 (0.12)	0.58 (0.08)	0.06 (0.02)	0.92 (0.04
Wasserstein-1-discriminator with uniform initialization on wide intervals	-0.97 (4.48)	-0.74 (4.19)	-1.31 (4.72)	-2.36 (4.95)	1.78 (1.88

Figure 4 shows the results: In the top row, a neural network discriminator approximates the JS divergence. The 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 (μ_1, μ_2) is small or big enough that the distributions become disjoint, while the Wasserstein-1 distance provides constant gradients. The right column plots the diagonal $\{(\mu_1, -\mu_2) = (m, m); m \in \mathbb{R}\}$: disjointness is also realized in this case for large absolute values of (μ_1, μ_2).

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 bottstrap 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.

5 Conclusion

This section concludes.

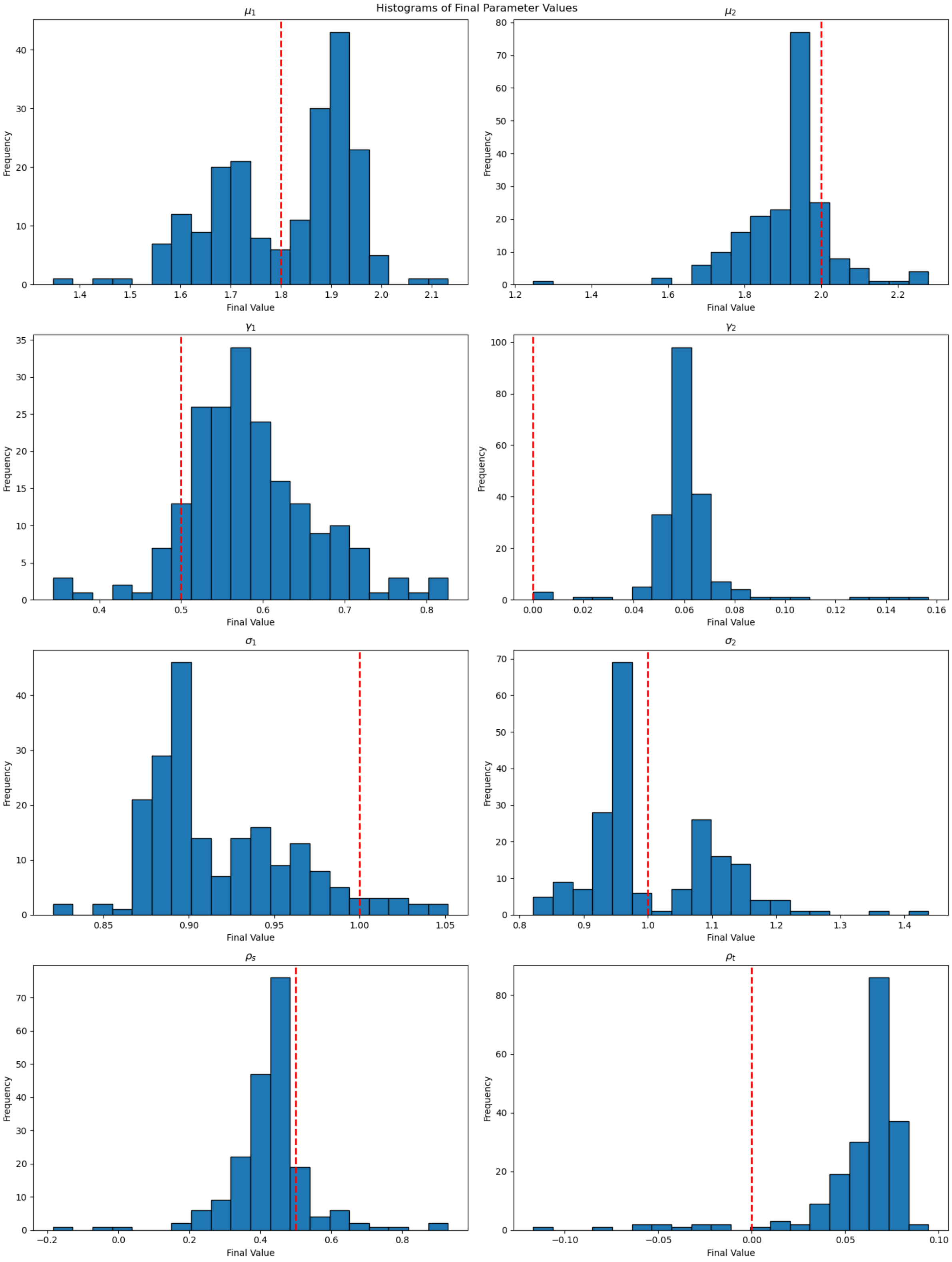

Figure 5. Results of 200 bootstrap estimations with Wasserstein-1 loss

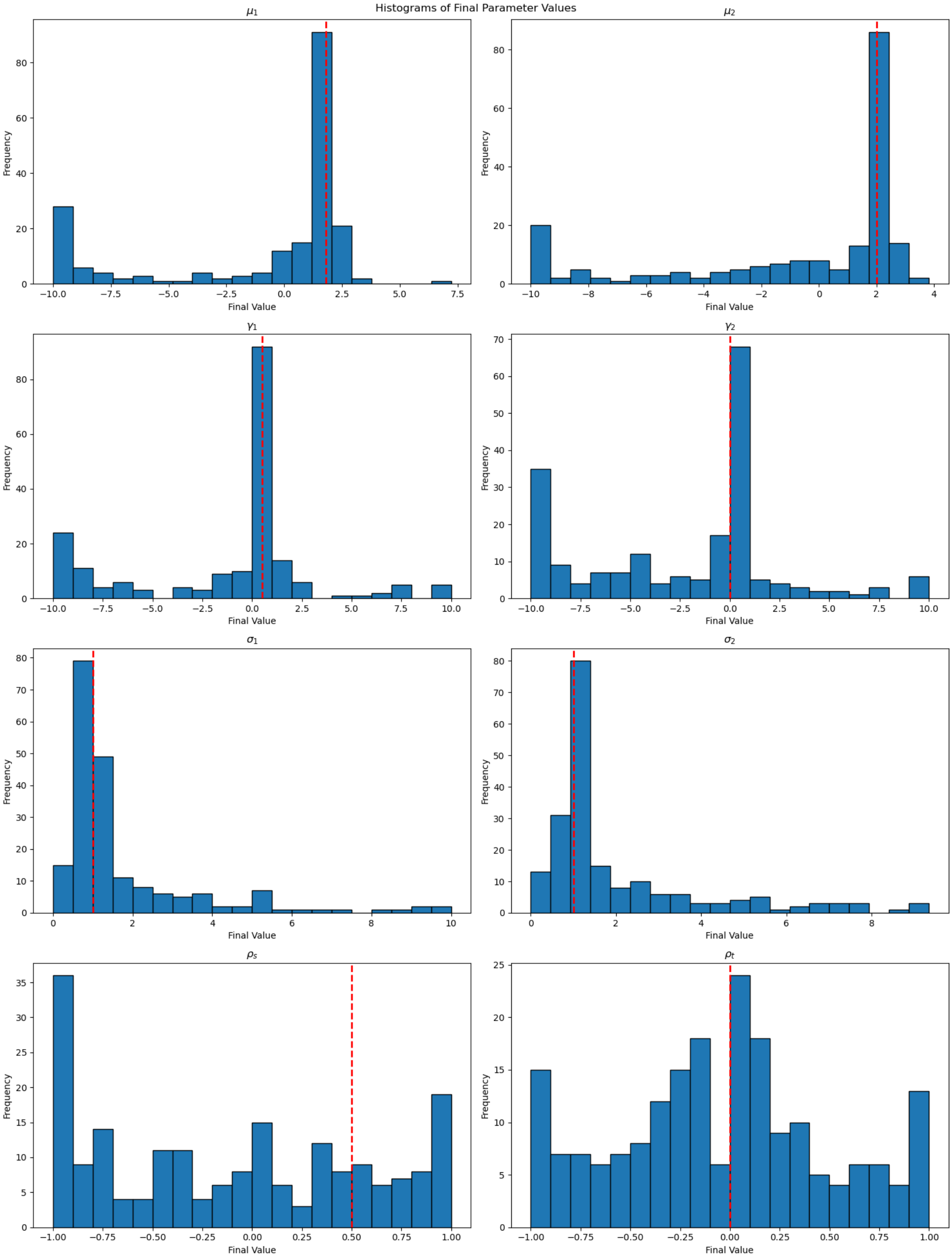

Figure 6. Results of 200 estimations with Wasserstein-1 loss and initial values that are uniform over broad intervals

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. [9]

Arjovsky, Martin, Soumith Chintala, and Léon Bottou. 2017. *Wasserstein GAN.* arXiv: 1701.07875 `[stat.ML]`. [6, 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. [7]

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. [9]

D'Errico, John. 2024. *fminsearchbnd, fminsearchcon.* https://www.mathworks.com/matlabcentral/fileexchange/8277-fminsearchbnd-fminsearchcon. MATLAB Central File Exchange. Accessed September 12, 2024. [11]

Diederik, P Kingma. 2014. "Adam: A method for stochastic optimization." *(No Title).* [3, 4, 11]

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. [9]

Gao, Fuchang, and Lixing Han. 2012. "Implementing the Nelder-Mead simplex algorithm with adaptive parameters." *Computational Optimization and Applications* 51 (1): 259–77. [12]

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. [3]

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. [3, 6]

Gulrajani, Ishaan, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron Courville. 2017. *Improved Training of Wasserstein GANs.* arXiv: 1704.00028 `[cs.LG]`. [7]

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. [9]

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. [9]

Kaji, Tetsuya, Elena Manresa, and Guillaume Pouliot. 2023. "An adversarial approach to structural estimation." *Econometrica* 91 (6): 2041–63. [1, 4, 5, 7–12, 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. [11]

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. [9]

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. [9]

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