Neural networks as an econometric estimator

Nonparametric regression with heterogeneous marginal effects through universally approximative partial derivatives

R. Bjørn, October 2018

Abstract

This paper argues that neural networks provide a well-behaved estimator for heterogeneity in the marginal effects of conditional expectation functions. The popular machine learning algorithm is a universal approximator, and thus allows fully nonparametric estimation. Although neural networks have already spread to econometrics, their partial derivatives have not yet received much attention. I investigate the properties of these in a simulation framework, and find that neural networks provide consistent estimates for both linear and non-linear underlying functions, although they may require relatively large samples. I also replicate a well-known instrumental variables study, where neural networks obtain similar average effects, but allow a more nuanced analysis of heterogeneity. All in all, my results indicate that neural networks may supplement modern microeconometric research.

1. Introduction: Neural networks might be the new regression

The argument presented in this paper can be concisely stated: Neural networks can be used much as an econometrician would any regression, but does not impose assumptions on the functional form of the relationship in question. This implies that they may be used to analyse how marginal effects depend on covariates in any microeconometric study.

1.1. Neural networks offer universally approximative marginal effects

Neural networks are an immensely popular algorithm within the literature on *machine learning*.[1] This field is focused on developing programs which can improve their performance on some task by learning from data. In the sub-field of supervised learning, this is done by optimally predicting some output y_i based on a set of regressors $\boldsymbol{x}_i$. Despite their original purpose, such algorithms may be useful for econometricians. They often achieve optimal prediction by estimating the *conditional expectation function* (CEF), which is the object of typical interest in econometric studies. Particularly, if such a CEF is itself causal, then any algorithm which estimates it may be used for the estimation of causal effects.

Such algorithms may offer advantages compared to traditional econometric estimators. Particularly, they may offer efficient *nonparametric estimation* of the CEF, which does not fall prey to the curse of dimensionality. The literature has successfully modelled a variety

[1] I will not introduce machine learning in detail. For good introductions, see Goodfellow *et al.* (2016) or Abu-Mostafa *et al.* (2012a) - however, familiarity with the field will not be necessary to follow here.

of complex problems using *data-driven, adaptive* models which impose no assumptions of functional form, but instead shape themselves to the underlying CEF. This implies that econometricians, given enough data, may leave restrictive parametric models behind.[2]

Neural networks offer exactly this as they are *universal approximators* which can approximate any target function $f(\boldsymbol{x}_i)$ as closely as one would like (by making the network more complex). Variations of them have been successfully applied to many academic and commercial settings, typically under the guise of '*deep learning*' (LeCun *et al.*, 2015).[3] Correctly specified, the target function will be the CEF, which has already led to their use for estimation of treatment effects in econometrics (Hartford *et al.*, 2016).

However, neural networks offer possibilities beyond the expectations themselves, as they also offer a natural estimator for the *marginal effects* of continuous regressors, since the partial derivatives are well-defined. Even though this is well-known, these are rarely used for analytical purposes, likely because the literature is focused on either pure prediction or low-dimensional treatment effects. However, the object of interest in applied econometrics is often how an outcome is affected by changes in high-dimensional or continuous variables. In such cases, the partial derivatives of neural networks provide an interesting estimator, so far largely ignored by the econometric literature, which allows for heterogeneous effects.

1.2. Contribution: Properties & illustration of novel estimator

I provide an analysis of the properties of this estimator. I use a series of simulation experiments to show it provides a well-behaved estimator for a wide range of data-generating processes (going from linear to highly complex). I also replicate a study by Angrist & Krueger (1991) on returns to education, showing that networks are able to replicate their average results, and provide a meaningful analysis of heterogeneity. Although further research would be welcome, I conclude that neural networks appear well suited for analysis of marginal effects. I also hope to illustrate how researchers comfortable with regression analysis should be able to add neural networks to their repertoire with relative ease.

The paper unfolds as follows.[4] Section 2 briefly reviews the existing literature on the use of machine learning in econometrics, placing the paper in a camp of econometricians optimistic that machine learning can be leveraged for causal inference. Section 3 provides the theoretical framework for the paper, mostly following the literature on policy evaluation. Section 4 defines neural networks, and discusses how they work in practice. Section 5 provides the simulation results, showing that neural networks appear consistent and efficient enough to allow analysis. Section 6 corroborates this with the replicated study.

[2]Some have argued that such models lead to the 'end of theory' as 'the data deluge makes the scientific method obsolete' (Anderson, 2008). This is misguided, as concepts such as causal identification and statistical inference remain as relevant as ever. However, the new tools do allow for far less rigid models.

[3]Examples include conquering the game of Go (Silver *et al.*, 2016), image recognition (Krizhevsky *et al.*, 2012; Simonyan & Zisserman, 2014; He *et al.*, 2016), speech recognition (Graves *et al.*, 2013), machine translation (Bahdanau *et al.*, 2014) and natural language processing (Collobert *et al.*, 2011).

[4]The paper is a shorter version of my thesis which is available on GitHub (Bjørn, 2018).

2. Literature review: Optimists & pessimists on causal learning

Machine learning is already becoming popular in econometrics, although there is some disagreement as to what it can be used for.[5] Some believe it is good at what it does but can do no more, while others see new tools to shed light on age-old questions of causality.

2.1. Goin' native: Prediction problems in econometrics

One camp of econometricians believe that machine learning may aid economic prediction, but is unlikely to be helpful for causal inference, succinctly summarized by Mullainathan & Spiess (2017): 'Machine learning not only provides new tools, it solves a different problem'. This view is also implicitly expressed by many economists who 'go native', and solely use the techniques for the prediction problems they were originally developed for.

This literature has investigated several purely predictive areas, particularly *policy problems* where policy-makers are only required to anticipate some response (Kleinberg *et al.*, 2015; Chakraborty & Joseph, 2017).[6] Researchers have also looked to *infer new data*, either through dimensionality reduction or by observing previously unavailable data[7].

Although the problems emphasized by this camp are certainly interesting, they do a poor job explaining why the buck must stop there. Mullainathan & Spiess (2017) argue that machine learning is not built for estimation of effects, and 'the danger in using these tools is taking an algorithm built for [prediction], and presuming their [estimates] have the properties we typically associate with estimation'. This is a valid concern, but there is no reason to dismiss such properties out of hand - rather, we should investigate which techniques may be useful. This paper does exactly that for one algorithm.

2.2. Gettin' busy: Adapting machine learning for causal inference

A different camp of econometricians believe that machine learning offers tools which can be adapted for causal inference, even if they are not ready out of the box (Athey, 2017).

One venue of research has looked to leverage algorithms for estimation of *average treatment effects* under unconfoundedness.[8] While such contributions are certainly interesting, existing estimators already perform quite well for average effects (Athey & Imbens, 2017).

A more promising venue is that on *heterogeneous treatment effects*, where existing estimators may have trouble in higher dimensions or larger samples, environments where machine learning thrives. Econometricians have successfully adapted *random forests* for causal inference, allowing for both limited heterogeneity through 'causal trees' (Athey &

[5]For surveys, see Athey (2017) and Mullainathan & Spiess (2017). See also Varian (2014).

[6]Recent work include prediction of loan repayment (Björkegren & Grissen, 2017), bankruptcy (Barboza *et al.*, 2017), worker productivity (Chalfin *et al.*, 2016) and youth violence (Chandler *et al.*, 2011).

[7]Recent examples include natural language processing (Evans & Aceves, 2016) and using satellite images to predict output (Donaldson & Storeygard, 2016) and track poverty (Blumenstock, 2016).

[8]Prominent examples have used regularized regression (Belloni *et al.*, 2014; Chernozhukov *et al.*, 2015; Athey *et al.*, 2016a) and 'double machine learning' (Chernozhukov *et al.*, 2017).

Imbens, 2016) and 'causal forests' (Wager & Athey, 2017), as well as fully local 'generalized random forests' (Athey *et al.*, 2016b). Similarly, Hartford *et al.* (2016) suggests an adapted neural network for two-stage estimation instrumental variables.

This paper places itself in extension of this, arguing that neural networks provide an interesting fully nonparametric estimator, in the sense that no assumptions are placed on the underlying process. I employ two key differences. First, neural networks are applied with no adaptation. Second, I focus on the marginal effects of continuous variables, a venue where random forests may have problems. These imply that neural networks may be more directly applicable for more general purposes than previously acknowledged.

3. Theory: Questions may be causal, but estimation is just statistics

The end goal of most econometric exercises is to answer causal questions, while machine learning is typically concerned with optimal prediction. Despite this, the end goal of either is often estimation of the CEF, which implies that some learning algorithms may be importable to econometrics. Neural networks are particularly interesting, as they are universal approximators with well-defined partial derivatives.

3.1. The irresistible allure of the conditional expectation function

The theoretical framework of this paper is based in the literature on *policy evaluation* within applied microeconometrics, which arguably provide the best causal estimates economics can boast of (Angrist & Pischke, 2010).[9] The literature focus on strong research designs through explicit identification strategies, typically by ensuring conditionally exogenous variation in causal variables. The key point is that causal identification is theoretical, based on assumptions on the data or clever research designs. Once identification is achieved effects can be expressed through finite differences or marginal effects of conditional expectations, and the statistical exercise is to model dependencies through these.[10]

Several approaches from machine learning also estimate the conditional expectation function (CEF). The goal of *supervised learning*, the subfield most easily portable to econometrics, is to predict some outcome y_i based on a set of features $\boldsymbol{x}_i$, typically by minimizing a loss function measuring the difference between true and predicted values. The theoretically optimal solutions to two often used loss functions - *minimum-squared error* for continuous outcomes and *classification error* for binary outcomes - are exactly the CEF.[11] The implication is that some (although certainly not all) machine learning algorithms may be directly applicable to well-identified econometric research. Neural networks are one such, which may provide an interesting estimator.

[9] For surveys of this literature, see Athey & Imbens (2017), Imbens & Wooldridge (2009) and Angrist & Krueger (1999). For a very popular textbook treatment on the approach, see Angrist & Pischke (2008).

[10] This concept of causality relies on the *potential outcomes* framework (Rubin, 1975), reviewed in more detail in the full thesis (Bjørn, 2018), as well as in Imbens & Rubin (2015) and Angrist & Pischke (2008).

[11] Proofs are provided in the thesis (Bjørn, 2018), but may be familiar to many econometricians.

3.2. Improving estimation through universal approximation

Estimation always requires assumptions due to the 'fundamental problem of causal inference' (Holland *et al.*, 1985, 959) where we only observe one outcome for each individual relationship. The literature typically relies on the stable unit treatment value assumption (Rubin, 1980), but I will invoke slightly stronger assumptions:[12]

Definition 3.1. CEFs are *independent given covariates* with *common functional form* if

$$\forall i: \quad \mathrm{E}\left[y_i | \, \boldsymbol{x}_i, \boldsymbol{x}_{-i}, \boldsymbol{y}_{-i}\right] = f_i\left(\boldsymbol{x}_i, \boldsymbol{x}_{-i}, \boldsymbol{y}_{-i}\right) = f\left(\boldsymbol{x}_i\right). \tag{3.1}$$

This may seem theoretically harsh (and may possibly be relaxed), but they often represent the best we can do in practice. Independence is common in microeconometrics, and common form simply states that we can only uncover differences expressed in covariates.

The statistical objective is then to estimate $f(\boldsymbol{x}_i)$ and its derivatives. *Nonparametric regression* focuses on doing so without restrictive assumptions on the functional form, but traditional approaches (such as kernel regression) often suffer greatly under the curse of dimensionality. Neural networks offer a relevant alternative as they do well in high dimensions, and are capable of approximating any function arbitrarily precisely:[13]

Theorem 3.1 (Universal approximator)**.** For any continuous function $f(\boldsymbol{x}_i) \in \mathbb{R}$ on a compact subset $\mathcal{X} \subseteq \mathbb{R}^k$, and any error $\epsilon > 0$, there exists a neural network with one hidden layer (as in definition 4.1, which is explained below)[14] with output $\tilde{f}(\boldsymbol{x}_i)$ such that

$$\forall\, \boldsymbol{x}_i \in \mathcal{X}: \qquad \left|\tilde{f}(\boldsymbol{x}_i) - f(\boldsymbol{x}_i)\right| < \epsilon. \tag{3.2}$$

Proof. Versions of the theorem were separately obtained for various 'squashing' activation functions (see section 4.1) by Hornik *et al.* (1989), Cybenko (1989) and Funahashi (1989). The allowed functions were then extended by Hornik (1991) and Leshno *et al.* (1993). □

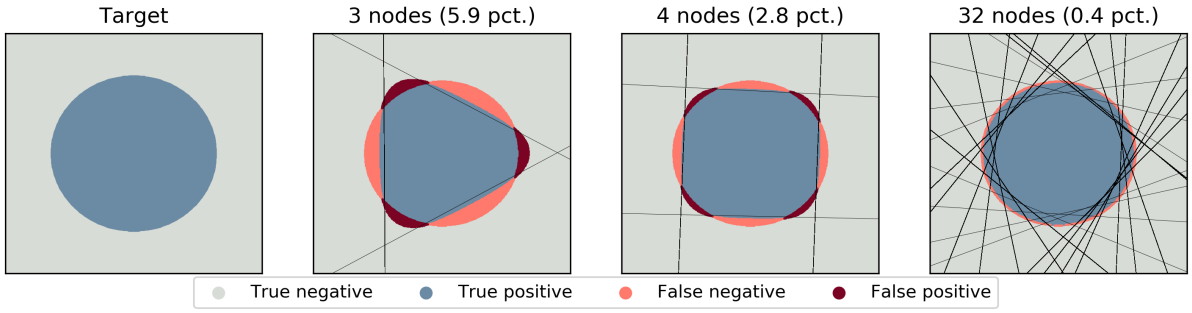
Hornik *et al.* (1990) showed that neural networks are capable of simultaneously providing an arbitrarily close approximation to the function's derivatives. White (1990) showed further that it is in fact possible to learn such optimal networks, by proving consistency for a class of 'connectionist sieve estimators' (Grenander, 1981; White & Wooldridge, 1990). Gallant & White (1992) extended the results by proving consistency for the marginal effects as well. However, modern neural networks are not trained using such sieve estimators. I will therefore rely on a simulation study to illustrate the continued consistency of estimates. First however, I will introduce and define neural networks themselves.

[12] The mathematical notation I employ is hopefully intuitive to follow, but is described in appendix I.

[13] I consider the resulting asymptotic approximation error an acceptable price for efficient estimation, following the view that 'empirical work [is] an effort to describe the essential features of statistical relationships without necessarily trying to pin them down exactly' (Angrist & Pischke, 2008, p.38).

[14] Formally, for linear output $\sigma(s) = s$ and a locally bounded, piecewise continuous and non-polynomial activation $\eta(\cdot)$, there exists $J^{(1)} \in \mathbb{N}$, $\boldsymbol{\beta}^{(1)} \in \mathbb{R}^{(k+1)\times J^{(1)}}$ and $\boldsymbol{\beta}^{(2)} \in \mathbb{R}^{(J^{(1)}+1)\times 1}$ such that the result holds.

Figure 4.1: *Alas, witness the approximation capacity of the mighty neural network!*



Notes: The classification error of each network is specified in parentheses.

4. Model: What are neural networks, and how can they possibly work?

Neural networks may seem daunting at first, but econometricians will find plenty familiar features within them. In short, they estimate an approximative maximum likelihood framework by combining a number of 'regression-like' models in a data-driven manner. A full introduction to neural networks is beyond this paper, but I will briefly review them.[15]

4.1. How do neural networks approximate functions?

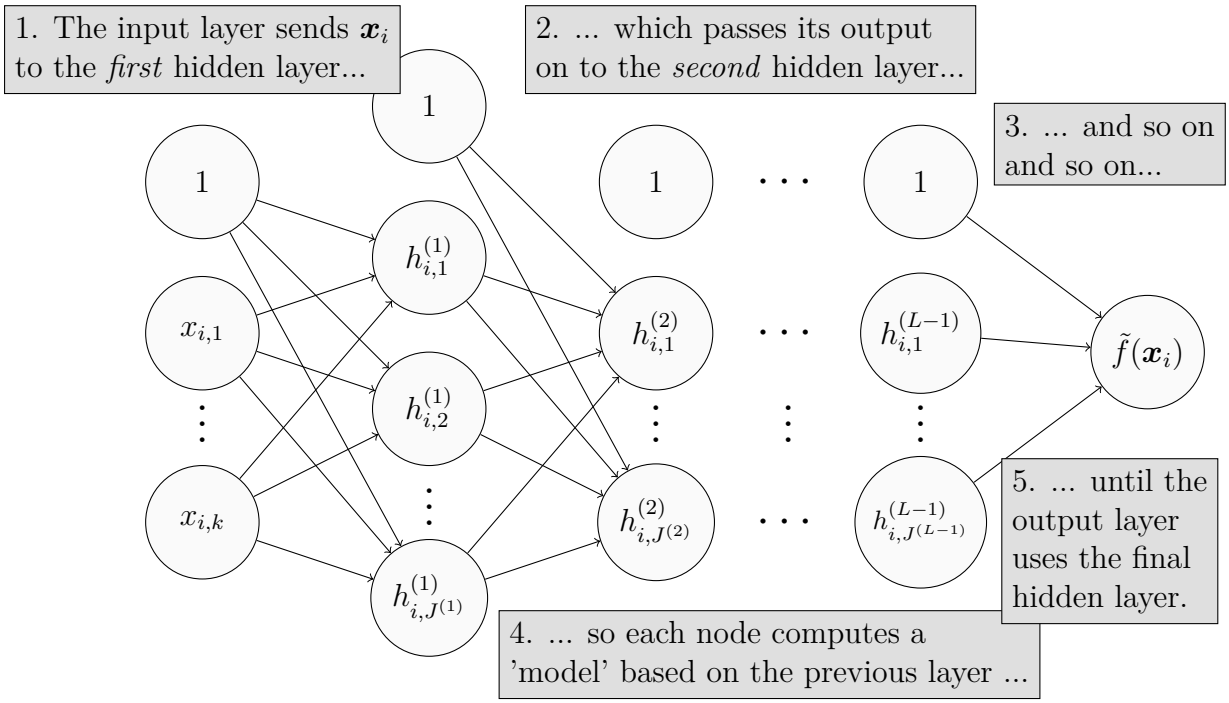
The approximation capacity of a neural network is determined by the number of included models. A small toy example may help shed light on how this is accomplished. Consider classifying the deterministic binary outcome in figure 4.1, where observations inside the circle are positive.[16] A basic linear classifier (such as a logistic regression) would try to separate observations using a single line, and hence cannot recreate the circle. A basic neural network on the other hand might combine several such lines, as shown in the following frames. A network with three 'nodes' (or models) may classify observations within a triangle, which does fairly well even though it cannot replicate the circle perfectly. Adding another node allows a box which does even better. As one adds arbitrarily more nodes, one may get as close to a true circle as one would like.

Although simplified, this illustrates well how neural networks generally work by combining enough simple models to approximate more complex ones. This is typically visualized as in figure 4.2, where computations are divided into various *layers* and *nodes*. The *input* layer is simply the set of regressors and an intercept, and the *output* layer produces the final prediction through a linear signal. The key feature is the existence of *hidden layers* between these. In the toy example above, there was one hidden layer, where each node produced one of the shown lines. Generally, any one hidden node computes a linear signal based on the output of the previous layer and then applies a non-linear *activation*

[15] The full thesis provides a thorough introduction (Bjørn, 2018), as does e.g. Abu-Mostafa *et al.* (2012b) and Goodfellow *et al.* (2016). There are also many good introductions available online.

[16] This example largely follows one in Abu-Mostafa *et al.* (2012b).

Figure 4.2: *A deep neural network with multiple hidden layers.*



function.[17] The hidden layers can be thought of as a form of *feature learning*, where the networks seek to deduce the optimal manner of including the regressors. Generally, one may allow for more complex networks by increasing the number of hidden nodes, or by allowing for multiple layers.[18] Finally, the output layer computes a linear signal based on these transformed variables precisely as a regression would.

4.2. How does one work with neural networks in practice?

It is easy to get lost in the nooks and crannies of neural networks. One particular pitfall is to attach too much meaning to their inner workings. Although networks will automatically include regressors in a clever manner, individual nodes need not tell us anything about the approximated function. In the toy example for instance, lines can only be interpreted as a 'circle' in relation to each other. Similarly, as networks provide a set of estimated coefficients, one could be forgiven for seeking to attach meaning to these. However, this is impossible without a deep understanding of the approximation undertaken by the network.

