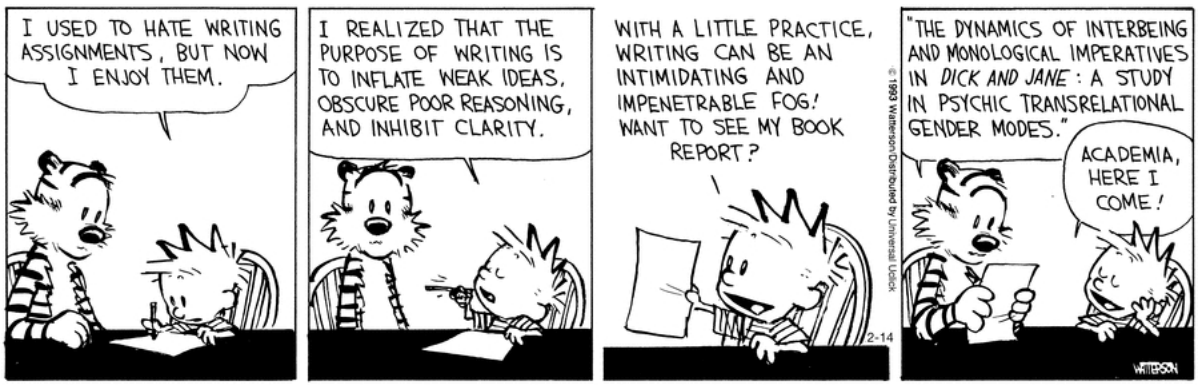
# Neural networks as an econometric estimator

A simulation study of the properties of neural networks
as an estimator of conditional marginal effects.

A master's thesis in *Economics* at the University of Copenhagen by

**R. Bjørn**

August 2018




**Abstract**

This paper argues that *neural networks* provide a well-behaved estimator of the marginal effects of conditional expectation functions, which impose no assumption of functional form, and therefore allows fully *heterogeneous* effects. This allows estimation of *causal effects* under assumptions similar to those employed in the microeconometric literature on treatment evaluation. Although neural networks (a popular *machine learning* technique often called *deep learning)* have already spread to econometrics, their partial derivatives have not received much attention. I investigate the properties of these in a simulation framework based on nine data-generating processes ranging from linear to highly complex. I find that under *unconfoundedness* they provide consistent estimates of individual marginal effects of continuous regressors in all explored cases, although they require relatively large samples. I also define a two-stage *instrumental variables* procedure based on neural networks, and show this provides consistent estimates in the face of *confounders*. Finally, I provide a bootstrapping procedure which allows statistical inference. Even though the scope of my simulation study is limited, this provides evidence that neural networks are a well-behaved big data estimator, which allows assumption-free analysis of heterogeneous marginal effects. I illustrate this by replicating a study on the returns to schooling by Angrist & Krueger (1991), where neural networks obtain similar average effects, but allows a more nuanced analysis of how the returns differ based on current education. All in all, this implies that neural networks may provide a relevant supplement to modern microeconometric research.

## Contents

## 1. Introduction: Neural networks might just be the new regression

'As always in life, people want a simple answer ... and it's always wrong'.

– *Susan Greenfield (Guardian, 2014)*

The argument presented in this paper can be concisely stated: Neural networks can do anything an econometrician would expect of a regression, without imposing assumptions on the functional form of the relationship in question. No matter the underlying process, they provide a consistent estimator for both the conditional expectation and its derivatives, the marginal effects which are often the object of interest in econometric studies. This representational capacity is bought through low efficiency, but networks do allow analysis of medium-sized and big data. I argue that neural networks provide an attractive supplement to existing estimation techniques since they allow assumption-free analysis of heterogeneous marginal effects, given enough data. However, they do not provide a magical machine which produces correct answers with no supervision. They are subject to most of the same issues that can be detrimental to identification of causal effects with traditional estimators, such as measurement error and omitted variable bias. Fortunately, existing solutions (such as instrumental variables) can be extended to neural networks. Consequently, they provide a relevant estimator in modern microeconometric research.

