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

Welcome to my thesis! It 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. For $k \in \{1, \ldots, K\}$, let

$$Y_k = f_\Theta(X_k, Z_k;), \tag{1}$$

where Y_k is a vector of outcome variables influenced by a vector of noise variables Z_k. The strength and functional form of the relationships between the variables is defined by a function f and its parameters Θ.

A common approach to this problem is maximum likelihood estimation, that is, to find $\hat{\theta}$ such that

$$\hat{\theta} = \arg \max_{\theta \in \Theta} \mathcal{L}_n(\theta; \boldsymbol{y}) . \tag{2}$$

However, for some more sophisticated economic models, it is not easy or even possible to calculate the likelihood function.

This motivates further approaches, such as simulation methods, which attempts to infer θ based on a simulation of the true data. Most notable among these is perhaps the simulated method of moments.

The question naturally arises of how to judge whether the simulated distribution comes sufficiently close to the real distribution. This is one motivation for adversarial estimation. It is also intuitive that one aproach to this involves classification. In machine learning, popular tool for classification are neural networks, which I introduce next.

2.2 Neural networks

Definition

Training

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*. They "play against each other" based on a parameter estimate $\hat{\theta}_t$ which gets updated iteratively. The generator $G(\hat{\theta}) : Z \to O$ creates simulated data based on a guess of the true parameter value $\hat{\theta}$. The discriminator $D_t : O \to [0, 1]$ is a classifier that returns the probability of a given observation being real rather than coming from the generator. If loss is some objective function that measures the distance between the fake and real samples, the adversarial estimator solve the problem

$$\hat{\theta_{adv}} = \arg\min_{\theta \in \Theta} \max_{D \in \mathcal{D}_n} \text{loss}(D(X_i), D(G(\theta))). \tag{3}$$

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 3 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 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 algorithm (Diederik (2014)), which requires setting a range of hyperparameters. 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 Some discriminators

Recall the unique Nash equilibrium from .... If the true densities p_0 and $p_\theta(x)$ are known, we get the *oracle discriminator*.

Definition 1 (Oracle discriminator)**.** The **oracle discriminator** assigns

$$D_\theta(x) \;:=\; \frac{p_0(x)}{p_0(x) + p_\theta(x)} \tag{4}$$

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. Inspired by 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 2 (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

$$D(x) \;=\; \Lambda(\beta_0 + \sum_{k=1}^{K} \beta_k x_k^{mom}) \tag{5}$$

to every $x \in$.

Note that this classifier has to be calculated anew after each update of θ. While this is calculation will usually be fast on modern computers, the same is not necessarily true of the potentially more powerful neural network discriminator.

Definition 3 (Neural network discriminator)**.** Define a classifier neural network $\mathcal{N} : \mathcal{X}^k \to [0, 1]$ by:

$$\mathcal{N}(x) = \sigma_L(W_L\sigma_{L-1}(W_{L-1} \cdots \sigma_1(W_1 x + b_1) \cdots + b_{L-1}) + b_L), \tag{6}$$

where $x \in \mathbb{R}^n$ is the input vector, L is the number of layers, W_i are weight matrices, b_i are bias vectors, σ_i are activation functions. Assume that this network has been trained at least one step on the classification problem at hand. Then the **neural network discriminator** assigns

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

to every $x \in$.

While neural networks are fully defined by their structure, activation functions, weights, and biases, the latter two are the result of their training. So to implement the neural net classifier in practice, its training has to be specified, including training algorithm, hyperparameters, and number of training steps. ...

3.2 Losses

3.2.1 Cross-entropy loss

Following I. J. Goodfellow et al. (2014), Kaji, Manresa, and Pouliot (2023) chose to minimize:

Definition 4. The empirical cross-entropy loss:

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

3.2.2 Wasserstein loss

4 Simulation

4.1 The Roy model

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

First, I reproduce parts of the author's 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

The replication package can be downloaded from the journal website. It contains the author's 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. Thereafter, real and fake observations are generated and the estimation is performed on them. It is implemented as a constrained minimization of a loss function which in turn calculates the discriminators. 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.

4.2 Implementation details

4.2.1 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 author's 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.

4.2.2 Generators

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 higher-dimensional problems, which shows improved convergence in my simulation.

I employ the `mp` module from Python's standard library to parallelize simulation runs on an HPC cluster (cf. A).

4.3 Wasserstein

To demonstrate the effect of adding the Wasserstein loss, I program another simulation using `pytorch` (Ansel et al. (2024)), a popular and highly developed neural network library. The library GeomLoss (Feydy et al. (2019)) allows me to add an approximation of the Wasserstein loss to it. Besides the Wasserstein loss, I also add several other best practices taken from Athey et al. (2021).

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. [6]

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. [6]

D'Errico, John. 2024. *fminsearchbnd, fminsearchcon.* https://www.mathworks.com/matlabcentral/fileexchange/8277-fminsearchbnd-fminsearchcon. MATLAB Central File Exchange. Accessed September 12, 2024. [6]

Diederik, P Kingma. 2014. "Adam: A method for stochastic optimization." *(No Title).* [3, 5]

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. [6]

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. [2]

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. [2, 4]

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. [5]

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

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. [6]

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. [5]

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. [5]

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.

17. September 2024

Marvin Benedikt Riemer