It is wiser to use neural networks in a nonparametric manner, and focus on their overall predictions. These may seem overwhelming in a single equation (and it may be wise to initially go through it step by step, as the thesis does (Bjørn, 2018)), but one may simply think of them as a regression equation shaped to approximate the underlying function.[19]

[17] The non-linearity is crucial for approximation, as the model could otherwise be reduced to a single linear computation. Typical choices include the *ReLU* $\eta(s) = \max\{0, s\}$ and *sigmoid* functions.

[18] All theoretical results hold for a single layer, but it may be possible to construct a less overall complex multi-layered network which achieves the same approximation error (Abu-Mostafa *et al.*, 2012b).

[19] Algorithms for easy and efficient implementation of all equations are provided in appendix IV.

Definition 4.1. A *feedforward neural network* with $L-1$ hidden layers indexed by $\ell = 0, ..., L$, output function $\sigma(\cdot)$, activation function $\eta(\cdot)$, a $1 \times (k+1)$ regressor and intercept vector $\boldsymbol{x}_i$, a $(J^{(\ell-1)}+1) \times J$ coefficient matrix $\boldsymbol{\beta}^{(\ell)}$ for each hidden layer, and a $(J^{(L-1)}+1) \times 1$ coefficient vector $\boldsymbol{\beta}^{(L)}$ for the output layer estimates

$$\begin{aligned}\tilde{f}(\boldsymbol{x}_i \,|\, \boldsymbol{\beta}) &= \sigma\left(\beta_0^{(L)} + \sum_{j_{L-1}=1}^{J^{(L-1)}} \beta_{j_{L-1}}^{(L)} \eta\left(\dots \eta\left(\beta_{j_2,0}^{(2)} + \sum_{j_1=1}^{J^{(1)}} \beta_{j_2,j_1}^{(2)} \eta\left(\beta_{j_1,0}^{(1)} + \sum_{p=1}^{k} \beta_{j_1,p}^{(1)} x_{i,k}\right)\right)\right)\right) \\ &= \sigma\left(\tilde{\eta}\left(\dots \tilde{\eta}\left(\tilde{\eta}\left(\boldsymbol{x}_i \, \boldsymbol{\beta}^{(1)}\right) \boldsymbol{\beta}^{(2)}\right) \dots\right) \boldsymbol{\beta}^{(L)}\right),\end{aligned}$$

with $\tilde{\eta}(\boldsymbol{s}) = \begin{pmatrix} 1 & \eta(\boldsymbol{s}) \end{pmatrix}$ where $\eta(\cdot)$ is applied elementwise to $s \in \boldsymbol{s}$.

The optimal set of coefficients is found through numerical optimization of an objective function, a concept familiar to econometricians. The typical objectives are inspired by maximum likelihood, and the theoretically optimal solution is then typically CEF.[20] Neural networks owe much of their success to the fact that gradients can be computed quite efficiently despite the large number of coefficients due to the *backpropagation algorithm* (Rumelhart *et al.*, 1986).[21] Algorithms for efficient gradient calculation can be complex, and practitioners typically need not dive into their technicalities.

However, one should certainly be vigilant when training networks, as they are more prone to failure than many econometric estimators. First, there is a higher risk of *optimization failure* since objective functions are typically non-convex (leading to local minima), and gradients may be inexact. Secondly, the representational capacity of neural networks allow them to easily *overfit* the sample data, leading to a strong need for regularization. Finally, the *network architecture* may influence how well the algorithm can approximate the target function, leading either to under- or overfitting (through asymptotic error). Fortunately, there is a large literature on navigating these issues.[22]

4.3. How may neural networks be used to estimate marginal effects?

Neural networks offer a natural estimator of marginal effects since the entire computation is nicely differentiable, allowing for derivation of the partial derivatives with regard to the regressors. This is well-known in the literature (and one uses such partial derivatives when optimizing networks), but the expressions are rarely used for analytical purposes.[23] However, both theory (see section 3.2) and my simulations suggests it is a valid procedure. It also makes intuitive sense - if the overall predictions are decent, then looking at how these change for different covariates should allow analysis of the effect of regressors. The derivation is straightforward, and simply entails successive application of the chain rule.

[20] Common choices include mean-squared error loss with assumed Gaussian errors for regression, and a Bernoulli distribution for binary classification. However, different objectives may be employed.

[21] For a good introduction to backpropagation see Abu-Mostafa *et al.* (2012b).

[22] Goodfellow *et al.* (2016) provides a comprehensive review of the estimation of neural networks.

[23] The only economic application I found was a presentation by Bergtold & Ramsey (2015).

Proposition 4.1. The *marginal effect* of x_p in a neural network (as in definition 4.1) is

$$\begin{aligned}\frac{\partial \tilde{f}(\boldsymbol{x}_i \,|\, \boldsymbol{\beta})}{\partial x_{i,p}} &= \sigma'\left(s_i^{(L)}\right)\left(\sum_{j_{L-1}=1}^{J^{(L-1)}} \beta_{j_{L-1}}^{(L)} \eta'\left(\boldsymbol{s}_i^{(L-1)}\right) \ldots \eta'\left(\boldsymbol{s}_i^{(2)}\right)\left(\sum_{j_1=1}^{J^{(1)}} \beta_{j_2,j_1}^{(2)} \beta_{j_1,p}^{(1)} \eta'\left(s_{i,j_1}^{(1)}\right)\right)\right) \\ &= \sigma'\left(s_i^{(L)}\right) \eta'\left(\boldsymbol{s}_i^{(L-1)}\right) \operatorname{diag}\left(\ldots \operatorname{diag}\left(\eta'\left(\boldsymbol{s}_i^{(1)}\right) \operatorname{diag}\left(\boldsymbol{\beta}_p^{(1)}\right) \boldsymbol{\beta}_{1:}^{(2)}\right) \ldots\right) \boldsymbol{\beta}_{1:}^{(L)},\end{aligned}$$

with $\sigma'(s) = \partial\sigma(s)/\partial s$, $s_i^{(1)} = \boldsymbol{x}_i \boldsymbol{\beta}^{(1)}$, $s_i^{(\ell)} = \begin{pmatrix} 1 & \eta\left(\boldsymbol{s}_i^{(\ell-1)}\right) \end{pmatrix} \boldsymbol{\beta}^{(\ell)}$ for $1 < \ell \leq L$, $\operatorname{diag}\left(\boldsymbol{\beta}_p^{(1)}\right)$ as a $J \times J$ diagonal matrix based on row p of $\boldsymbol{\beta}^{(1)}$, and $\boldsymbol{\beta}_{1:}^{(\ell)}$ is $\boldsymbol{\beta}^{(\ell)}$ excluding $\beta_0^{(\ell)}$.

Proof. The expressions are derived in appendix VI. □

Regressors do not affect outcomes directly in neural networks. Instead, changes first affect the nodes in the initial hidden layer, and this in turn affects the following layer, until the final layer affects the prediction. This illustrates how the network uses the hidden nodes for feature transformation. Even so, the marginal effects remain well-defined.

The expression shows how it is difficult to assign meaning to the estimated coefficients, even though neural networks provide a parametric model. The effect of each coefficient will depend not just on current covariates (as is typical in non-linear models), but also on the specific use of each node decided upon by the network. Therefore, the expression itself may not allow much direct analysis. Instead, one may use the networks in a nonparametric manner, and calculate the marginal effects for each observation. One can then look at how these marginal effects differ across the observed covariate space.

The existence of analytically derived marginal effects is one clear advantage of neural networks over other popular approaches from machine learning. As reviewed in section 2.2, tree-based models (particularly random forests) have been adapted for causal inference with decent success (Athey, 2017). However, decision trees do not provide well-defined marginal effects.[24]. One reason this has not been considered problematic is that literature on program evaluation typical focus on low-dimensional treatment effects, where one simply considers the finite difference in conditional expectations. Given multivalued or continuous regressors, the lack of well-defined marginal effects may be problematic.[25]

Do the partial derivatives of neural networks really capture the true marginal effects for the CEF? It is natural to suspect that as $\tilde{f}(\boldsymbol{x}_i)$ approximates $f(\boldsymbol{x}_i)$, their partial derivatives also approximate each other. As reviewed in section 3.2, the initial theoretical results for neural networks also indicated that this is the case for optimal networks. The key question is whether it is also the case for the neural networks one may estimate in practice. The simulation analysis below suggests that this is indeed the case.

[24]They work by iteratively 'splitting' data at thresholds of the regressors, and finally offering different predictions for each partition of the covariate space. This does not provide a differentiable decision rule.

[25]Practitioners could rely on numerical procedures, such as those underlying partial dependency and individual conditional expectation (ICE) plots (Friedman, 2001; Goldstein *et al.*, 2015). However, this will not alter the underlying theoretical problem: the partial derivatives are not well-defined.

Table 5.1: *Scenarios range from linear to highly complex data-generating processes*

Name	Description
Linear	Basic case for OLS/logistic regression ($\boldsymbol{x}_i\,\boldsymbol{\beta}$).
Polynomial (2)	Adds quadratic terms to the above ($+\,\boldsymbol{x}_i^2\,\boldsymbol{\beta}_2$).
Polynomial (3)	Adds cubic terms to the above ($+\,\boldsymbol{x}_i^3\,\boldsymbol{\beta}_3$).
Wiggly	Global trend, but periodic deviations ('random walks').
Pointy	Global trend, but one large deviation.
Trig. pol (i)	Trigonometric polynomial of order *i*. Periodic.
Ackley	Multimodal, with a sudden drop in the middle.
Rastrigin	Multimodal, with many regularly distributed local minima.
Drop-Wave	Multimodal, with waves emerging from center.

Notes: Formulas and visualizations for the scenarios can be found in appendix V.

5. Simulation results: Consistent, heterogeneous marginal effects

The implemented simulations show that neural networks provide a well-behaved estimator for marginal effects in all investigated scenarios. Estimates appear consistent, although not very efficient (which is reasonable for nonparametric estimation), as well as capable of uncovering both simple and complex underlying heterogeneity in effects.

5.1. Methodology of the simulation study

The simulation analysis is carried out using Monte Carlo simulation, which allows analysis of the expectations of estimators within the simulated cases.[26] Generally, I simulate a $n \times 1$ outcome vector $\boldsymbol{y}$ based on a $n \times k$ regressor matrix $\boldsymbol{x}$. Independent variables are nicely Gaussian with unit variance and non-zero correlation, ensuring they are *weakly unconfounded*, which ensures easy causal interpretation (Imbens, 2000).[27]

Definition 5.1. The *data-generating process* (DGP) consist of

$$\begin{aligned} y_i &= g(\boldsymbol{x}_i, \boldsymbol{\beta}) + u_i, \\ \boldsymbol{x}_i \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Omega}), \quad \boldsymbol{\mu} &\sim \mathcal{N}(0, \boldsymbol{I}_k), \quad \operatorname{diag}(\boldsymbol{\Omega}) = \mathbf{1}, \\ \boldsymbol{\beta} \sim \mathcal{N}(\mathbf{1}, \boldsymbol{I}_k), \quad u_i &\sim \mathcal{N}(0, 1). \end{aligned} \tag{5.1}$$

The CEF is then $\mathrm{E}[y_i|\,\boldsymbol{x}_i] = g(\boldsymbol{x}_i, \boldsymbol{\beta})$ and the marginal effects are its derivatives. The simulation is fully defined by specifying $g(\cdot)$. I investigate a broad class of scenarios to allow for somewhat general claims about properties. They are summarized in table 5.1, while appendix V provides formal definitions, derivations and visualizations for the functions.

[26] Appendix III.2 provides a detailed algorithm for the implemented simulation. For reviews of Monte Carlo simulations, see Cameron & Trivedi (2005), Davidson & MacKinnon (1993) or Hendry (1984).

[27] The thesis explores violations of unconfoundedness, where omitted variable bias, measurement error and irrelevant variables affect networks much in the way an econometrician would expect (Bjørn, 2018).

Table 5.2: *Neural estimators and econometric comparisons*

Estimator	Description
NN (I)	Feedforward neural network with *single* hidden layer.
NN (II)	Feedforward neural network with *two* hidden layers.
MLE	Maximum likelihood based on knowledge of true $g(\cdot)$.
OLS (I)	Ordinary least squares regression.
OLS (II)	Second-order polynomial expansion of $\boldsymbol{x}_i$.

Notes: Implementation of each estimator is described in detail in appendix IV.

The initial three scenarios should be well-known to any econometrician. The next three present straightforward non-linearities, which could conceivably have economic relevance, but economists are unlikely to augment their models for. The final three scenarios were originally designed to test algorithms for numerical optimization (Surjanovic & Bingham, 2017). They provide worst-case scenarios, as they are highly non-linear with many local minima. All in all, the simulated scenarios present a fairly wide range of non-linearity.

The estimators of main interest are the neural networks. I implemented two versions: a basic network with a single hidden layer, and a deep neural network with two hidden layers. Hyperparameters were kept fixed. For comparison, I also implemented two baseline econometric estimators. Maximum likelihood provides a first-best estimator (but is implausible as it relies on knowledge of the true DGP), and OLS provides the optimal linear approximations. Implementations are described in more detail in appendix IV.

Simulation results are only valid for the specified cases, and I can only show a few of these. However, the analysis was carried out using a fairly sophisticated code module (described in appendix III), which allowed varying most aspects of the simulations. I analysed the differences across many pseudo-random seeds and parameters, and I did not find cases where neural networks performed differently than suggested by the results presented here. All code is available on GitHub, so results are fully reproducible.

5.2. Neural estimates are consistent, but not terribly efficient

The simulation study allows one to observe both estimated and true marginal effects in any given case. Denoting true and estimated effects as $\gamma(\boldsymbol{x}_i)$ and $\hat{\gamma}(\boldsymbol{x}_i)$ respectively, the simulations offers a direct estimate of the *mean squared error* (MSE) in expectation.

$$\text{Mean MSE}\left(\hat{\gamma}\right): \quad \frac{1}{Mnk}\sum_{m=1}^{M}\sum_{p=1}^{k}\sum_{i=1}^{n}\left(\hat{\gamma}_p(\boldsymbol{x}_i)-\gamma_p(\boldsymbol{x}_i)\right)^2 . \underset{M\to\infty}{\to} \mathrm{E}\left[\left(\hat{\gamma}-\gamma\right)^2\right] \tag{5.2}$$

This shows the average error made in a given simulation across observations n and regressors k. Averaging across M simulations then provide an estimate of the expectation. Results across the scenarios are shown in table 5.3, with the preferred (plausible) estimator marked in bold. Neural networks generally perform quite well, although they

Table 5.3: *Neural networks are preferred based on MSE!*

	DGP	MLE	OLS (I)	OLS (II)	NN (I)	NN (II)
Linear	0.00	0.00	**0.00**	0.00	0.01	0.01
	(0.000)	(0.000)	(0.000)	(0.000)	(0.001)	(0.002)
Polynomial (2)	0.00	0.00	7.46	**0.00**	0.54	0.31
	(0.000)	(0.000)	(0.025)	(0.000)	(0.068)	(0.048)
Polynomial (3)	0.00	0.00	16.15	6.58	2.44	**1.22**
	(0.000)	(0.000)	(0.086)	(0.057)	(0.293)	(0.187)
Wiggly	0.00	0.00	4.41	3.36	0.48	**0.36**
	(0.000)	(0.000)	(0.014)	(0.011)	(0.038)	(0.029)
Pointy	0.00	29.75	46.80	18.50	1.96	**1.65**
	(0.000)	(26.579)	(0.067)	(0.084)	(0.236)	(0.171)
Trig. pol (3)	0.00	0.00	0.68	0.49	0.05	**0.04**
	(0.000)	(0.000)	(0.002)	(0.003)	(0.006)	(0.003)
Ackley	0.00	7.86	8.00	7.95	2.71	**1.97**
	(0.000)	(0.220)	(0.020)	(0.020)	(0.160)	(0.117)
Rastrigin	0.00	0.00	102.21	102.22	20.37	**12.06**
	(0.000)	(0.000)	(0.150)	(0.150)	(3.118)	(2.224)
Drop-Wave	0.00	...	7.29	7.13	2.80	**0.52**
	(0.000)	(...)	(0.015)	(0.016)	(0.145)	(0.051)

Notes: The table shows MSE between estimated and true marginal effects, averaged across all regressors. Each row represents 100 simulations with 5 regressors and continuous output. Cells show averages across simulations, with standard errors in parentheses. DGPs (rows) and estimators (columns) are described in table 5.1 and 5.2. Estimators were trained on 100,000 observations. Reported values are from a test set of equal size.

come nowhere near the first-best MLE. They outperform linear approximations with more complex non-linearity, and also seem to provide decent estimates in the more linear cases.

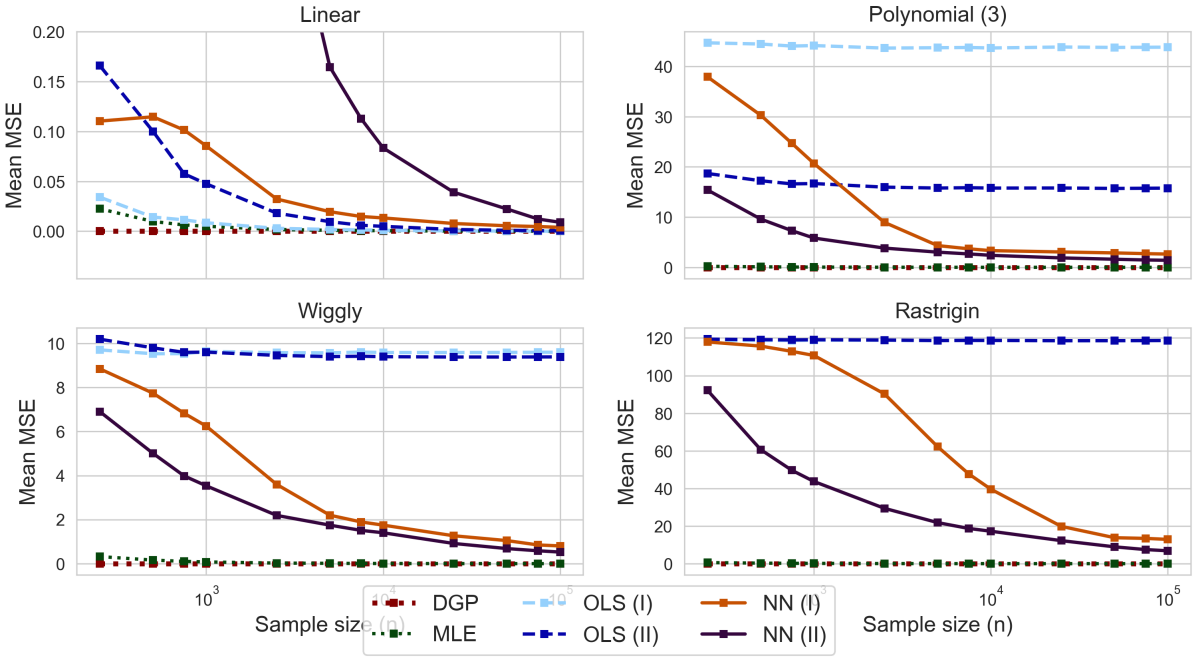
Consider instead *consistency*. Neural networks are unlikely to be consistent for the true effects, since they are approximative models. Recall from the universal approximation theorem (see section 3.2) that a given neural network may achieve an error rate lower than some $\epsilon > 0$ for all covariates. This does mean that some error may remain even in infinity - however, do note that this error can be made arbitrarily small by expanding network architecture. Consistency is better thought of as converging in probability to the optimal possible network. Denote the approximative effects in these as $\tilde{\gamma}(\boldsymbol{x}_i)$.

Theorem 5.1. An estimator $\hat{\gamma}$ is consistent for $\tilde{\gamma}$ if it converges in mean square:

$$\lim_{n\to\infty} \mathrm{E}\left[(\hat{\gamma} - \tilde{\gamma})^2\right] = 0 \qquad \Rightarrow \qquad \lim_{n\to\infty} \mathrm{P}\left(|\hat{\gamma} - \tilde{\gamma}| > \epsilon\right) = 0 \tag{5.3}$$

Proof. The proposition follows directly from Chebyshev's inequality (Rao, 1973). □

Figure 5.1: *MSE converges nicely towards zero, implying consistency*



Notes: The figure shows MSE between estimated and true marginal effects, averaged across all regressors. Each figure represents a simulation study for the given DGP with 3 regressors. In each study, 100 simulations were repeated for each sample size indicated by a marker. Sample sizes can be seen on the log-scaled first axis. Reported values are from a test set of equal size. The second axis show averages across the simulations. DGPs (panels) and estimators (lines) are described in table 5.1 and 5.2.