### 1.1. Contribution: Properties & illustration of a new estimator

Neural networks are immensely popular in the machine learning community (often under the hipper name of *deep learning*), where they are used with great success in both academic and commercial settings, in numerous applications ranging from image recognition (Simonyan & Zisserman, 2014) to conquering the game of Go (Silver *et al.*, 2016). Their use is also entering the world of econometrics (see e.g. Hartford *et al.* (2016)). Despite this, there has been little focus on the properties of the *marginal effects* of explanatory variables in neural networks, even though it is well-known that partial derivatives of input variables are well-defined and easily derivable (see e.g. Goodfellow *et al.* (2016)). This is likely due to the fact that machine learning is focused on prediction, and data scientists hence put little stock in deriving properties of partial derivatives. Current machine learning in econometrics is either focused on how prediction may be useful, or uses expectations in treatment evaluations, as I review below. However, I argue that theoretical interest in economics often lies in the marginal effects for conditional expectations, and propose the partial derivatives in neural networks as a natural estimator of these.

My main contribution is an analysis of the properties of this estimator, which has previously been largely ignored in the econometric literature. I use a series of simulation experiments to show that neural networks provide a well-behaved estimator for a range of data-generating processes (going from linear to highly complex), indicating that they are generally usable. I also design a two-stage estimator based on instrumental variables, and show how this can eliminate bias from confounders in a series of experiments, and a replicated study.

As a less formal contribution, I hope to illustrate that researchers comfortable with regression analysis should be able to add neural networks to their toolbox with relative ease. I argue that more technical aspects can be safely left to computer scientists, and provide an introduction aimed at showing the familiarity of most used concepts, and the relevance of a few new ideas.

### 1.2. Roadmap: The paper in one page

Section 2 reviews the existing literature at the intersection between machine learning and econometrics, and provides the theoretical framework for the paper. I place myself within a camp of econometricians who believe machine learning algorithms can be leveraged for causal estimation. I base this view on the fact that both econometrics and machine learning are interested in estimation of *conditional expectations*. Econometricians have the further goal of assigning a causal interpretation, but this is typically done through assumptions on the data, not on the estimation technique. I review a set of sufficient assumptions for estimation and causal interpretation, and define a formal framework for evaluating estimators. The framework accounts for the fact that the optimal estimator for a given sample may only be able to approximate the true conditional expectations, a notion often ignored in econometrics. This opens the doors for neural networks, which provide *universal approximators* to any function.

Section 3 provides the promised introduction to neural networks. I illustrate the approximation capacity of neural networks through a simple example, and provide a formal framework for the simplest version of a neural network, which basically combine a single set of 'regression-like' models in an optimal manner based on a maximum likelihood framework. The main challenges of relevance to econometric practitioners are failed optimization, overfitting and choice of network architecture, which are reviewed. Finally, I extend the formal framework to *deep* networks, which may perform better in practice.

Section 4 covers the methodology of the study, leaving most technicalities for the appendices. I implemented a sophisticated Monte Carlo simulation set-up, capable of representing a general class of continuous and binary scenarios. I use this to analyse a set of nine data-generating processes, ranging from completely linear to highly complex.

Section 5 analyses the performance of neural networks as an estimator of marginal effects in a 'clean' environment, where all reviewed assumptions are fulfilled. Neural networks provide a consistent estimator in all scenarios, and is typically efficient enough to allow analysis. This allows for completely heterogeneous causal effects, a large advantage over linear methods which are only capable of recovering average effects. I also provide a bootstrapping procedure for evaluating sampling (and training) variation, and show how this allows hypothesis testing and other robustness checks. The end result is a well-behaved estimator with all the moving parts.

Section 6 shows how neural networks respond to classical issues much as a linear regression would: Measurement error leads to attenuation bias, and confounders lead to omitted variable bias. There is no reason to fret though - I provide a simple two-stage instrumental variables procedure, and show how this restores consistency. I illustrate this further by replicating a classical study by Angrist & Krueger (1991), where they use instruments based on a natural experiment in the form of compulsory schooling to estimate the returns to education. I replicate their average effects using neural networks, but argue that their analysis may miss significant heterogeneity in the returns to extra schooling, which I find diminish for higher education.

All in all, I conclude neural networks provide a well-behaved estimator of marginal effects given relatively big data, which may be extended to modern research designs. It is highly relevant for applied research, as it allows for assumption-free analysis of the heterogeneity of effects.

## 2. Theory: Econometrics vs. machine learning

'Applied econometrics . . . is the original data science'.