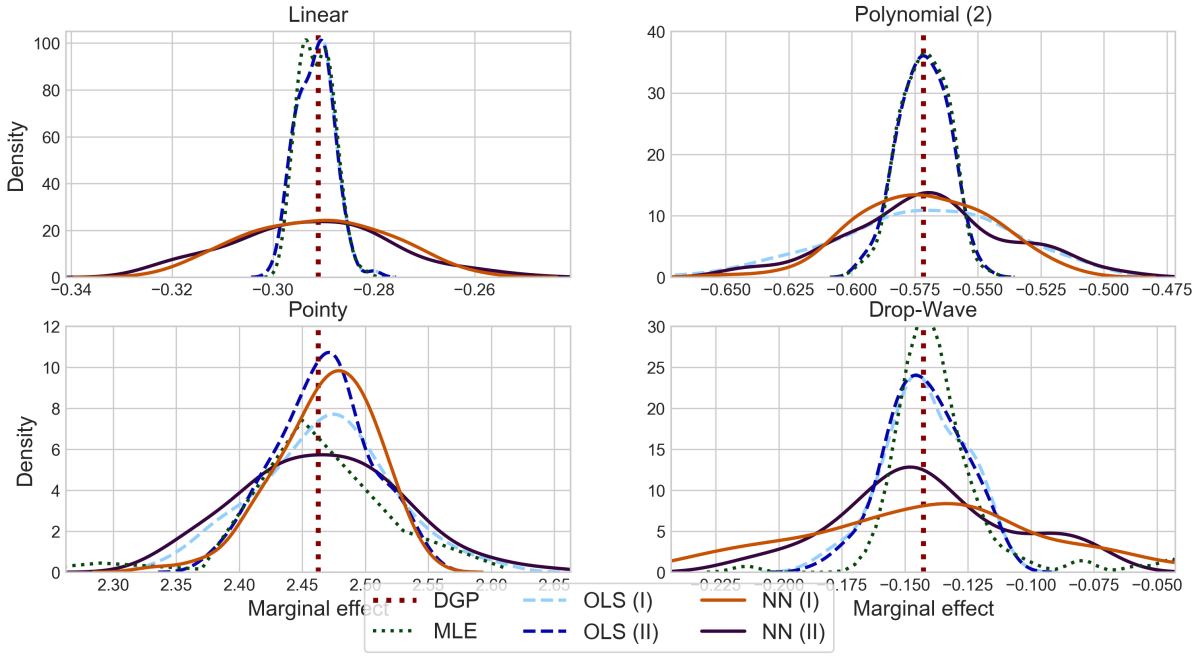
Two problems present themselves. First, optimal networks are unknown. However, since we require both consistency and that networks achieve a low error, it is sufficient to focus on whether errors converge to a very low value in practice. Secondly, simulations do not allow us to let $n \to \infty$. Instead, I will investigate progressively larger samples, and consider whether MSE continues to decrease within observable bounds.

Figure 5.1 illustrates the consistency of networks in such a series of simulations. As the sample increases from 250 to 100,000 observations, the average MSE across simulations converges downwards (non-linearly, as the scale is logarithmic). This is the case in all analysed scenarios (remaining scenarios can be seen in appendix II), indicating that neural networks remain consistent in the face of arbitrary non-linearity.

Networks appear far less efficient than MLE, likely because they exhibit high variance due to their great representational capacity.[28] However, lower efficiency is reasonable for nonparametric estimation, and the results do indicate that networks may reach acceptable error levels within realistic samples. It is also apparent that network architecture may affect efficiency, and should therefore be carefully chosen. The implication is that neural networks should mainly be considered as a big data estimator, but may then work well.

[28]MSE can be decomposed into the variance and bias of an estimator (see e.g. Friedman *et al.* (2009)). If one looks at the estimated bias (i.e. the simple difference between effects), these indicate that bias converges much faster to zero (see figure A.2 in appendix II), leaving variance as the culprit.

Figure 5.2: *Estimates of average marginal effects tend to be less efficient...*



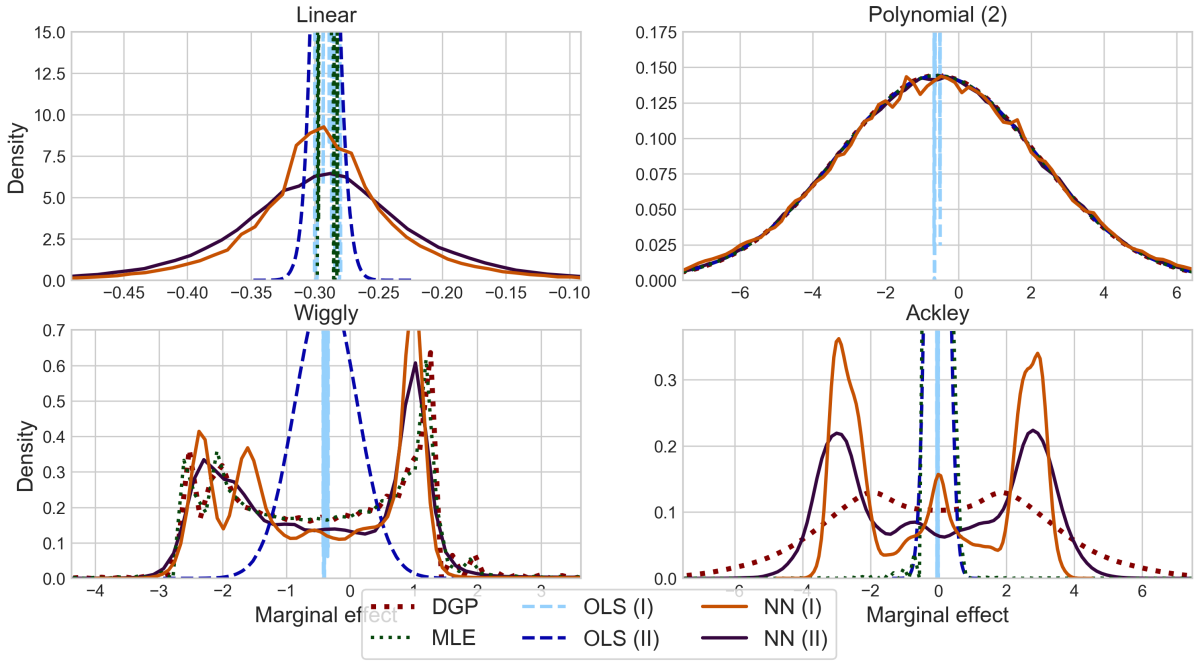
Notes: The figures shows the distribution of estimates of the average marginal effect for a single regressor. Each figure represents 100 simulations for the given DGP with 5 regressors. DGPs (panels) and estimators (lines) are described in table 5.1 and 5.2. Estimators were trained on 100,000 observations. Reported values are from a test set of equal size.

5.3. Neural estimates correctly reflect underlying heterogeneity in effects

The inefficiency has implications for what neural networks should be used for. Particularly, if one is focused mainly on *average effects* (as much of the microeconometric literature is), they may not bring much new to the table. Figure 5.2 shows the distribution of estimates of the average effect of a regressor across the simulations for various scenarios. Notice that although neural networks are nicely centered on the true effects, their distribution tends to be somewhat wide, indicating volatility of estimates. Indeed, if one looks at MSE for average effects instead, neural networks are often beat by one of the linear methods (which tend to provide decent average estimates). Furthermore, there is already a large literature with quite successful estimators of average effects (Athey & Imbens, 2017).

However, such estimators may fail to capture possible heterogeneity in effects, which in turn can cause one to miss much of the overall picture. Figure 5.3 illustrates this. It shows the distribution of different marginal effects across a pooled sample of all the simulations. Notice that the fully linear estimator (light blue) always offers the same estimated effect for everyone. This is sufficient in the first panel, where the true effect is indeed constant. However, the remaining panels show how this misses the overall picture in the case of non-linearity. The final panel shows the worst case possible, where OLS mistakenly concludes that there is no effect, while in reality there are symmetric negative and positive effects which cancel out on average.

Figure 5.3: *... but neural networks allow analysis of heterogeneous effects*



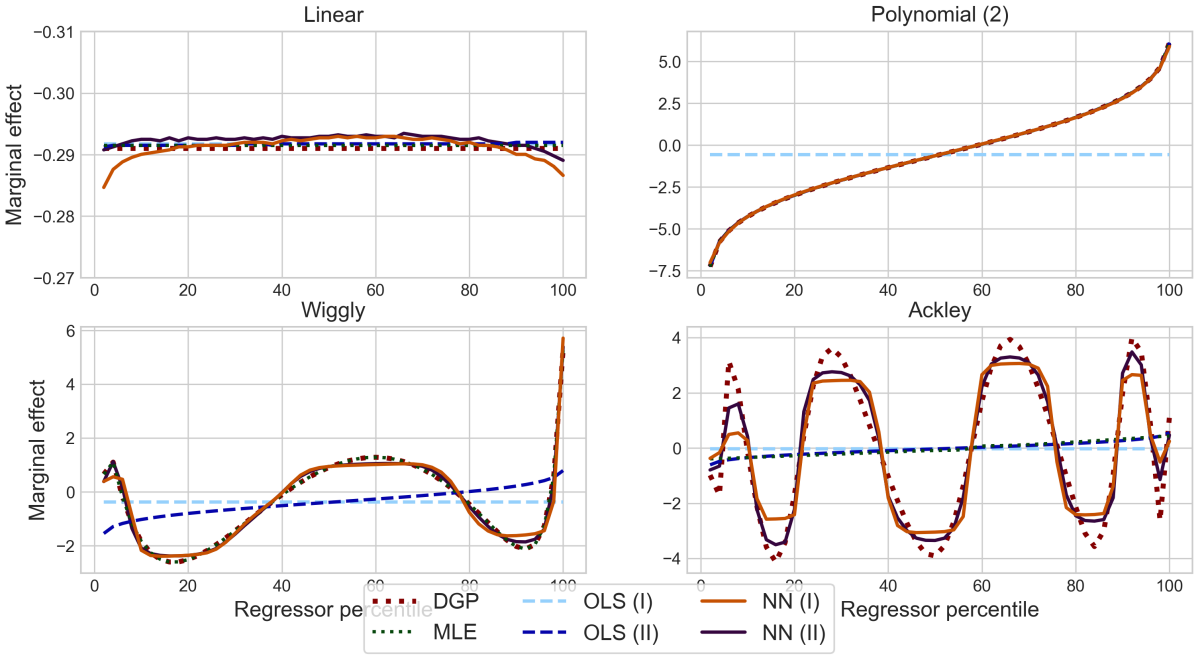
Notes: The figures shows the distribution of estimates of the individual marginal effect of a single regressor. Each figure represents a pooled sample across 100 simulations for the given DGP with 5 regressors. DGPs (panels) and estimators (lines) are described in table 5.1 and 5.2. Estimators were trained on 100,000 observations. Reported values are from a test set of equal size

Neural networks are capable of capturing such effects, which is what allows them to remain consistent for individual effects in the face of non-linearity. Consider again figure 5.3, where neural networks provides decent overall pictures of heterogeneity. In the linear case, they do mistakenly suggest some heterogeneity, but it is likely the bell curve would collapse further given more data. In the case with quadratic terms neural estimates closely follow the actual shape. With the more advanced non-linearity, they follow the overall shape, although with some volatility. One should perhaps not trust small twists and turns in marginal effects (unless the sample is quite large), but neural networks certainly appear to provide a decent picture of the overall heterogeneity.

While looking at the distribution may be initially interesting to gauge whether there is any heterogeneity, researchers will likely be interested in uncovering its shape. As previously noted, it would be difficult to do so by analysing the parametric form of the networks. Instead, one can nonparametrically analyse the estimated effects, and how they depend on various covariates. One could analyse the relationship with any covariate, but an easy example is to look at how effects $\hat{\gamma}_p$ depend on the current level of the same regressor $x_{i,p}$. Figure 5.4 analyses exactly that, by looking at how the marginal effects change as $x_{i,p}$ increases.[29] As always, appendix II provides a similar figure for all scenarios.

[29]I use bins based on percentiles to ensure that each is of equal size. For instance, the bin denoted by 40 shows the average marginal effect among observations with $x_{i,p}$ between the 38th and 40th percentile.

Figure 5.4: *For instance, marginal effects may depend on current* $x_{i,p}$



Notes: The figures show how the marginal effect of a single regressor depends on the current level of the variable. The first axis shows the percentiles of the regressor. The second axis shows the average marginal effect for each percentile. Each figure represents a pooled sample across 100 simulations for the given DGP with 5 regressors. DGPs (panels) and estimators (lines) are described in table 5.1 and 5.2. Estimators were trained on 100,000 observations. Reported values are from a test set of equal size

The neural network appears capable of providing a quite precise picture of the heterogeneity. It is capable of both reflecting the constancy of linear effects, the quadratic changes and the more complex functions.[30] For the latter functions, the expanded linear model miss the overall picture of fluctuating effects because they impose constant global changes. Neural networks have no such restriction. All in all, they appear to provide a well-behaved estimator of individual marginal effects, which may be used both to check for heterogeneity, and for analysing its shape.

6. Case: Returns to schooling may be both increasing and decreasing

I will illustrate how neural networks may enrich analysis by replicating a classical study estimating the returns to schooling (Angrist & Krueger, 1991). The authors employ an instrumental variables (IV) design to eliminate ability bias, leveraging a natural experiment where compulsory schooling requirements in the US affect education. The validity of their design is not crucial for the illustration here, since the key point is that networks may replicate effects and allow heterogeneity.[31] The case also illustrates how neural networks can be extended to modern research designs such as IV (and likely others).

[30]The picture is slightly more volatile in a single simulation. This indicates that neural networks are capable of reflecting the heterogeneity in expectation, but the picture may be less smooth in practice.

[31]The full thesis provides a discussion. See also Angrist & Pischke (2014) and Bound *et al.* (1995).

6.1. Neural networks may easily allow for instrumental variables

Neural networks can be extended for an IV set-up by designing a two-stage estimator.[32] Some care is necessary though, since a naive implementation similar to two-stage least squares (2SLS) would lead to inconsistent estimates (Cameron & Trivedi, 2005, p. 199).[33] Instead, one may utilize a similarly motivated procedure which adds the residuals from the first-stage as controls in the second stage (Newey *et al.*, 1999; Kasy, 2011):[34]

$$\begin{aligned} x_{i,p} &= \mathrm{E}\left[x_{i,p} \middle| \boldsymbol{z}_i\right] + u_{i,x_p}, && \text{(First stage)} \\ y_i &= \mathrm{E}\left[y_i \middle| \boldsymbol{x}_i, \boldsymbol{u}_{i,x}\right]) + u_i. && \text{(Second stage)} \end{aligned} \tag{6.1}$$

The method provides equivalent estimation as 2SLS in the linear case, but is generally valid for non-parametric estimation in non-linear cases. It allows for neural networks in both the first and second stage (or just one of them), each allowing for general non-linearity in said equation. The full thesis provides simulation evidence that this provides consistent estimation, although two-stage neural networks (2SNN) tend to require both quite large samples and strong instruments compared to its linear big brother (Bjørn, 2018).

6.2. Neural estimation & inference in practice

Estimating neural networks in practice requires care, as noted in section 4.2. One must select hyperparameters, particularly the *regularization* and *network architecture*, which affect the network's approximation capacity. Data scientists use a data-driven approach, selecting the ones which perform best in a validation framework (Goodfellow *et al.*, 2016). However, such an 'adaptive' approach may affect inference, as spurious effects in the data affect both conclusions and model structure. A simple solution is to employ 'honest estimation', where separate data partitions are used for selection and estimation respectively, so that model structure can be taken as exogenously given (Athey & Imbens, 2016).

Another pitfall is that networks may fail to optimize properly, so one should always carefully consider the training of the network. One practical point is that regressors should be *standardized* since networks optimize poorly with very large or small inputs.

Finally, inference can be handled for neural networks by the *bootstrap* (Efron, 1979, 1982).[35] Nonparametric bootstrapping will account for both sampling variation and the different local minima the network may end up in. Bootstrapping with a sufficient number of repetitions may be computationally costly but since it should really only be done once to discourage p-hacking (Aschwanden, 2015; Nuzzo, 2014), I consider it feasible.[36]

[32]For traditional IV, see Cameron & Trivedi (2005), Angrist & Pischke (2008), or Wooldridge (2015).

[33]Such an approach would add $\hat{x}_{i,p} = \hat{\mathrm{E}}[x_{i,p}| \boldsymbol{z}_i]$ from a first-stage, which would lead to estimating $g(\hat{x}_{i,p}, \boldsymbol{\beta})$. However, for non-linear $g(\cdot)$ one would really need to estimate $\hat{g}(x_{i,p}, \beta)$.

[34]The approach depends on the assumption of linear separability. If this does not hold one may need to use the CDF of $x_{i,p}$ given $\boldsymbol{z}_i$ (Imbens & Newey, 2009), or no appropriate control may exist (Kasy, 2011).

[35]For an introduction, see Cameron & Trivedi (2005). The thesis provides simulation evidence.

[36]I use relatively few repetitions. For discussions on necessary repetitions, see Cameron & Trivedi

Table 6.1: *Estimates of the average return to schooling*

	OLS (I)	OLS (II)	NN	2SLS	2SNN
Basic	0.071***	0.071***	0.071***	0.089***	0.037***
	(0.0004)	(0.0004)	(0.0034)	(0.0162)	(0.0022)
	[0.070,0.072]	[0.070,0.072]	[0.065,0.078]	[0.063,0.116]	[0.031,0.040]
w/age	0.071***	0.071***	0.067***	0.076**	0.035***
	(0.0004)	(0.0004)	(0.0028)	(0.0286)	(0.0020)
	[0.070,0.072]	[0.070,0.072]	[0.065,0.076]	[0.017,0.121]	[0.032,0.040]
w/dummies	0.063***	0.063***	0.058***	0.081***	0.042***
	(0.0004)	(0.0004)	(0.0030)	(0.0155)	(0.0032)
	[0.063,0.064]	[0.062,0.063]	[0.056,0.067]	[0.055,0.110]	[0.040,0.051]
Full	0.063***	0.062***	0.067***	0.060**	0.044***
	(0.0004)	(0.0004)	(0.0030)	(0.0270)	(0.0037)
	[0.063,0.064]	[0.062,0.063]	[0.057,0.068]	[0.002,0.101]	[0.039,0.053]

Notes: The table shows the average marginal effect of an extra year of education (unstandardized). The dependent value is log of weekly earnings. Covariates for the model specifications are described in the text. Stars show a bootstrapped significance test with *** p<0.01, ** p<0.05, * p<0.1. The bootstrap was based on 99 replications. Parentheses show bootstrapped standard errors. Brackets show a 95 pct. bootstrapped confidence interval (percentile method). Data consists of 329,509 males born in the United States between 1930-1939 and measured in 1980. It is sampled from a 5 pct. sample of that year's US census. Instruments are a full set of quarter of birth times year of birth interactions. Neural networks are estimated on 70 pct. of data unused for hyperparameter selection.

6.3. Neural networks may both replicate and enrich analysis

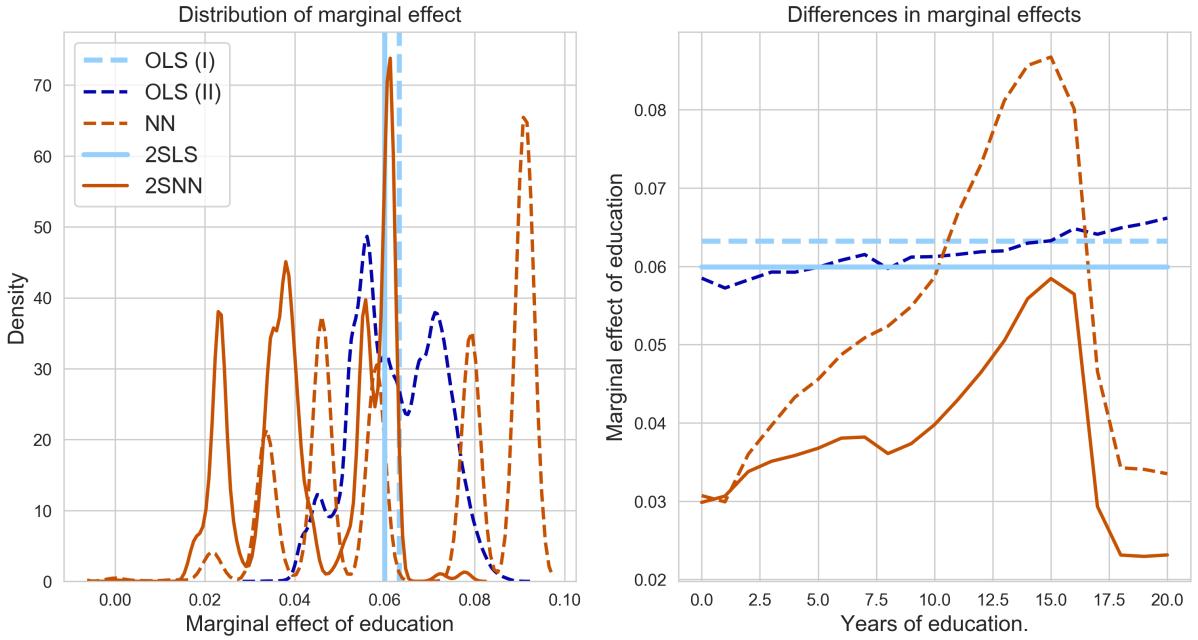
Table 6.1 replicates table V in Angrist & Krueger (1991), which presents the average return to education for various model specifications.[37] The linear methods present the exact same results as the article. The neural network presents almost exactly the same returns in the basic case (6-7 pct. per year), although the estimates appear more volatile. In the IV case, networks provides slightly lower estimates (around 4 pct.), possibly reflecting captured ability bias. Significance is largely similar across the two methods, although it does depend somewhat on the network configuration. Generally, the networks appear capable of replicating the average results and their inference.

However, the linear methods appears to miss significant heterogeneity in the effects. The left panel in figure 6.1 shows the distribution of marginal effects in the data. These are multimodal for networks, suggesting several groups who respond differently to increased schooling. Neural networks then allow us to analyse exactly how effects depend on covariates, which in turn may affect optimal education policy.

(2005), Davidson & MacKinnon (2004), MacKinnon (2002) and Andrews & Buchinsky (2000).

[37]The model and sample are described in more detail in the article. The *basic* set-up includes only year of birth as control dummies. The next model adds age, while the third model instead adds race, region of residence and marriage status. The final specification includes all mentioned covariates.

Figure 6.1: *But look! Heterogeneity!*



Notes: The left figure shows the distribution of estimates of the marginal effect of education. The right figure shows the average marginal effect of education for various levels of education. For every year of education, observations were grouped and the average marginal effect calculated. The estimated models correspond to the 'Full' model specification in table 6.1, which provide further notes.

One might ask is whether there are increasing or decreasing returns to education. The right panel in figure 6.1 shows how the marginal effect depends on current education. It appears the question itself may be too simple - returns to education seems to be increasing at first, and then to sharply decrease for very high education. Basic OLS certainly cannot reflect this, but nor can a simple expansion which imposes global changes on the returns. One could easily analyse the dependency on further covariates, such as age. All in all, neural networks may allow a much deeper understanding of the heterogeneity of effects.

7. Conclusion: Neural networks provide an exciting econometric estimator

Neural networks appear to provide a well-behaved estimator for marginal effects, which allow for full heterogeneity since no functional form is assumed on the underlying relationship. The nonparametric estimator is mainly relevant with larger data sets due to high variance, but bootstrapping may allow researchers to judge robustness for any sample size. The estimator thus provides a valuable addition to any econometrician's toolbox.

7.1. Limitations and venues for further research

One should be aware that further research is necessary to cement these findings. This paper introduces and motivates neural networks used in this context (which has been largely ignored in the econometric literature), but by no means provides the final word

on it. The simulation study employed is naturally limited to the analysed cases. These should be corroborated with further studies using either simulations, actual data or possibly asymptotic theory. However, the broad swathe of scenarios analysed allows some confidence that results are likely to hold for arbitrary non-linearity.

Further research could also compare networks to plausible non- and semiparametric estimators (such as kernel regressions), as well as other developments from machine learning (such as causal forests). However, due to the success of deep learning in solving complex tasks, it is likely that networks will be competitive, particularly in high-dimensional data where traditional approaches has trouble due to the curse of dimensionality.

Finally, one could likely optimize the use of neural networks for econometrics. There is a massive literature on the optimal use of deep learning, but econometricians could explore which approaches are well-suited for the estimation of effects.

7.2. Summary: Fully nonparametric, heterogeneous marginal effects within reach

The results from my simulation study indicates that the partial derivatives of neural network provide a consistent estimator in all the analysed scenarios, in the sense that errors converge to a low enough approximation error to be analytically useful. The representational capacity is bought through fairly low efficiency, but this is to be expected for fully nonparametric approaches and the estimator appears relevant with realistic samples. However, neural networks should mainly be used with larger datasets.

Such networks allow analysis of the heterogeneity of marginal effects without prior specification of its shape. Simulation results showed that neural networks were capable of uncovering both simple and highly complex underlying structures, enriching analysis beyond average effects. This allows both a general analysis of how heterogeneous effects appear to be, and analysis of specific dependencies on covariates.

Such possibilities are highly relevant for econometric research. The replication of Angrist & Krueger (1991) illustrates this perfectly, since returns to schooling appear to differ significantly across the covariate space. As one example, returns appear increasing for most years of schooling, but then to decrease rapidly for advanced education. Such a picture might be difficult to obtain with traditional estimators, but comes easy for neural networks. Any microeconometric study could be similarly enriched.

These results underlie the claim made at the beginning of this paper: Neural networks can be used just as a regression, but allows heterogeneity. This implies both that econometricians could fruitfully learn the technique, and that this will not be particularly difficult. Evaluating networks relies on concepts introduced in undergraduate courses, and the critical issues in estimation are covered in graduate courses on microeconometrics (such as numerical optimization). A few new concepts (such as overfitting) may enrich analysis further, while technicalities (such as efficient gradients) can safely be left to data engineers. There is really no good excuse not to study us some heterogeneity.

References

Abu-Mostafa, Yaser S, Magdon-Ismail, Malik, & Lin, Hsuan-Tien. 2012a. *Learning from data*. Vol. 4. AMLBook New York, NY, USA:.

Abu-Mostafa, Yaser S, Magdon-Ismail, Malik, & Lin, Hsuan-Tien. 2012b. *Learning from data*. Vol. 4. AMLBook New York, NY, USA:. Chap. e-Chapter 7: 'Neural networks'.

Anderson, Chris. 2008. The end of theory: The data deluge makes the scientific method obsolete. *Wired*.

Andrews, Donald WK, & Buchinsky, Moshe. 2000. A three-step method for choosing the number of bootstrap repetitions. *Econometrica*, **68**(1), 23–51.

Angrist, Joshua D, & Krueger, Alan B. 1991. Does compulsory school attendance affect schooling and earnings? *The Quarterly Journal of Economics*, **106**(4), 979–1014.

Angrist, Joshua D, & Krueger, Alan B. 1999. Empirical strategies in labor economics. *Pages 1277–1366 of: Handbook of labor economics*, vol. 3. Elsevier.

Angrist, Joshua D, & Pischke, Jörn-Steffen. 2008. *Mostly harmless econometrics: An empiricist's companion*. Princeton university press.

Angrist, Joshua D, & Pischke, Jörn-Steffen. 2010. The credibility revolution in empirical economics: How better research design is taking the con out of econometrics. *Journal of economic perspectives*, **24**(2), 3–30.

Angrist, Joshua D, & Pischke, Jörn-Steffen. 2014. *Mastering' Metrics: The path from cause to effect*. Princeton University Press.

Aschwanden, Christie. 2015. Science Isn't Broken. It's just a hell of a lot harder than we give it credit for. *FiveThirtyEight*.

Athey, Susan. 2017. The Impact of Machine Learning on Economics. *In: Economics of Artificial Intelligence*. University of Chicago Press.

Athey, Susan, & Imbens, Guido. 2016. Recursive partitioning for heterogeneous causal effects. *Proceedings of the National Academy of Sciences*, **113**(27), 7353–7360.

Athey, Susan, & Imbens, Guido W. 2017. The state of applied econometrics: Causality and policy evaluation. *Journal of Economic Perspectives*, **31**(2), 3–32.

Athey, Susan, Imbens, Guido W, & Wager, Stefan. 2016a. Approximate residual balancing: De-biased inference of average treatment effects in high dimensions. *arXiv preprint arXiv:1604.07125*.

Athey, Susan, Tibshirani, Julie, & Wager, Stefan. 2016b. Generalized Random Forests. *arXiv preprint arXiv:1610.01271*.

Bahdanau, Dzmitry, Cho, Kyunghyun, & Bengio, Yoshua. 2014. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*.

Barboza, Flavio, Kimura, Herbert, & Altman, Edward. 2017. Machine learning models and bankruptcy prediction. *Expert Systems with Applications*, **83**, 405–417.

Belloni, Alexandre, Chernozhukov, Victor, & Hansen, Christian. 2014. High-dimensional

methods and inference on structural and treatment effects. *Journal of Economic Perspectives*, **28**(2), 29–50.

Bergtold, Jason S, & Ramsey, Steven M. 2015. Neural Network Estimators of Binary Choice Process: Estimation, Marginal Effects, and WTP.

Björkegren, Daniel, & Grissen, Darrell. 2017. Behavior revealed in mobile phone usage predicts loan repayment. *arXiv preprint arXiv:1712.05840*.

Bjørn, R. 2018. *Neural networks as an econometric estimator*. Master's thesis, University of Copenhagen. `https://github.com/rbjoern/NeuralEconometrics`.

Blumenstock, Joshua Evan. 2016. Fighting poverty with data. *Science*, **353**(6301), 753–754.

Bound, John, Jaeger, David A, & Baker, Regina M. 1995. Problems with instrumental variables estimation when the correlation between the instruments and the endogenous explanatory variable is weak. *Journal of the American statistical association*, **90**(430), 443–450.

Cameron, A.C., & Trivedi, P.K. 2005. *Microeconometrics: Methods and Applications*. Cambridge University Press.

Chakraborty, Chiranjit, & Joseph, Andreas. 2017. Machine learning at central banks.

Chalfin, Aaron, Danieli, Oren, Hillis, Andrew, Jelveh, Zubin, Luca, Michael, Ludwig, Jens, & Mullainathan, Sendhil. 2016. Productivity and selection of human capital with machine learning. *American Economic Review*, **106**(5), 124–27.

Chandler, Dana, Levitt, Steven D, & List, John A. 2011. Predicting and preventing shootings among at-risk youth. *American Economic Review*, **101**(3), 288–92.

Chernozhukov, Victor, Hansen, Christian, & Spindler, Martin. 2015. Valid post-selection and post-regularization inference: An elementary, general approach.

Chernozhukov, Victor, Chetverikov, Denis, Demirer, Mert, Duflo, Esther, Hansen, Christian, & Newey, Whitney. 2017. Double/Debiased/Neyman machine learning of treatment effects. *American Economic Review*, **107**(5), 261–65.

Clark, Kevin. 2018. Computing Neural Network Gradients. *Lecture notes in Natural Language Processing with Deep Learning*, Stanford University.

Collobert, Ronan, Weston, Jason, Bottou, Léon, Karlen, Michael, Kavukcuoglu, Koray, & Kuksa, Pavel. 2011. Natural language processing (almost) from scratch. *Journal of Machine Learning Research*, **12**(Aug), 2493–2537.

Cybenko, George. 1989. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, **2**(4), 303–314.

Davidson, R., & MacKinnon, J.G. 1993. *Estimation and Inference in Econometrics*. Oxford, Oxford university press.

Davidson, Russell, & MacKinnon, James G. 2004. *Econometric theory and methods*. Vol. 5. Oxford University Press New York.

Donaldson, Dave, & Storeygard, Adam. 2016. The view from above: Applications of

satellite data in economics. *Journal of Economic Perspectives*, **30**(4), 171–98.

Efron, B. 1979. Bootstrap Methods: Another Look at the Jackknife. *The Annals of Statistics,*, **7**(1), 1–26.

Efron, Bradley. 1982. *The jackknife, the bootstrap, and other resampling plans*. Vol. 38. Siam.

Evans, James A, & Aceves, Pedro. 2016. Machine translation: mining text for social theory. *Annual Review of Sociology*, **42**.

Friedman, Jerome, Hastie, Trevor, & Tibshirani, Robert. 2009. *The elements of statistical learning*. Vol. 2. Springer series in statistics New York.

Friedman, Jerome H. 2001. Greedy function approximation: a gradient boosting machine. *Annals of statistics*, 1189–1232.

Funahashi, Ken-Ichi. 1989. On the approximate realization of continuous mappings by neural networks. *Neural networks*, **2**(3), 183–192.

Gallant, A Ronald, & White, Halbert. 1992. On learning the derivatives of an unknown mapping with multilayer feedforward networks. *Neural Networks*, **5**(1), 129–138.

Goldstein, Alex, Kapelner, Adam, Bleich, Justin, & Pitkin, Emil. 2015. Peeking inside the black box: Visualizing statistical learning with plots of individual conditional expectation. *Journal of Computational and Graphical Statistics*, **24**(1), 44–65.

Goodfellow, Ian, Bengio, Yoshua, Courville, Aaron, & Bengio, Yoshua. 2016. *Deep learning*. Vol. 1. MIT press Cambridge.

Graves, Alex, Mohamed, Abdel-rahman, & Hinton, Geoffrey. 2013. Speech recognition with deep recurrent neural networks. *Pages 6645–6649 of: Acoustics, speech and signal processing (icassp), 2013 ieee international conference on*. IEEE.

Grenander, Ulf. 1981. *Abstract inference*. Tech. rept.

Hartford, Jason, Lewis, Greg, Leyton-Brown, Kevin, & Taddy, Matt. 2016. Counterfactual Prediction with Deep Instrumental Variables Networks. *arXiv preprint arXiv:1612.09596*.

He, Kaiming, Zhang, Xiangyu, Ren, Shaoqing, & Sun, Jian. 2016. Deep residual learning for image recognition. *Pages 770–778 of: Proceedings of the IEEE conference on computer vision and pattern recognition*.

Hendry, David F. 1984. Monte Carlo experimentation in econometrics. *Handbook of econometrics*, **2**, 937–976.

Holland, Paul W, Glymour, Clark, & Granger, Clive. 1985. Statistics and causal inference. *ETS Research Report Series*, **1985**(2).

Hornik, Kurt. 1991. Approximation capabilities of multilayer feedforward networks. *Neural networks*, **4**(2), 251–257.

Hornik, Kurt, Stinchcombe, Maxwell, & White, Halbert. 1989. Multilayer feedforward networks are universal approximators. *Neural networks*, **2**(5), 359–366.

Hornik, Kurt, Stinchcombe, Maxwell, & White, Halbert. 1990. Universal approximation of

an unknown mapping and its derivatives using multilayer feedforward networks. *Neural networks*, **3**(5), 551–560.

Imbens, Guido W. 2000. The role of the propensity score in estimating dose-response functions. *Biometrika*, **87**(3), 706–710.

Imbens, Guido W, & Newey, Whitney K. 2009. Identification and estimation of triangular simultaneous equations models without additivity. *Econometrica*, **77**(5), 1481–1512.

Imbens, Guido W, & Rubin, Donald B. 2015. *Causal inference in statistics, social, and biomedical sciences*. Cambridge University Press.

Imbens, Guido W, & Wooldridge, Jeffrey M. 2009. Recent developments in the econometrics of program evaluation. *Journal of economic literature*, **47**(1), 5–86.

Kasy, Maximilian. 2011. Identification in triangular systems using control functions. *Econometric Theory*, **27**(3), 663–671.

Kingma, Diederik P, & Ba, Jimmy. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Kleinberg, Jon, Ludwig, Jens, Mullainathan, Sendhil, & Obermeyer, Ziad. 2015. Prediction policy problems. *American Economic Review*, **105**(5), 491–95.

Krizhevsky, Alex, Sutskever, Ilya, & Hinton, Geoffrey E. 2012. Imagenet classification with deep convolutional neural networks. *Pages 1097–1105 of: Advances in neural information processing systems*.

LeCun, Yann, Bengio, Yoshua, & Hinton, Geoffrey. 2015. Deep learning. *nature*, **521**(7553), 436.

Leshno, Moshe, Lin, Vladimir Ya, Pinkus, Allan, & Schocken, Shimon. 1993. Multilayer feedforward networks with a nonpolynomial activation function can approximate any function. *Neural networks*, **6**(6), 861–867.

MacKinnon, James G. 2002. Bootstrap inference in econometrics. *Canadian Journal of Economics/Revue canadienne d'économique*, **35**(4), 615–645.

Mullainathan, Sendhil, & Spiess, Jann. 2017. Machine learning: an applied econometric approach. *Journal of Economic Perspectives*, **31**(2), 87–106.

Newey, Whitney K, Powell, James L, & Vella, Francis. 1999. Nonparametric estimation of triangular simultaneous equations models. *Econometrica*, **67**(3), 565–603.

Nielsen, Michael A. 2015. *Neural Networks and Deep Learning*. Determination Press. Chap. A visual proof that neural nets can compute any function.

Nuzzo, Regina. 2014. Scientific method: statistical errors. *Nature News*, **506**(7487), 150.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., & Duchesnay, E. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, **12**, 2825–2830.

Rao, Calyampudi Radhakrishna. 1973. *Linear statistical inference and its applications*. Vol. 2. Wiley New York.

Rubin, Donald B. 1975. Bayesian inference for causality: The importance of randomization. *Pages 233–239 of: The Proceedings of the social statistics section of the American Statistical Association*.

Rubin, Donald B. 1980. Discussion of paper by D. Base. *J. Am. Statist. Assoc.*, **75**(3).

Rumelhart, David E, Hinton, Geoffrey E, & Williams, Ronald J. 1986. Learning representations by back-propagating errors. *nature*, **323**(6088), 533.

Scikit-learn. 2018a. *1.17. Neural network models (supervised)*. `http://scikit-learn.org/stable/modules/neural_networks_supervised.html`.

Scikit-learn. 2018b. *sklearn.neural_network.MLPClassifier*. `http://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html`.

Scikit-learn. 2018c. *sklearn.neural_network.MLPRegressor*. `http://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPRegressor.html`.

Seabold, Skipper, & Perktold, Josef. 2010. Statsmodels: Econometric and statistical modeling with python. *In: 9th Python in Science Conference*.

Silver, David, Huang, Aja, Maddison, Chris J, Guez, Arthur, Sifre, Laurent, Van Den Driessche, George, Schrittwieser, Julian, Antonoglou, Ioannis, Panneershelvam, Veda, Lanctot, Marc, *et al.* 2016. Mastering the game of Go with deep neural networks and tree search. *nature*, **529**(7587), 484–489.

Simonyan, Karen, & Zisserman, Andrew. 2014. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.

Statsmodels. 2018a. *statsmodels.base.model.GenericLikelihoodModel*. `http://www.statsmodels.org/dev/dev/generated/statsmodels.base.model.GenericLikelihoodModel.html#statsmodels-base-model-genericlikelihoodmodel`.

Statsmodels. 2018b. *statsmodels.regression.linear_model.OLS*. `https://www.statsmodels.org/dev/generated/statsmodels.regression.linear_model.OLS.html`.

Surjanovic, Sonja, & Bingham, Derek. 2017. *Optimization Test Problems*. `https://www.sfu.ca/~ssurjano/optimization.html`. Simon Fraser University.

Varian, Hal R. 2014. Big data: New tricks for econometrics. *Journal of Economic Perspectives*, **28**(2), 3–28.

Wager, Stefan, & Athey, Susan. 2017. Estimation and inference of heterogeneous treatment effects using random forests. *Journal of the American Statistical Association*.

White, Halbert. 1990. Connectionist nonparametric regression: Multilayer feedforward networks can learn arbitrary mappings. *Neural networks*, **3**(5), 535–549.

White, Halbert, & Wooldridge, J. 1990. Some results on sieve estimation with dependent observations. *Nonparametric and Semiparametric Methods in Economics*, 459–493.

Wooldridge, Jeffrey M. 2015. *Introductory econometrics: A modern approach*. Nelson Education.

Appendices

I. Notation

This appendix provides a brief but hopefully exhaustive list of the mathematical notation used. I generally strive for a nice middle road between mathematical rigour, and ease of representation. The reader may disagree with whether that is achieved.

I.1. Parameters & indices

Generally, I denote total parameters with upper case letters, and index by lower cases (e.g. simulations $m = 1, ..., M$), except where convention states otherwise (e.g. observations $i = 1, ..., n$). Indices may be used differently (particularly i, j) if clear in context.

n	Observations in the sample. Typically indexed $i = 1, ..., n$.
k	Regressors for each observation. Typically indexed $p = 1, ..., k$.
M	Monte Carlo simulations. Typically indexed $m = 1, ..., M$.
L	Hidden layers in the neural network. Typically indexed $\ell = 0, ...,$.
$J^{(\ell)}$	Nodes in hidden layer ℓ. Typically indexed $j = 0, ..., J^{(\ell)}$.

I.2. Scalars, vectors & matrices

Generally, I follow convention and denote scalars with normal font (e.g. $x_{i,p}$) and vectors and matrices with bold (e.g. $\boldsymbol{x}_i$). Minus indicates 'everything but' (as in game theory).

$\boldsymbol{y}$	$n \times 1$ column vector of dependent variables for every observation.
$\boldsymbol{y}_{-i}$	$(n-1) \times 1$ column vector of output for every other observation $j \neq i$.
y_i	Dependent variable for a single observation i ($y_{\in} \boldsymbol{y}$).
$\mathcal{Y}$	Dependent variable space (allowed values of y_i).
$\boldsymbol{x}$	$n \times k$ (or $n \times (k+1)$) feature matrix with row observations and variable columns. It may feature a constant, so that $\boldsymbol{x} = \begin{pmatrix} 1 & \tilde{\boldsymbol{x}} \end{pmatrix}$, hence the variable dimension.
$\boldsymbol{x}_i$	$1 \times k$ (or $1 \times (k+1)$) row vector of regressors for a observation i ($\boldsymbol{x}_i \in \boldsymbol{x}$).
$\boldsymbol{x}_{-i}$	$(n-1) \times k$ (or $(n-1) \times (k+1)$) matrix of features for all observations $j \neq i$.
$x_{i,p}$	Specific independent variable/feature p for a single observation i ($x_{i,p} \in \boldsymbol{x}_i$).
$\mathcal{X}$	Covariate space (allowed values of $\boldsymbol{x}_i$).

$\boldsymbol{u}$, $n \times 1$ row vector of I.I.D. error terms. Substructure follows $\boldsymbol{y}$.

$\boldsymbol{\Omega}$ Symmetric, semi-positive definite covariance matrix.
$\boldsymbol{I}_k$ $k \times k$ identity matrix (e.g. standard normal covariance matrix).
$\boldsymbol{\mu}$ Vector of expected values.

ϵ Typically used as an error measure for an estimator.

$\boldsymbol{\beta}$ Coefficient vector or matrix for a statistical model. Dimensions vary depending on the model, but often a $k \times 1$ column vector. They are typically specified otherwise.
$\boldsymbol{\beta}^{(\ell)}$ $(J^{(\ell-1)}+1) \times J^{\ell}$ matrix of coefficients for all nodes in layer ℓ.
$\boldsymbol{\beta}_j^{(\ell)}$ $(J^{(\ell-1)}+1) \times 1$ row vector of coefficients for node j in layer ℓ
$\boldsymbol{\beta}_p^{(\ell)}$ $1 \times J^{\ell}$ column vector of coefficients for regressor p across all nodes.
$\boldsymbol{\beta}_{j,p}^{(\ell)}$ Coefficient for feature p in node j in layer ℓ.

$\boldsymbol{h}^{(\ell)}$ $n \times (J^{(\ell)}+1)$ output of nodes in hidden layer ℓ.
$\boldsymbol{h}_i^{(\ell)}$ $1 \times (J^{(\ell)}+1)$ output of nodes in hidden layer ℓ for observation i.
$h_{i,j}^{(\ell)}$ Output of node j in layer ℓ for observation i.

$\boldsymbol{s}^{(\ell)}$ $n \times (J^{(\ell)})$ linear signals from $\boldsymbol{h}^{(\ell-1)}$ to $\boldsymbol{h}^{(\ell)}$.
$\boldsymbol{s}_i^{(\ell)}$ $1 \times (J^{(\ell)})$ linear signal from $\boldsymbol{h}^{(\ell-1)}$ to $\boldsymbol{h}^{(\ell)}$ for observation i.
$s_{i,j}^{(\ell)}$ Linear signal from $\boldsymbol{h}^{(\ell-1)}$ to node j in layer ℓ for observation i.

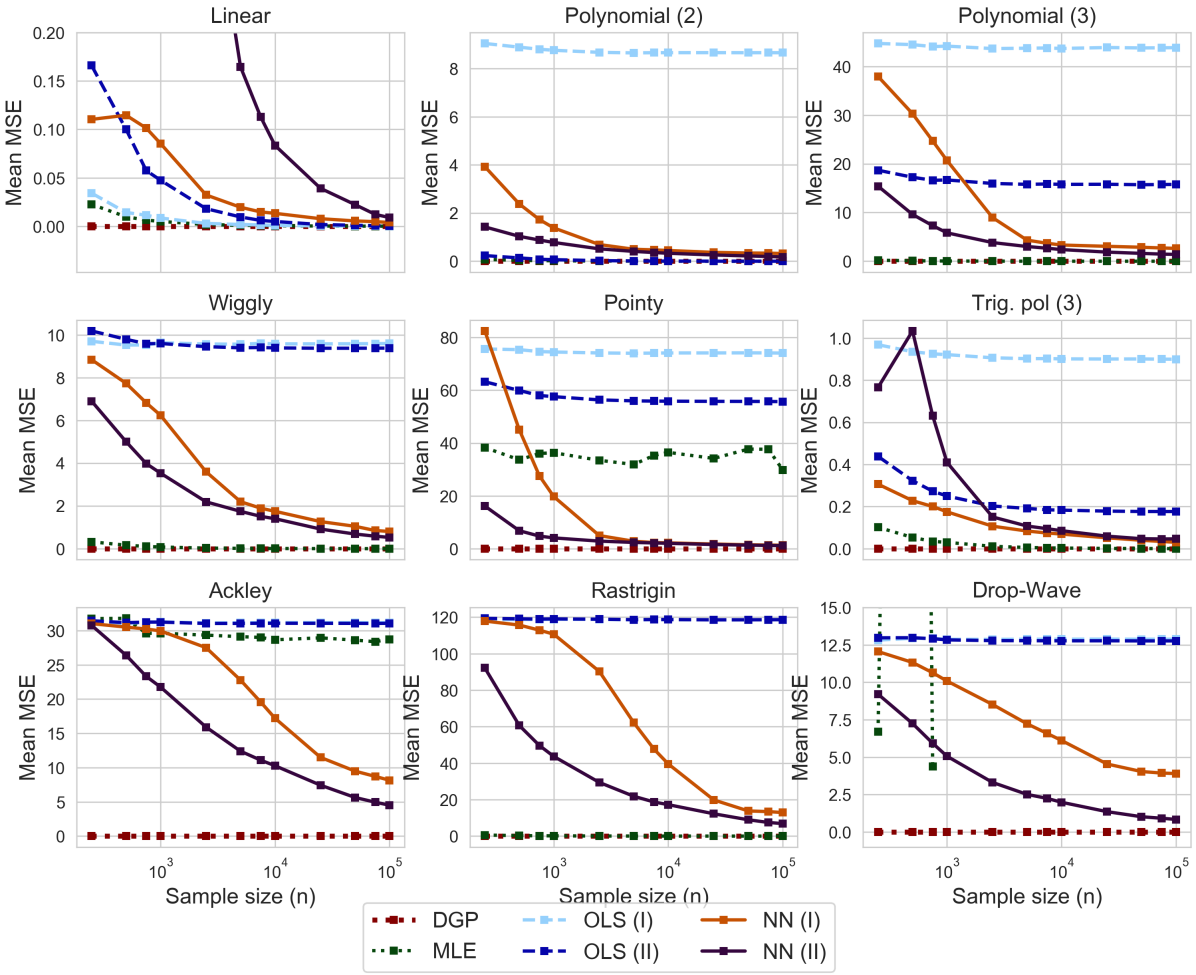
I.3. Functions, derivatives & models

All models can be divided into true models (e.g. $f(\boldsymbol{x}_i)$), approximate models (marked with tilde, $\tilde{f}(\boldsymbol{x}_i)$) and estimated models (marked with hats, e.g. $\hat{f}(\boldsymbol{x}_i)$). They may be either common (e.g. $f(\boldsymbol{x}_i)$) or individual-specific (e.g. $f_i(\boldsymbol{x}_i)$).

$\boldsymbol{f}(\boldsymbol{x})$ Model which outputs a $n \times 1$ row vector of all observations.
$f(\boldsymbol{x}_i)$ Model for a single observation i.
$f_i(\boldsymbol{x}_i)$ Individual-specific model, typically potential outcomes.
$\boldsymbol{\gamma}(\boldsymbol{x})$ $n \times k$ matrix with derivatives of $\boldsymbol{f}(\boldsymbol{x})$ w.r.t. features, evaluated at $\boldsymbol{x}$.
$\boldsymbol{\gamma}(\boldsymbol{x}_i)$ $1 \times k$ row vector with derivatives of $f(\boldsymbol{x}_i)$ w.r.t features, evaluated at $\boldsymbol{x}_i$.
$\gamma_p(\boldsymbol{x}_i)$ Derivative of $f(\boldsymbol{x}_i)$ w.r.t. feature p, evaluated at $x_{i,p}$.
$g(\cdot)$ DGP function.
$L(\cdot|\cdot)$ Loss function resulting from estimates $\hat{y}_i$ or $\hat{\boldsymbol{\beta}}$ given expectation over data distribution.
$\hat{L}(\cdot|\cdot)$ Sample analog of loss function given actual data.
$\sigma(\cdot)$ Output function. Either sigmoid $\sigma(s) = 1/(1+e^{-s})$ for binary $y_i \in \{0,1\}$ or linear $\sigma(s) = s$ for continuous $y_i \in \mathbb{R}$.
$\eta(s)$ Activation function. If unspecified, ReLU $\eta(s) = \max\{0, s\}$.

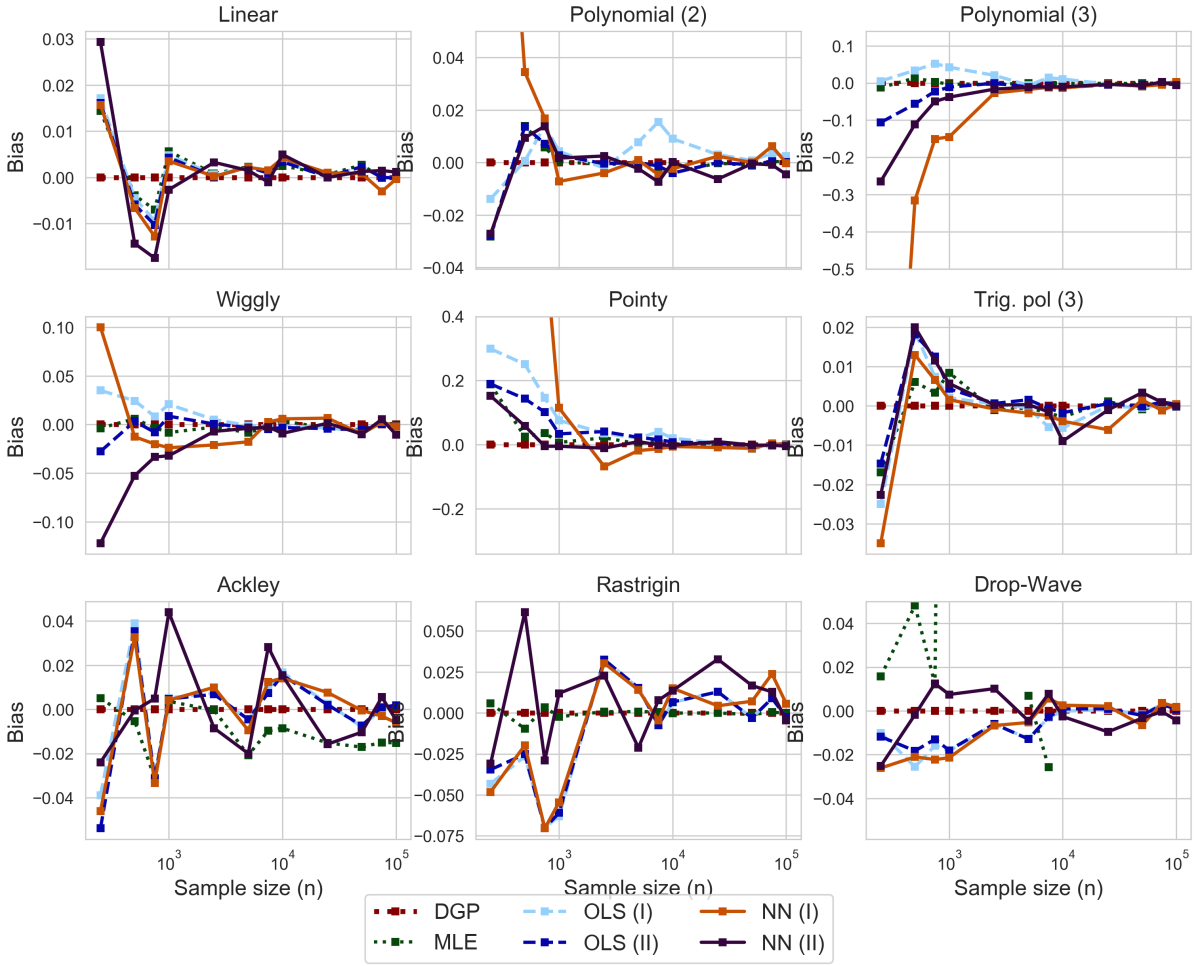
II. Supplementary figures

Figure A.1: *MSE converges nicely towards zero, implying consistency*



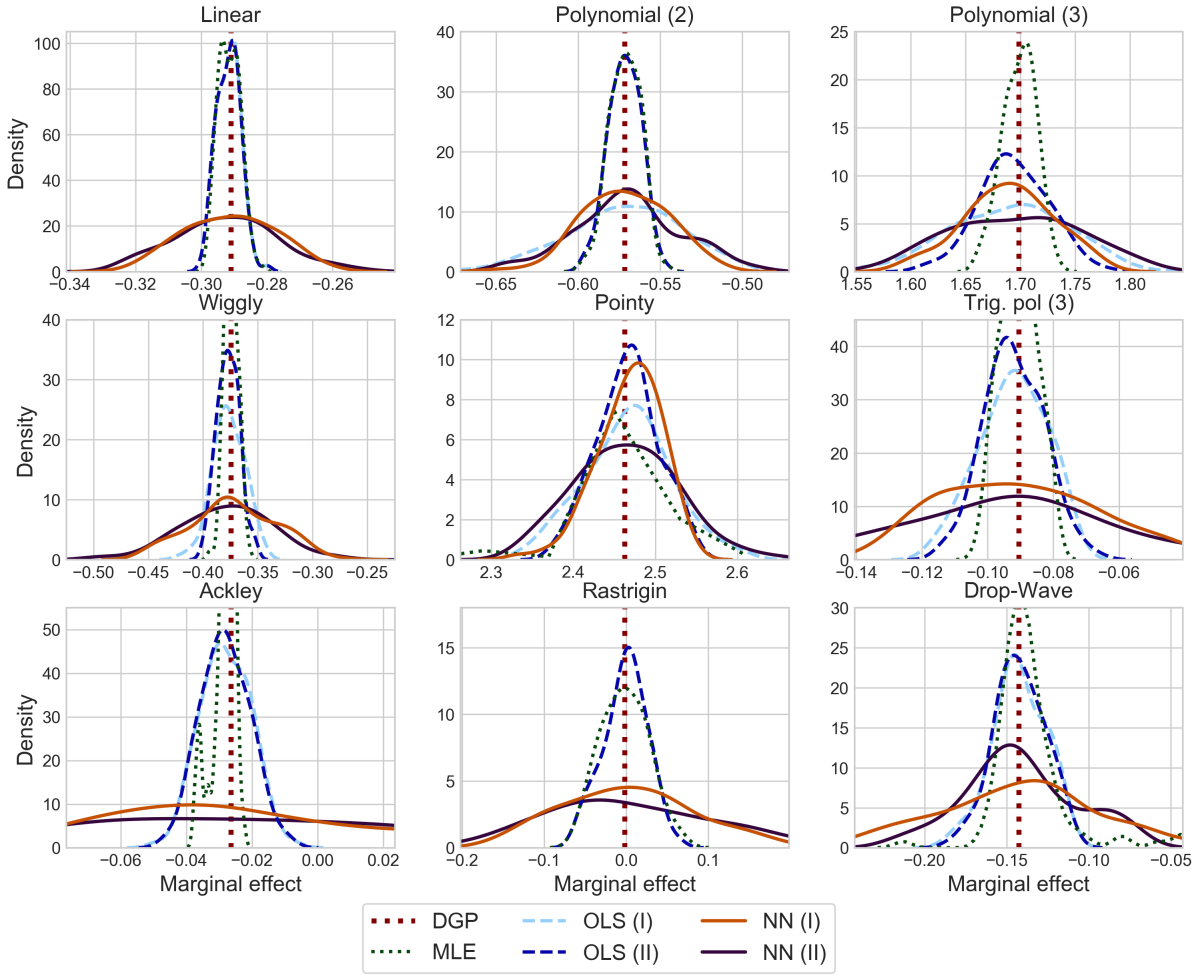
Notes: The figure shows MSE between estimated and true marginal effects, averaged across all regressors. Each figure represents a simulation study for the given DGP with 3 regressors. In each study, 100 simulations were repeated for each sample size indicated by a marker. Sample sizes can be seen on the log-scaled first axis. Reported values are from a test set of equal size. The second axis show averages across the simulations. DGPs (panels) and estimators (lines) are described in table 5.1 and 5.2.

Figure A.2: *Mean error (bias) also goes nicely to zero for a single regressor*



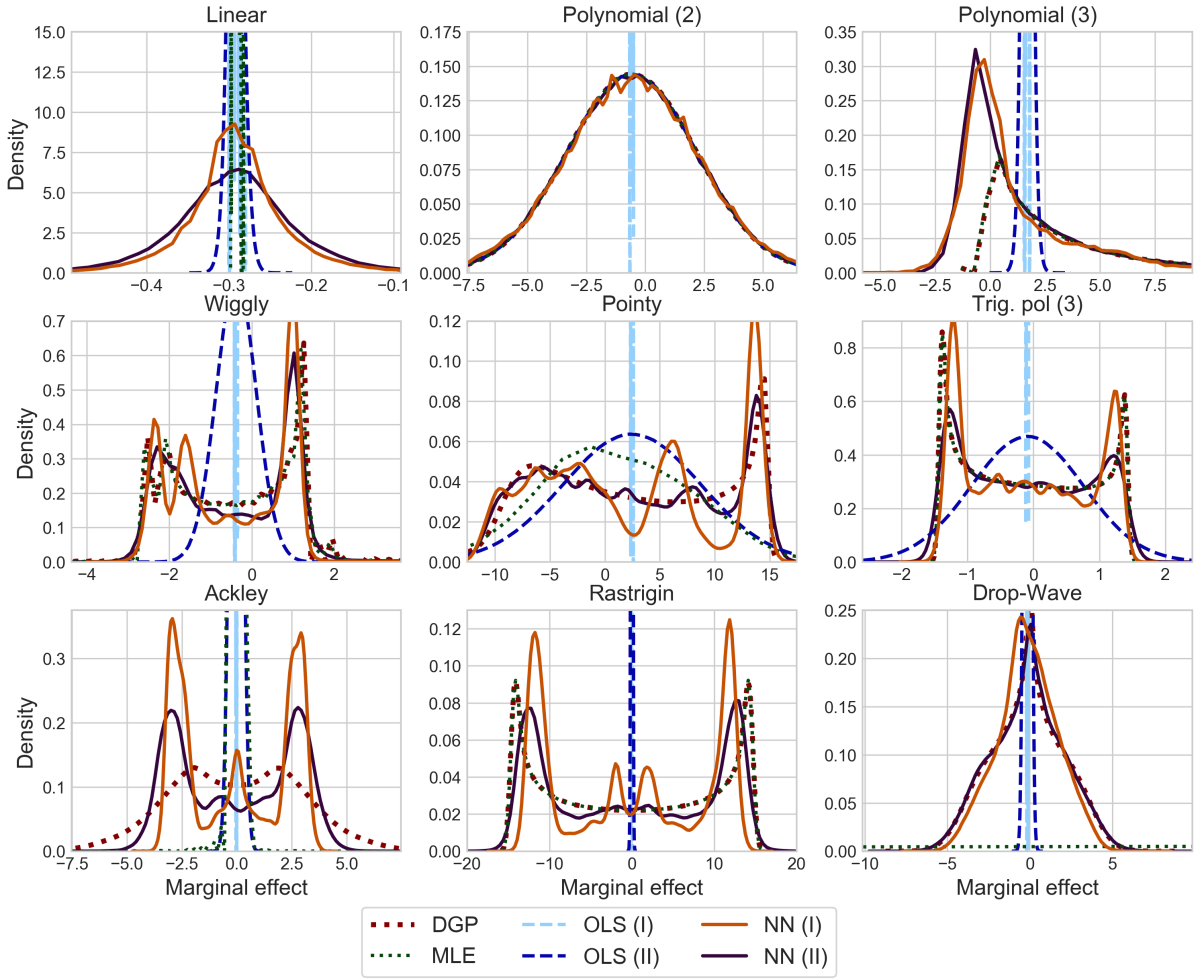
Notes: The figure shows average difference between estimated and true marginal effects (i.e. bias) for a single regressor. Each figure represents a simulation study for the given DGP with 3 regressors. In each study, 100 simulations were repeated for each sample size indicated by a marker. Sample sizes can be seen on the log-scaled first axis. Reported values are from a test set of equal size. The second axis show averages across the simulations. DGPs (panels) and estimators (lines) are described in table 5.1 and 5.2.

Figure A.3: *Estimates of average marginal effects tend to be less efficient...*



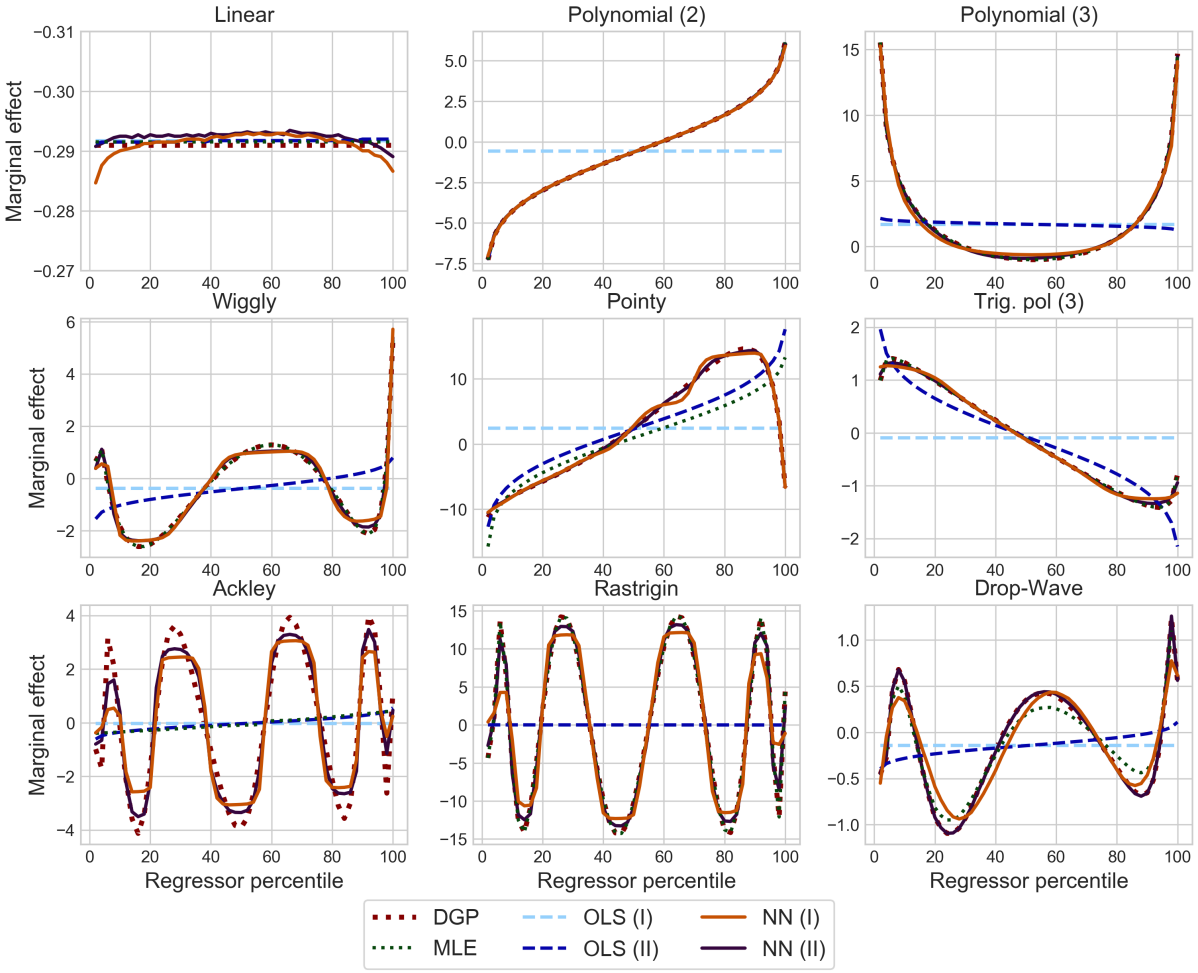
Notes: The figures shows the distribution of estimates of the average marginal effect for a single regressor. Each figure represents 100 simulations for the given DGP with 5 regressors. DGPs (panels) and estimators (lines) are described in table 5.1 and 5.2. Estimators were trained on 100,000 observations. Reported values are from a test set of equal size.

Figure A.4: *... but neural networks allow analysis of heterogeneous effects*



Notes: The figures shows the distribution of estimates of the individual marginal effect of a single regressor. Each figure represents a pooled sample across 100 simulations for the given DGP with 5 regressors. DGPs (panels) and estimators (lines) are described in table 5.1 and 5.2. Estimators were trained on 100,000 observations. Reported values are from a test set of equal size

Figure A.5: *For instance, marginal effects may depend on current* $x_{i,p}$



Notes: The figures show how the marginal effect of a single regressor depends on the current level of the variable. The first axis shows the percentiles of the regressor. The second axis shows the average marginal effect for each percentile. Each figure represents a pooled sample across 100 simulations for the given DGP with 5 regressors. DGPs (panels) and estimators (lines) are described in table 5.1 and 5.2. Estimators were trained on 100,000 observations. Reported values are from a test set of equal size

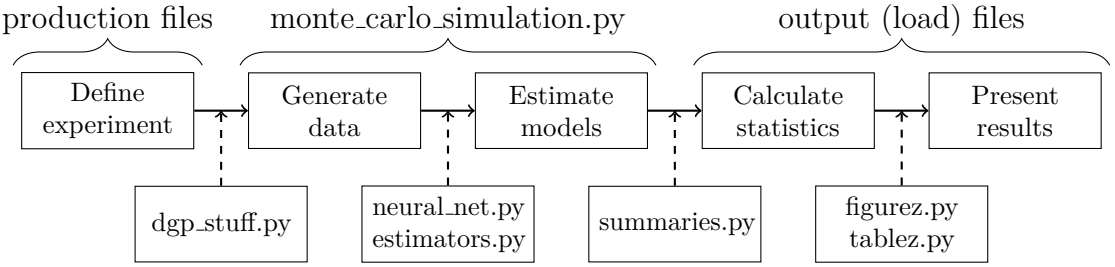
III. Code (available at GitHub)

This appendix describes the code set-up used in the paper. All Python files are available at `https://github.com/rbjoern/NeuralEconometrics`. I designed and implemented a quite general Monte Carlo set-up, which allows for many different scenarios, estimators and summaries. The end result is a somewhat sprawling 4,500 line code structure, which may be slightly hard to read due to the generality of many functions. It is described in exhaustive detail in the thesis appendix (Bjørn, 2018). Here, I provide a brief overview.

III.1. General overview of the code structure

Generally, the code functions like this: A given output file defines the parameters of the desired simulation experiment, and then calls the appropriate simulation function. Afterwards, one calls a number of summary functions to analyse the results.

Figure A.6: *Monte Carlo flow*



A typical flow goes like this: The experiment is defined in a production file, which calls a simulation function to perform the next two steps, and return a set of simulated data and estimated parameters, expectations and marginal effects. The function is provided in the main simulation module, which draws on submodules for defining data-generating processes and estimators. Results are then presented in an output file, which relies on submodules which provide functions for summarizing the simulation results, and presenting it in tables and figures. All in all, my implementation relies on a general module consisting of seven files (available in the folder 'functions'):

monte_carlo_simulation provides a number of simulation algorithms.

dgp_stuff provides tools for generating data. This includes drawing data from stochastic distribution, as well as all $g(\boldsymbol{x}_i, \boldsymbol{\beta})$ functions and derivatives.

neural_net implements the estimators and functions related to neural nets.

estimators implements econometric estimators and functions related to these.

summaries performs computations on the simulation results.

figurez/tablez provides tables and figures for the summaries.

III.2. Description of the main simulation module

The simulation module provides the cornerstone of the code set-up. I designed it to be as flexible as possible, so all stochastic distributions, DGP functions and estimators are provided as input to a general function, and may be varied as one chooses. All simulation relies on a basic algorithm described below. However, the actual implementation is slightly more complex, as described in the thesis.

Algorithm A.1. Monte Carlo simulation.
Input: Parameters, DGP, estimators
Note: Steps marked with * are repeated for a train and test set.
Procedure:

1. Draw population parameters:
   1.1. Draw $\boldsymbol{\beta}$ for $g(\boldsymbol{x}_i, \boldsymbol{\beta})$ as specified.
   1.2. Draw $\boldsymbol{\mu}$, $\boldsymbol{\Omega}$ for $\boldsymbol{x}_i$ as specified.
2. For simulation $m = 1, ..., M$ (either serially or in parallel)
   2.1. Generate data:
      2.1.1. Draw $\boldsymbol{x}$ and $\boldsymbol{u}$ as specified.*
      2.1.2. Compute $\boldsymbol{y}$ as specified based on $g(\boldsymbol{x}_i, \boldsymbol{\beta})$ and $\boldsymbol{u}$.*
   2.2. Estimate models:
      2.2.1. Compute true expectation and marginal effects for the DGP.*
      2.2.2. For estimator in estimators:
         2.2.2.1. Train estimator on $\boldsymbol{x}$.
         2.2.2.2. Compute expectation and marginal effects.

Output: $\boldsymbol{y}$ and $\boldsymbol{x}$, coefficients, expectations and marginal effects for DGP/estimators.

III.3. Output files

The results in the paper were generated using a series of output files, which rely on the described modules. They are available in the folder 'output_files_article'. They are named after the section where they are used. The files marked 'production' runs a simulation analysis, and saves the results to file. The files marked 'figures' then present the result from the corresponding production file.Note that simulations may take a long time to run (12-24 hours). However, the last run results should still be visible in the notebooks.

A few auxiliary files are also provided. The 'replication' file runs the entire code for the analysis of the case. It relies on a dataset which is available from `https://economics.mit.edu/faculty/angrist/data1/data/angkru1991`, and should be placed in the folder 'data'. Finally, there are files for the approximation example and function visualizations.

Any questions regarding the code can be directed to me at rbjoern@outlook.com.

IV. Implementation of estimators

The following appendix provides technical details for the estimation procedures employed in the paper, and references to documentation of the used modules.

IV.1. Neural networks

Estimated neural networks are based on the implementation provided by *Scikit-learn* (Pedregosa *et al.*, 2011), described generally in their handbook (Scikit-learn, 2018a), and more specifically in the technical documentation for both regression (Scikit-learn, 2018c) and classification (Scikit-learn, 2018b). I implemented some algorithms myself to obtain marginal effects. The implementations are found in the code file *neural_net.py*.

The optimal parameters are estimated by minimizing the following loss functions for regression and classification, respectively. They correspond to basic maximum likelihood with added L2-regularization. $f(\boldsymbol{x}_i \,|\, \boldsymbol{x}_i)$ follows the neural network in definition 4.1.

$$\begin{aligned}\hat{L}_C(\hat{\boldsymbol{\beta}}|\, \boldsymbol{y}, \boldsymbol{x}) &= -\frac{1}{n}\sum_{i=1}^{n}\left\{-y_i \ln \hat{f}\left(\boldsymbol{x}_i \,|\hat{\boldsymbol{\beta}}\right) - (1-y_i)\ln\left(1-\hat{f}\left(\boldsymbol{x}_i \,|\hat{\boldsymbol{\beta}}\right)\right)\right\} + \alpha\sum_{\beta\in\boldsymbol{\beta}}\beta^2, \\ \hat{L}_R(\hat{\boldsymbol{\beta}}|\, \boldsymbol{y}, \boldsymbol{x}) &= \frac{1}{2n}\sum_{i=1}^{n}\left\{\left(\hat{f}\left(\boldsymbol{x}_i \,|\hat{\boldsymbol{\beta}}\right) - y_i\right)^2\right\} + \frac{1}{2}\alpha\sum_{\beta\in\boldsymbol{\beta}}\beta^2.\end{aligned} \tag{A.1}$$

Optimization: Adam. Numerical optimization of equation (A.1) is done using the *Adam* algorithm, which provides a good default algorithm for relatively large datasets (+1000 obs) (Kingma & Ba, 2014) I used default parameters from *Scikit-learn* (particularly a training rate of 10^{-4}), except for increasing the allowed epochs to 500.

Regularization: L2. The objective above adds a *weight decay* term to the basic likelihood function through s $L2$-regularization in the form of $\Omega(\boldsymbol{\beta}) = ||\,\boldsymbol{\beta}\,||_2^2$ (Goodfellow *et al.*, 2016, p. 224). The regularization magnitude is set to $\alpha = 10^{-7}$ which is slightly weaker than defaults, reflecting little overfitting in my clean scenario.

No early stopping. A popular type of implicit regularization is *early stopping*, where one stops the optimization when a validation score stops improving (Abu-Mostafa *et al.*, 2012b, p. 24). While this may be attractive for predictive purposes, halting the algorithm before log-likelihood had converged proved detrimental to estimates of marginal effects. I strongly discourage early stopping for analytical purposes.

Activation: ReLU. All hidden nodes employed *ReLU* activation functions $\eta(s) = \max\{0, s\}$, which are the default in the current literature (Goodfellow *et al.*, 2016, 187). The function is not differentiable in zero. My implementation arbitrarily assigns the derivative of zero to one so $\eta'(s) = 1$ if $s \geq 0$ and $\eta'(s) = 0$ if $s < 0$.

Output: Linear/sigmoid. The implementation uses the linear output $\sigma(s) = s$ for regression and sigmoid units in the binary. Both are the typical choices in the literature (Goodfellow *et al.*, 2016, 176).

Network architecture. The network architecture is allowed to depend on the DGP, allowing for more complex networks for more complex functions (see table 5.1 for a list of scenarios). In the single-layered neural network (NN (I) and all IV estimators) I allowed for 30 hidden nodes in the three basic scenarios (linear, quadratic and cubic terms), 80 hidden nodes in the next three (wiggly, pointy and trigonometric polynomial), and a hundred hidden nodes in highly multimodal scenarios (Ackley, Rastrigin and drop-wave). In the estimator with two hidden layers (NN (II)) both layers have the same number of nodes as the previous estimator.

The equations implemented in the paper are analytically elegant, but presents problems in practice since the number of layers may differ, and they hit a dimensional mismatch for more than one observation. Fortunately, one can easily replace them with algorithms.

Algorithm A.2. Forward pass through neural network as in definition 4.1.
Input: Feature matrix $\boldsymbol{x}$, coefficient tensor $\boldsymbol{\beta}$, activation $\eta(\cdot)$, output $\sigma(\cdot)$, layers L.
Procedure:

1. Initialize with
   1.1. $\boldsymbol{s}^{(0)} = \begin{pmatrix} \mathbf{1} & \boldsymbol{x} \end{pmatrix}$,
   1.2. $\boldsymbol{h}^{(0)} = \boldsymbol{s}^{(0)}$.
2. For $\ell = 1, ..., L$:
   2.1. $\boldsymbol{s}^{(\ell)} = h^{(\ell-1)}\, \boldsymbol{\beta}^{(\ell)}$,
   2.2. $\boldsymbol{h}^{(\ell)} = \begin{cases} \begin{pmatrix} \mathbf{1} & \eta\left(\boldsymbol{s}^{(\ell)}\right) \end{pmatrix}, & \ell < L. \\ \sigma\left(s^{(L)}\right), & \ell = L. \end{cases}$

Output: $\boldsymbol{f}(\boldsymbol{x} \,|\, \boldsymbol{\beta}) = \boldsymbol{h}^{(L)}$.

Scikit-learn does not provide marginal effects. To obtain them, I designed and implemented the following simple algorithm (which is equivalent with proposition 4.1).

Algorithm A.3. Computation of marginal effects for a neural network .
Input: Feature matrix $\boldsymbol{x}$, coefficient tensor $\boldsymbol{\beta}$, activation $\eta(\cdot)$, output $\sigma(\cdot)$, layers L.
Procedure:

1. Obtain $\boldsymbol{s}$ through a forward pass as in algorithm A.4.
2. For regressor $p = 1, ..., k$:
   2.1. Initialize with $\boldsymbol{\gamma}_p = \boldsymbol{\beta}_p^{(1)}$.
   2.2. For $\ell = 1, ..., L$:
      2.2.1. $\boldsymbol{\gamma}_p \leftarrow \begin{cases} \eta'\left(\boldsymbol{s}^{(\ell)}\right) \circ \boldsymbol{\gamma}_p\, \boldsymbol{\beta}_{1:}^{(\ell+1)}, & \ell < L. \\ \sigma'\left(\boldsymbol{s}^{(L)}\right) \circ \boldsymbol{\gamma}_p, & \ell = L. \end{cases}$

Output: A matrix of marginal effects $\boldsymbol{\gamma} = \begin{pmatrix} \boldsymbol{\gamma}_1 & ... & \boldsymbol{\gamma}_p \end{pmatrix}$.

Note that $\circ$ indicates elementwise multiplication of matrices, which corresponds to the diagonal products in the proposition. The algorithm does not depend particularly on the Scikit-learn implementation.

IV.2. Implementation of econometric estimators

Maximum likelihood was implemented using the Python module *statsmodels* (Seabold & Perktold, 2010), which provides a generic likelihood framework allowing custom likelihoods (Statsmodels, 2018a). I implemented basic maximum likelihood for regression with Gaussian errors, corresponding to the derivations in e.g. Cameron & Trivedi (2005). I employed BFGS for numerical optimization based on the gradients derived in appendix V. The implementation is found in the code file *estimators.py*.

The linear estimation was also implemented using the Python module *statsmodels* (Seabold & Perktold, 2010), which provides an entirely standard implementation of OLS (Statsmodels, 2018b). I implemented a polynomial expansion of this:

$$\tilde{x_1}\tilde{\boldsymbol{\beta}} = \beta_1^{(1)}x_{i,1} + ... + \beta_k^{(1)}x_{i,k} + \beta_1^{(2)}x_{i,1}^2 + ... + \beta_k^{(2)}x_{i,k}^2 + \beta_{1,2}^{(I)}x_{i,1}x_{i,2} + ... + \beta_{k-1,k}^{(I)}x_{i,k-1}x_{i,k},$$

$$\frac{\partial \tilde{x}_i\tilde{\boldsymbol{\beta}}}{\partial x_{i,p}} = \beta_p^{(1)} + 2\beta_p^{(2)}x_{i,p} + \sum_{q\neq p}\beta_{p,q}^{(I)}x_{i,q}.$$

IV.3. IV estimation

All IV estimators are implemented using a simple two-stage procedure. Various combinations of linear methods and NN for the first and second stage respectively are implemented in *estimators.py* and *neural_net.py*, but they all follow the same recipe. Let $\boldsymbol{x}_i$ denote the endogenous (confounded) regressors here, and $\tilde{\boldsymbol{x}}_i$ the exogenous ones.

Algorithm A.4. Two-stage IV procedure as described in section 6.1.
Input: Endogenous features $\boldsymbol{x}$, exogenous features $\tilde{\boldsymbol{x}}$, instruments $\boldsymbol{z}$.
Procedure:

1. *1st stage*: Filter out endogenous variation.
   1.1. For each endogenous regressors $\boldsymbol{x}_p \in \boldsymbol{x}$:
      1.1.1. Estimate a model for $\mathrm{E}[x_{i,p}|\, \boldsymbol{z}_i, \tilde{\boldsymbol{x}}_i]$.
      1.1.2. Save predictions $\hat{x}_{i,p} = \hat{\mathrm{E}}[x_{i,p}|\, \boldsymbol{z}_i, \tilde{\boldsymbol{x}}_i]$ for all i.
   1.2. Save predictions for every endogenous regressor as $\hat{\boldsymbol{x}}$.
2. *2nd stage*: Estimate final models.
   2.1. Calculate residuals $\boldsymbol{u}_x = \boldsymbol{x} - \hat{\boldsymbol{x}}$.
   2.2. Estimate a model for $\mathrm{E}[y_i|\, \boldsymbol{x}_i, \tilde{\boldsymbol{x}}_i, \boldsymbol{u}_{x,i}]$.
   2.3. Compute expectation and marginal effects for $\boldsymbol{x}_i$ and $\tilde{\boldsymbol{x}}_i$ (but not $\boldsymbol{u}_{x,i}$).

Output: Expectation and marginal effects from the second stage.

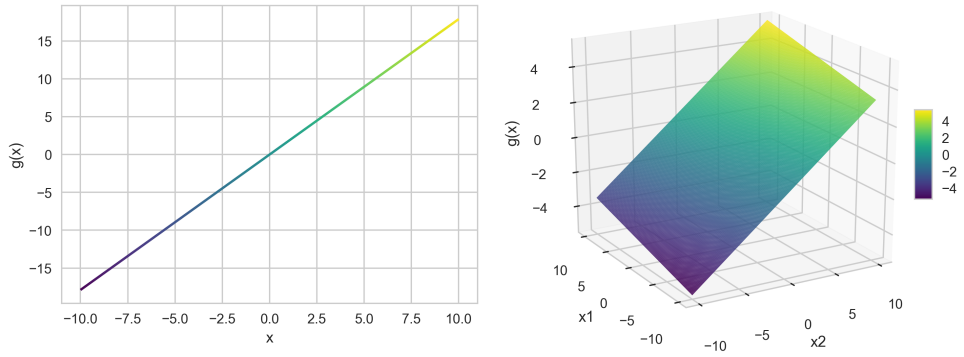
V. Data-generating processes

This appendix formalises and visualizes the scenarios explored in the thesis. For each, a function $g(\boldsymbol{x}_i, \boldsymbol{\beta})$ is defined, and its derivatives computed.

V.1. Linear model (OLS/logistic regression)

$$g(\boldsymbol{x}_i, \boldsymbol{\beta}) = \sum_{p=1}^{k} \beta_p x_{i,p} = \boldsymbol{x}_i \, \boldsymbol{\beta}, \qquad \frac{\partial g(\boldsymbol{x}_i, \boldsymbol{\beta})}{\partial x_{i,p}} = \beta_p, \qquad \frac{\partial g(\boldsymbol{x}_i, \boldsymbol{\beta})}{\partial \beta_p} = x_{i,p}. \tag{A.2}$$

Figure A.7: *Linear function*



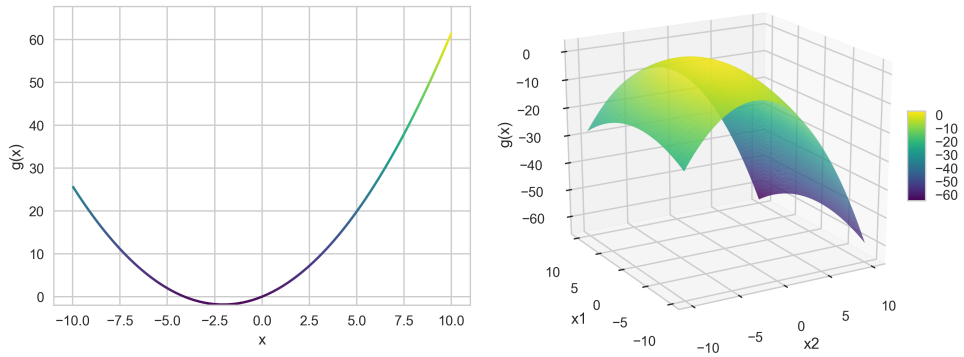
V.2. Polynomial function(s)

A simple approach to allowing non-linearity is to add polynomial terms (typically just quadratic ones). Generally, for a polynomial of order o:

$$g(\boldsymbol{x}_i, \boldsymbol{\beta}) = \sum_{q=1}^{o} \sum_{p=1}^{k} \beta_{p,q} x_{i,p}^{q}, \qquad \frac{\partial g(\boldsymbol{x}_i, \boldsymbol{\beta})}{\partial x_{i,p}} = \sum_{q=1}^{o} q \beta_{p,q} x_{i,p}^{(q-1)}, \qquad \frac{\partial g(\boldsymbol{x}_i, \boldsymbol{\beta})}{\partial \beta_{p,q}} = x_{i,p}^{q}. \tag{A.3}$$

If one imposes no constraints, there will be ok coefficients in $\boldsymbol{\beta}$. Such models remain linear in parameters, so it is quite easy to adapt linear estimators to account for them.

Figure A.8: *Adding quadratic terms*



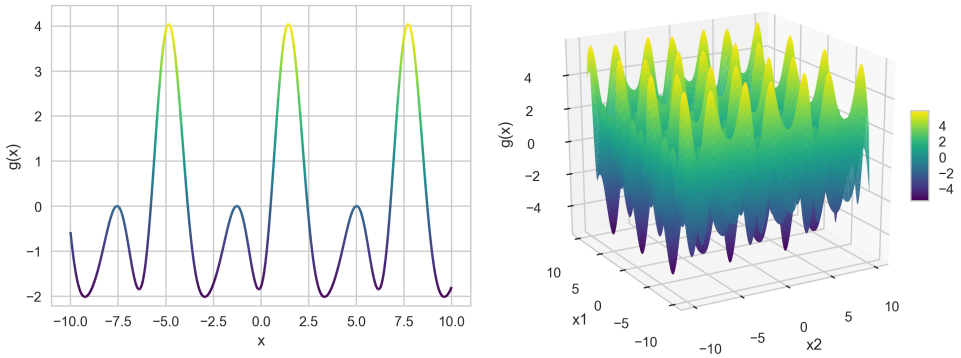
V.3. Trigonometric polynomials

Trigonometric functions are are widely used outside economics, and they are particularly difficult for linear functions because they are periodic rather than monotonic. A composite form is *trigonometric polynomials* (which can be written as a polynomial of $\sin(\boldsymbol{x}_i)$ and $\cos(\boldsymbol{x}_i)$):

$$\begin{aligned} g(\boldsymbol{x}_i, \boldsymbol{\beta}) &= \sum_{q=1}^{o} \sum_{p=1}^{k} \left(\beta_{p,q,1} \sin(q x_{i,p}) + \beta_{p,q,2} \cos(q x_{i,p})\right), \\ \frac{\partial g(\boldsymbol{x}_i, \boldsymbol{\beta})}{\partial x_{i,p}} &= \sum_{q=1}^{o} \left(q\beta_{p,q,1} \cos(q x_{i,p}) - q\beta_{p,q,2} \sin(q x_{i,p})\right), \\ \frac{\partial g(\boldsymbol{x}_i, \boldsymbol{\beta})}{\partial \beta_{p,q,1}} &= \sin(q x_{i,p}), \qquad \frac{\partial g(\boldsymbol{x}_i, \boldsymbol{\beta})}{\partial \beta_{p,q,2}} = \cos(q x_{i,p}). \end{aligned} \tag{A.4}$$

There are $2ok$ coefficients in $\boldsymbol{\beta}$. Although quite non-linear, it remains linear in parameters, and I could use a non-linear transform of the data, and feed that to a linear algorithm.

Figure A.9: *Trigonometric polynomial of third order*



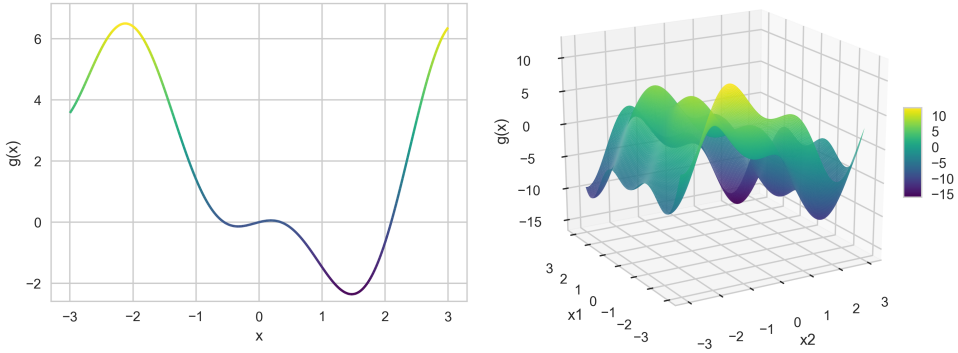
V.4. Wiggle, wiggle, wiggle

The following function aim to recreate a quintessentially non-linear function, which walks about in a somewhat random fashion (or 'wiggles', hence the endearing nickname. It does so by combining global trends (first terms) which periodicity (trigonometric terms). It was inspired by Nielsen (2015).

$$\begin{aligned} g(\boldsymbol{x}_i, \boldsymbol{\beta}) &= \sum_{p=1}^{k} \left(\beta_{p,1} x_{i,p} + a\beta_{p,2} x_{i,p}^2 + \beta_{p,3} x_{i,p} \sin(2x_{i,p}) + \beta_{p,4} x_{i,p} \cos(2x_{i,p})\right), \\ \frac{\partial g(\boldsymbol{x}_i, \boldsymbol{\beta})}{\partial x_{i,p}} &= \beta_{p,1} + 2a\beta_{p,2} x_{i,p} + \beta_{p,3} \sin(2x_{i,p}) + \beta_{p,4} \cos(2x_{i,p}) \\ &\quad + 2\beta_{p,3} x_{i,p} \cos(2x_{i,p}) - 2\beta_{p,4} x_{i,p} \sin(2x_{i,p}), \\ \frac{\partial g(\boldsymbol{x}_i, \boldsymbol{\beta})}{\partial \beta_{p,1}} &= x_{i,p}, \quad \frac{\partial g(\boldsymbol{x}_i, \boldsymbol{\beta})}{\partial a\beta_{p,2}} = x_{i,p}^2, \quad \frac{\partial g(\boldsymbol{x}_i, \boldsymbol{\beta})}{\partial \beta_{p,3}} = x_{i,p} \sin(2x_{i,p}), \quad \frac{\partial g(\boldsymbol{x}_i, \boldsymbol{\beta})}{\partial \beta_{p,3}} = x_{i,p} \cos(2x_{i,p}). \end{aligned} \tag{A.5}$$

There will be $4k$ parameters (since a is not separately identified from $\boldsymbol{\beta}_{p,2}$).

Figure A.10: *How do you fit all that, in them figures?*



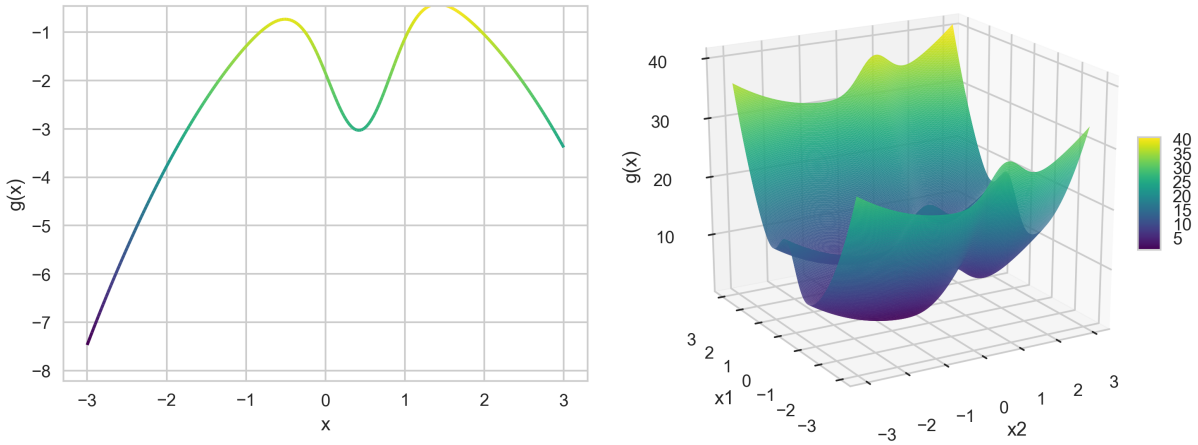
V.5. 'Pointy' function

The following function creates a function with a quadratic trend, and a single pointy break from that trend. It was inspired by Hartford *et al.* (2016).

$$
\begin{aligned}
g(\boldsymbol{x}_i, \boldsymbol{\beta}) &= \sum_{p=1}^{k} \left(\beta_{p,1} x_{i,p} + \beta_{p,2} x_{i,p}^2 + a\beta_{p,3} \exp\left(-b \left(x_{i,p} - \beta_{p,4} \right)^2 \right) \right), \\
\frac{\partial g(\boldsymbol{x}_i, \boldsymbol{\beta})}{\partial x_{i,p}} &= \beta_{p,1} + 2\beta_{p,2} x_{i,p} - 2ab\beta_{p,3} \exp\left(-b \left(x_{i,p} - \beta_{p,4} \right)^2 \right) \left(x_{i,p} - \beta_{p,4} \right), \\
\frac{\partial g(\boldsymbol{x}_i, \boldsymbol{\beta})}{\partial \beta_{p,1}} &= x_{i,p}, \quad \frac{\partial g(\boldsymbol{x}_i, \boldsymbol{\beta})}{\partial \beta_{p,2}} = x_{i,p}^2, \quad \frac{\partial g(\boldsymbol{x}_i, \boldsymbol{\beta})}{\partial a\beta_{p,3}} = \exp\left(-b \left(x_{i,p} - \beta_{p,4} \right)^2 \right), \\
\frac{\partial g(\boldsymbol{x}_i, \boldsymbol{\beta})}{\partial \beta_{p,4}} &= 2ab\beta_{p,3} \exp\left(-b \left(x_{i,p} - \beta_{p,4} \right)^2 \right) \left(x_{i,p} - \beta_{p,4} \right), \\
\frac{\partial g(\boldsymbol{x}_i, \boldsymbol{\beta})}{\partial b} &= -\sum_{p=1}^{k} \left(a\beta_{p,3} \exp\left(-b \left(x_{i,p} - \beta_{p,4} \right)^2 \right) \left(x_{i,p} - \beta_{p,4} \right)^2 \right).
\end{aligned}
\tag{A.6}
$$

There will be $4k + 1$ parameters (since a is not separately identified).

Figure A.11: *Pointy function*



V.6. Ackley function

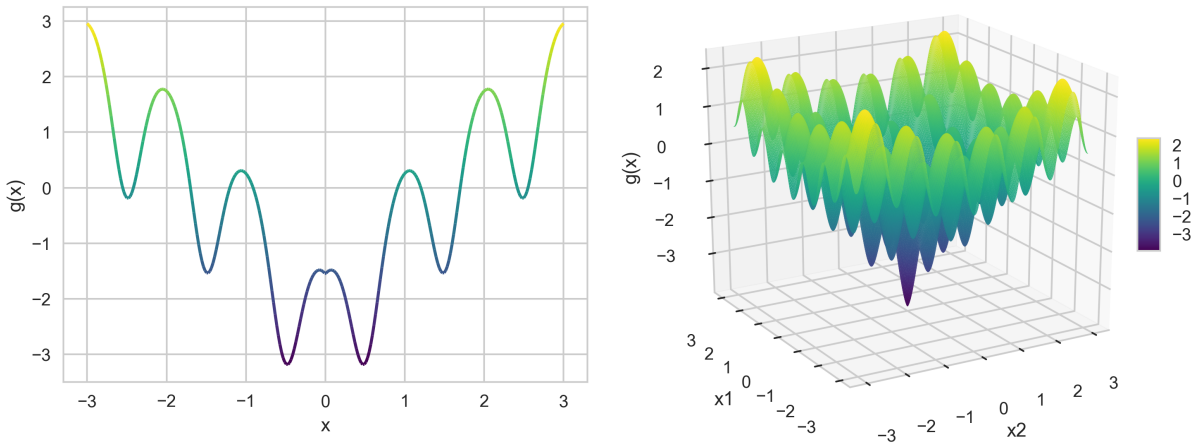
The Ackley function is characterized by a relatively flat outer region, and a sudden drop in the middle (Surjanovic & Bingham, 2017). In the traditional version (a=1, f=1), the minimum is at zero. Reducing d allows us to obtain both positive and negative values.

$$
\begin{aligned}
g(\boldsymbol{x}_i, \boldsymbol{\beta}) &= a\left(-b\exp\left(-c\sqrt{\frac{1}{k}\sum_{p=1}^{k}|\beta_{p,1}|x_{i,p}^2}\right) - d\exp\left(\frac{1}{k}\sum_{p=1}^{k}\beta_{p,2}\cos(ex_{i,p})\right) + f\left(b+\exp(1)\right)\right), \\
\frac{\partial g(\boldsymbol{x}_i, \boldsymbol{\beta})}{\partial x_{i,p}} &= a\left(bc\frac{\exp\left(-c\sqrt{\frac{1}{k}\sum_{p=1}^{k}|\beta_{p,1}|x_{i,p}^2}\right)}{k\sqrt{\frac{1}{k}\sum_{p=1}^{k}|\beta_{p,1}|x_{i,p}^2}}|\beta_{p,1}|x_{i,p} + \frac{de}{k}\exp\left(\frac{1}{k}\sum_{p=1}^{k}\beta_{p,2}\cos(ex_{i,p})\right)\beta_{p,2}\sin(ex_{i,p})\right).
\end{aligned}
\tag{A.7}
$$

There are $2k+6$ parameters. I generally fix $\beta_{p,1}=1$ as a simplification (because elements inside the root must not be negative), and pick the scale parameters to keep values meaningful

$$
\begin{aligned}
\frac{\partial g(\boldsymbol{x}_i, \boldsymbol{\beta})}{\partial \beta_{p,1}} &= abc\frac{\exp\left(-c\sqrt{\frac{1}{k}\sum_{p=1}^{k}|\beta_{p,1}|x_{i,p}^2}\right)}{2k\sqrt{\frac{1}{k}\sum_{p=1}^{k}|\beta_{p,1}|x_{i,p}^2}}\frac{\beta_{p,1}}{|\beta_{p,1}|}x_{i,p}^2, \\
\frac{\partial g(\boldsymbol{x}_i, \boldsymbol{\beta})}{\partial \beta_{p,2}} &= -\frac{ad}{k}\exp\left(\frac{1}{k}\sum_{p=1}^{k}\beta_{p,2}\cos(ex_{i,p})\right)\cos(ex_{i,p}), \\
\frac{\partial g(\boldsymbol{x}_i, \boldsymbol{\beta})}{\partial a} &= \frac{1}{a}g(\boldsymbol{x}_i, \boldsymbol{\beta}), \quad \frac{\partial g(\boldsymbol{x}_i, \boldsymbol{\beta})}{\partial b} = -a\exp\left(-c\sqrt{\frac{1}{k}\sum_{p=1}^{k}|\beta_{p,1}|x_{i,p}^2}\right) + af, \\
\frac{\partial g(\boldsymbol{x}_i, \boldsymbol{\beta})}{\partial c} &= ab\exp\left(-c\sqrt{\frac{1}{k}\sum_{p=1}^{k}|\beta_{p,1}|x_{i,p}^2}\right)\sqrt{\frac{1}{k}\sum_{p=1}^{k}|\beta_{p,1}|x_{i,p}^2}, \quad \frac{\partial g(\boldsymbol{x}_i, \boldsymbol{\beta})}{\partial d} = -a\exp\left(\frac{1}{k}\sum_{p=1}^{k}\beta_{p,2}\cos(ex_{i,p})\right) \\
\frac{\partial g(\boldsymbol{x}_i, \boldsymbol{\beta})}{\partial e} &= ad\exp\left(\frac{1}{k}\sum_{p=1}^{k}\beta_{p,2}\cos(ex_{i,p})\right)\left(\frac{1}{k}\sum_{i=1}^{k}\beta_{p,2}\sin(ex_{i,p})x_{i,p}\right), \quad \frac{\partial g(\boldsymbol{x}_i, \boldsymbol{\beta})}{\partial f} = b+\exp(1).
\end{aligned}
\tag{A.8}
$$

Figure A.12: *Ackley function*



V.7. Rastrigin function

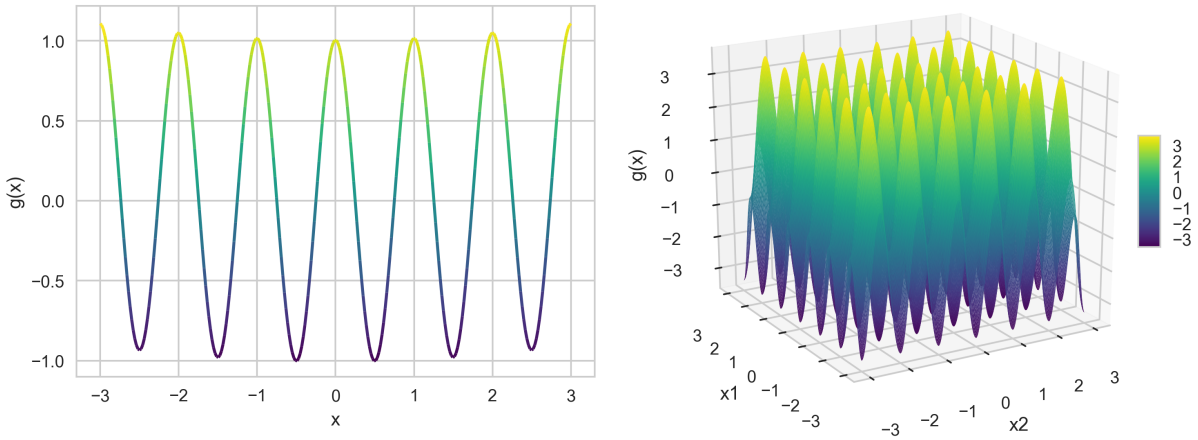
The Rastrigin function is highly multimodal, with many regularly distributed local minima, which in the binary cases yields a spotted pattern of successes and failures (Surjanovic & Bingham, 2017). In its regular form the minimum is at zero, but removing an added constant yields both positive and negative values.

$$\begin{aligned} g(\boldsymbol{x}_i, \boldsymbol{\beta}) &= a\left(\sum_{p=1}^{k}\left(b\beta_{p,1}x_{i,p}^2 - c\beta_{p,2}\cos(2\pi x_{i,p})\right)\right), \\ \frac{\partial g(\boldsymbol{x}_i, \boldsymbol{\beta})}{\partial x_{i,p}} &= a\left(2b\beta_{p,1}x_{i,p} + 2\pi c\beta_{p,2}\sin(2\pi x_{i,p})\right). \end{aligned} \tag{A.9}$$

There are $2k$ parameters (since a, b and c are not separately identified).

$$\frac{\partial g(\boldsymbol{x}_i, \boldsymbol{\beta})}{\partial \beta_{p,1}} = abx_{i,p}^2, \quad \frac{\partial g(\boldsymbol{x}_i, \boldsymbol{\beta})}{\partial \beta_{p,2}} = -ac\cos(2\pi x_{i,p}). \tag{A.10}$$

Figure A.13: *Rastrigin function*



V.8. Drop-Wave function

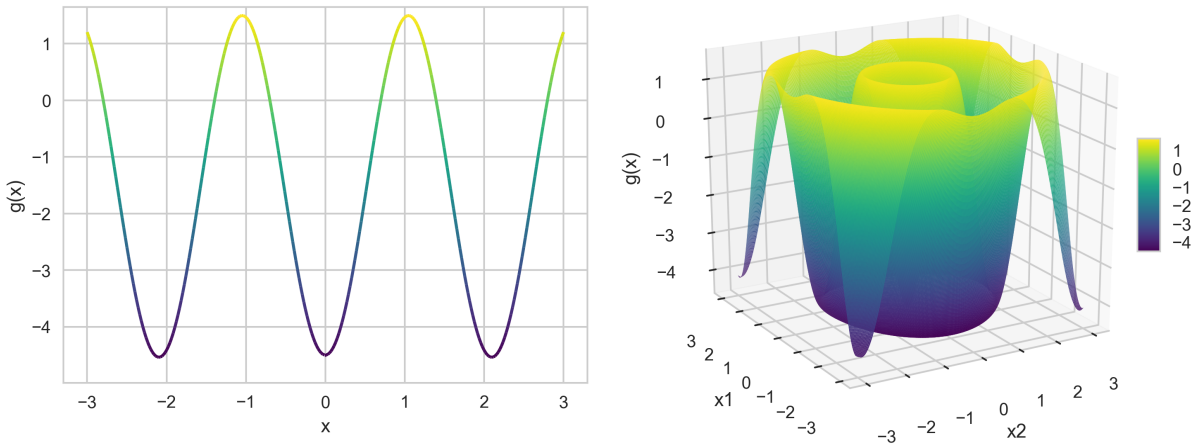
The Drop-wave function is once again complex and multimodal, and leads to shape of waves emerging from a center (Surjanovic & Bingham, 2017). Hence the name.

$$\begin{aligned} g(\boldsymbol{x}_i, \boldsymbol{\beta}) &= a\left(-\frac{b + \cos\left(c\sqrt{\sum_{p=1}^{k}|\beta_{p,1}|x_{i,p}^2}\right)}{d + e\sum_{p=1}^{k}\beta_{p,2}x_{i,p}^2} + f\right), \\ \frac{\partial g(\boldsymbol{x}_i, \boldsymbol{\beta})}{\partial x_{i,p}} &= \frac{ac\sin\left(c\sqrt{\sum_{p=1}^{k}|\beta_{p,1}|x_{i,p}^2}\right)}{\sqrt{\sum_{p=1}^{k}|\beta_{p,1}|x_{i,p}^2}\left(d + e\sum_{p=1}^{k}\beta_{p,2}x_{i,p}^2\right)}|\beta_{p,1}|x_{i,p} \\ &\quad + \frac{2ae\left(b + \cos\left(c\sqrt{\sum_{p=1}^{k}|\beta_{p,1}|x_{i,p}^2}\right)\right)}{\left(d + e\sum_{p=1}^{k}\beta_{p,2}x_{i,p}^2\right)^2}\beta_{p,2}x_{i,p}. \end{aligned} \tag{A.11}$$

There are $2k+5$ free parameters (since a is not identified),but again, I fix $\beta_{p,1}=1$ to keep out of root-troubles and set the scale parameters to ensure meaningful values.

$$
\begin{aligned}
\frac{\partial g(\boldsymbol{x}_i,\boldsymbol{\beta})}{\partial \beta_{p,1}} &= \frac{ac\sin\left(c\sqrt{\sum_{p=1}^{k}|\beta_{p,1}|x_{i,p}^2}\right)}{2\sqrt{\sum_{p=1}^{k}|\beta_{p,1}|x_{i,p}^2}\left(d+e\sum_{p=1}^{k}\beta_{p,2}x_{i,p}^2\right)}\frac{\beta_{p,1}}{|\beta_{p,1}|}x_{i,p}^2,\\
\frac{\partial g(\boldsymbol{x}_i,\boldsymbol{\beta})}{\partial \beta_{p,2}} &= \frac{ae\left(b+\cos\left(c\sqrt{\sum_{p=1}^{k}|\beta_{p,1}|x_{i,p}^2}\right)\right)}{\left(d+e\sum_{p=1}^{k}\beta_{p,2}x_{i,p}^2\right)^2}x_{i,p}^2,\\
\frac{\partial g(\boldsymbol{x}_i,\boldsymbol{\beta})}{\partial b} &= -\frac{a}{d+e\sum_{p=1}^{k}\beta_{p,2}x_{i,p}^2},\\
\frac{\partial g(\boldsymbol{x}_i,\boldsymbol{\beta})}{\partial c} &= \frac{a\sin\left(c\sqrt{\sum_{p=1}^{k}|\beta_{p,1}|x_{i,p}^2}\right)}{d+e\sum_{p=1}^{k}\beta_{p,2}x_{i,p}^2}\sqrt{\sum_{p=1}^{k}|\beta_{p,1}|x_{i,p}^2},\\
\frac{\partial g(\boldsymbol{x}_i,\boldsymbol{\beta})}{\partial d} &= \frac{a\left(b+\cos\left(c\sqrt{\sum_{p=1}^{k}|\beta_{p,1}|x_{i,p}^2}\right)\right)}{\left(d+e\sum_{p=1}^{k}\beta_{p,2}x_{i,p}^2\right)^2},\\
\frac{\partial g(\boldsymbol{x}_i,\boldsymbol{\beta})}{\partial e} &= \frac{a\left(b+\cos\left(c\sqrt{\sum_{p=1}^{k}|\beta_{p,1}|x_{i,p}^2}\right)\right)}{\left(d+e\sum_{p=1}^{k}\beta_{p,2}x_{i,p}^2\right)^2}\sum_{p=1}^{k}\beta_{p,2}x_{i,p}^2,\\
\frac{\partial g(\boldsymbol{x}_i,\boldsymbol{\beta})}{\partial f} &= a.
\end{aligned}
\tag{A.12}
$$

Figure A.14: *Drop-Wave function*



Generally, for the non-identified parameters (for which gradients are not derived) I simply insert ones in maximum likelihood (letting other coefficients adjust). They are still maintained because they make it easier to achieve well-behaved functions in the DGP.

VI. Derivation of the marginal effects in proposition 4.1

The following derives the expression for the partial derivatives. A simpler version for a single hidden layer is derived in the thesis (Bjørn, 2018). An algorithmic version more useful for implementation is provided in appendix IV. Start from the expression for neural network output from definition 4.1:

$$\tilde{f}(\boldsymbol{x}_i) = \sigma\left(\beta_0^{(L)} + \sum_{j_{L-1}=1}^{J^{(L-1)}} \beta_{j_{L-1}}^{(L)} \eta\left(\ldots \eta\left(\beta_{j_2,0}^{(2)} + \sum_{j_1=1}^{J^{(1)}} \beta_{j_2,j_1}^{(2)} \eta\left(\beta_{j_1,0}^{(1)} + \sum_{p=1}^{k} \beta_{j_1,p}^{(1)} x_{i,k}\right)\right)\right)\right). \tag{A.13}$$

Denote the linear signals by $s_{i,j_1}^{(1)} = \beta_{j_1,0}^{(1)} + \sum_{p=1}^{k} \beta_{j_1,p}^{(1)} x_{i,k}$ and $s_{i,j_2}^{(2)} = \beta_{j_2,0}^{(2)} + \sum_{j_2=1}^{J^{(1)}} \beta_{j_2,j_1}^{(2)} \eta\left(s_{i,j}^{(1)}\right)$, ..., $s_i^{(L)} = \beta_0^{(L)} + \sum_{j_{L-1}=1}^{J^{(L-1)}} \beta_{j_{L-1},j_{L-2}}^{(2)} \eta\left(s_{i,j_{L-2}}^{(1)}\right)$. The result is obtained by the chain rule:

$$\frac{\partial \tilde{f}(\boldsymbol{x}_i)}{\partial x_{i,p}} = \frac{\partial \sigma\left(s_i^{(L)}\right)}{\partial s_i^{(L)}} \cdot \frac{\partial s_i^{(L)}\left(\boldsymbol{s}_i^{(L-1)}\right)}{\partial x_{i,p}} \tag{A.14}$$

$$= \sigma'\left(s_i^{(L)}\right)\left(\sum_{j_{L-1}=1}^{J^{(L-1)}} \beta_{j_{L-1}}^{(L)} \frac{\partial \eta(s_{i,j_{L-1}}^{(L-1)})}{\partial x_{i,p}}\right) \tag{A.15}$$

$$= \sigma'\left(s_i^{(L)}\right)\left(\sum_{j_{L-1}=1}^{J^{(L-1)}} \beta_{j_{L-1}}^{(L)} \frac{\partial \eta(s_{i,j_{L-1}}^{(L-1)})}{\partial s_{i,j_{L-1}}^{(L-1)}} \cdot \frac{\partial s_{i,j_{L-1}}^{(L-1)}}{\partial x_{i,p}}\right) \tag{A.16}$$

$$= \sigma'\left(s_i^{(L)}\right)\left(\sum_{j_{L-1}=1}^{J^{(L-1)}} \beta_{j_{L-1}}^{(L)} \eta'\left(\boldsymbol{s}_i^{(L-1)}\right) \cdot \ldots \cdot \frac{\partial \eta(s_{i,j_2}^{(2)})}{\partial x_{i,p}}\right) \tag{A.17}$$

$$= \sigma'\left(s_i^{(L)}\right)\left(\sum_{j_{L-1}=1}^{J^{(L-1)}} \beta_{j_{L-1}}^{(L)} \eta'\left(\boldsymbol{s}_i^{(L-1)}\right) \cdot \ldots \cdot \frac{\partial \eta(s_{i,j_2}^{(2)})}{\partial s_{i,j_2}^{(2)}} \cdot \frac{\partial s_{i,j_2}^{(2)}}{\partial x_{i,p}}\right) \tag{A.18}$$

$$= \sigma'\left(s_i^{(L)}\right)\left(\sum_{j_{L-1}=1}^{J^{(L-1)}} \beta_{j_{L-1}}^{(L)} \eta'\left(\boldsymbol{s}_i^{(L-1)}\right) \ldots \eta'\left(\boldsymbol{s}_i^{(2)}\right)\left(\sum_{j_1=1}^{J^{(1)}} \beta_{j_2,j_1}^{(2)} \frac{\partial \eta\left(s_{i,j}^{(1)}\right)}{\partial x_{i,p}}\right)\right) \tag{A.19}$$

$$= \sigma'\left(s_i^{(L)}\right)\left(\sum_{j_{L-1}=1}^{J^{(L-1)}} \beta_{j_{L-1}}^{(L)} \eta'\left(\boldsymbol{s}_i^{(L-1)}\right) \ldots \eta'\left(\boldsymbol{s}_i^{(2)}\right)\left(\sum_{j_1=1}^{J^{(1)}} \beta_{j_2,j_1}^{(2)} \frac{\partial \eta\left(s_{i,j}^{(1)}\right)}{\partial s_{i,j}^{(1)}} \frac{\partial s_{i,j}^{(1)}}{\partial x_{i,p}}\right)\right) \tag{A.20}$$

$$= \sigma'\left(s_i^{(L)}\right)\left(\sum_{j_{L-1}=1}^{J^{(L-1)}} \beta_{j_{L-1}}^{(L)} \eta'\left(\boldsymbol{s}_i^{(L-1)}\right) \ldots \eta'\left(\boldsymbol{s}_i^{(2)}\right)\left(\sum_{j_1=1}^{J^{(1)}} \beta_{j_2,j_1}^{(2)} \eta'\left(s_{i,j_1}^{(1)}\right) \beta_{j_1,p}^{(1)}\right)\right). \tag{A.21}$$

The dots show repeated applications of the chain rule in the same manner for all layers in between. For each, the outer derivative is computed based on the linear signals, and a sum of weighted inner derivatives is computed. And on I go.

Consider the matrix notation instead.

$$\tilde{f}(\boldsymbol{x}_i) = \sigma\left(\eta\left(\ldots\eta\left(\eta\left(\boldsymbol{x}_i\,\boldsymbol{\beta}^{(1)}\right)\boldsymbol{\beta}^{(2)}\right)\ldots\right)\boldsymbol{\beta}^{(L)}\right). \tag{A.22}$$

Let $\boldsymbol{s}_i^{(1)} = \boldsymbol{x}_i\,\boldsymbol{\beta}^{(1)}$ and $s_i^{(2)} = \eta(s_i^{(1)})\,\boldsymbol{\beta}^{(2)}$, ..., $s_i^{(L)} = \eta(s_i^{(L-1)})\,\boldsymbol{\beta}^{(L)}$. Apply the chain rule:

$$\frac{\partial\tilde{f}(\boldsymbol{x}_i)}{\partial x_{i,p}} = \frac{\partial\sigma\left(s_i^{(L)}\right)}{\partial s_i^{(L)}}\cdot\frac{\partial s_i^{(L)}\left(\boldsymbol{s}_i^{(L-1)}\right)}{\partial x_{i,p}} \tag{A.23}$$

$$= \sigma'\left(s_i^{(L)}\right)\left(\frac{\partial\eta\left(\boldsymbol{s}_i^{(L-1)}\right)}{\partial x_{i,p}}\,\boldsymbol{\beta}_{1:}^{(L)}\right) \tag{A.24}$$

$$= \sigma'\left(s_i^{(L)}\right)\left(\frac{\partial\eta\left(\boldsymbol{s}_i^{(L-1)}\right)}{\partial\,\boldsymbol{s}_i^{(L-1)}}\mathrm{diag}\left(\frac{\partial\,\boldsymbol{s}_i^{(L-1)}}{\partial x_{i,p}}\right)\boldsymbol{\beta}_{1:}^{(L)}\right) \tag{A.25}$$

$$= \sigma'\left(s_i^{(L)}\right)\left(\eta'\left(\boldsymbol{s}_i^{(L-1)}\right)\mathrm{diag}\left(\ldots\mathrm{diag}\left(\frac{\partial\eta\left(\boldsymbol{s}_i^{(L-1)}\right)}{\partial x_{i,p}}\right)\boldsymbol{\beta}_{1:}^{(2)}\ldots\right)\boldsymbol{\beta}_{1:}^{(L)}\right) \tag{A.26}$$

$$= \sigma'\left(s_i^{(L)}\right)\left(\eta'\left(\boldsymbol{s}_i^{(1)}\right)\mathrm{diag}\left(\ldots\mathrm{diag}\left(\frac{\partial\eta\left(\boldsymbol{s}_i^{(1)}\right)}{\partial\,\boldsymbol{s}_i^{(1)}}\mathrm{diag}\left(\frac{\partial\,\boldsymbol{s}_i^{(1)}}{\partial x_{i,p}}\right)\right)\boldsymbol{\beta}_{1:}^{(2)}\ldots\right)\boldsymbol{\beta}_{1:}^{(L)}\right) \tag{A.27}$$

$$= \underbrace{\sigma'\left(s_i^{(L)}\right)}_{1\times1}\underbrace{\eta'\left(\boldsymbol{s}_i^{(L-1)}\right)}_{1\times J^{(L-1)}}\underbrace{\mathrm{diag}\left(\begin{array}{c}\ldots\mathrm{diag}\Bigg(\underbrace{\underbrace{\eta'\left(\boldsymbol{s}_i^{(1)}\right)}_{1\times J^{(1)}}\underbrace{\mathrm{diag}\left(\boldsymbol{\beta}_p^{(1)}\right)}_{J^{(1)}\times J^{(1)}}\underbrace{\boldsymbol{\beta}_{1:}^{(2)}}_{J^{(1)}\times J^{(2)}}}_{J^{(2)}\times J^{(2)}}\Bigg)\ldots\end{array}\right)}_{J^{(L-1)}\times J^{(L-1)}}\underbrace{\boldsymbol{\beta}_{1:}^{(L)}}_{J^{(L-1)}\times1}. \tag{A.28}$$

The diagonal enters the fray because the derivative for a function applied *element-wise* to a vector (as $\eta(\cdot)$ is) is a diagonal matrix of the derivative (Clark, 2018). It can be seen most easily by noting that without it, the dimensions would not match in the final calculation. Intuitively, note that multiplication by a diagonal matrix is the same as element-wise multiplication by the diagonal. Therefore each $\eta'(s_{i,k}^{(\ell)})$ is multiplied by its own derivative, as we'd like. $\boldsymbol{\beta}_{1:}^{(\ell)}$ excludes the intercept $\beta_0^{(\ell)}$ of $\boldsymbol{\beta}^{(\ell)}$, because it is accompanied by a zero in the derivative.