*– Joshua D. Angrist & Jörn-Steffen Pischke (2014)*

Applied econometrics and machine learning have fundamentally different goals – prediction versus estimation of causal effects – but the road to these goals may overlap more than it appears at first glance. Although there is some disagreement, an emerging literature argues that machine learning algorithms can be leveraged for causal estimation, as I review below. I provide a simple argument for this view by noting that optimal prediction relies on modelling the conditional expectation function (CEF), which is the object of interest for most econometric studies. I provide the traditional assumptions that allow causal interpretation of such expectations, and argue that they do not depend on the estimation technique. I then provide a formal framework where estimators are judged by the mean squared error of their estimates, and argue that approximation methods such as neural networks may provide a superior estimator within it.

### 2.1. A brief review: Optimists & pessimists on causal learning

Machine learning is already becoming popular in econometrics, but there is some disagreement as to what it can be used for.[1] Some argue that machine learning is good at what it does but can do no more. Others believe that machine learning offers new tools to shed light on the age-old question of causality. This paper places itself squarely in the latter camp.

#### 2.1.1. Goin' native: Prediction problems in econometrics

One camp of economists argue that machine learning may be useful for economic problems concerning prediction, but is unlikely to be helpful for causal inference.[2] Even if not explicitly stated, many econometricians implicitly exhibit this view when they 'go native', and only use the tools to solve prediction problems in the same manner a data scientist would.[3] The view is succintly summarized by Mullainathan & Spiess (2017): 'Machine learning not only provides new tools, it solves a different problem'. Such authors are not dismissive of machine learning in econometrics - indeed, no matter whether they have gone explicitly or implicitly native, they are quite excited by the possibilities machine learning offers. There is reason for excitement, as there are indeed many economic problems which can be described as purely predictive.

One such area which has received attention is *prediction policy problems*.[4] Policy problems are not always causal, but may only require the policy-maker to anticipate some response to take an optimal decision (Kleinberg *et al.*, 2015). The key assumption is that the policy-maker cannot herself affect the outcome, leaving a pure prediction problem (Chakraborty & Joseph, 2017). In such cases, machine learning techniques are argued to be applicable pretty much 'off the shelf' (Athey, 2017). Recent examples of such research include prediction of loan repayment using mobile phone data (Björkegren & Grissen, 2017), bankruptcy prediction using accounting data

[1] For surves of machine learning in econometrics see Athey (2017) and Mullainathan & Spiess (2017).

[2] Mullainathan & Spiess (2017) elegantly state an argument of this sort.

[3] An example is the textbook introduction to social data science provided in Foster *et al.* (2017). The book does an excellent job showing how new tools can provide new data for traditional questions, but when it covers machine learning it is solely focused on prediction, and might as well be offered to data scientists.

[4] For surveys of recent work in the area, see Athey (2017) or Mullainathan & Spiess (2017).

(Barboza *et al.*, 2017), targeting mentoring interventions using prediction of which youths are high-risk (Chandler *et al.*, 2011) and personnel decisions using prediction of worker productivity (Chalfin *et al.*, 2016). Such questions are certain to be relevant for practitioners. Researchers, however, are still likely to ultimately be interested in the *causes* of all these events.

Another area of interest is that of *inferring new data.*[5] Machine learning can be used to take unconventional, high-dimensional data and condense it down to variables which can be used with traditional econometric techniques (Mullainathan & Spiess, 2017). Natural language processing allows researchers to translate text into relevant data (Evans & Aceves, 2016). Image recognition allows one to use satellite pictures to predict economic output (Donaldson & Storeygard, 2016). The new approaches can also allow us to observe old school data sources, but for areas and people where data was previously unavailable or unreliable. For instance, Blumenstock (2016) suggest using data from satellites, the internet and mobile phones to track poverty in developing countries, where national surveys are infeasible. Such measurements are valuable in and of themselves, but if the predictions are precise enough, they may also be used for causal estimation.

Finally, it is argued that prediction can be useful *in service of estimation.* Many estimation techniques involve intermediate steps which are just prediction problems. One example is the first stage of two-stage least squares instrumental variables, which is essentially a prediction problem of the endogenous regressor given the instruments (Mullainathan & Spiess, 2017). Varian (2014) argues that a prediction problem also occurs in the estimation of causal impact, because it requires comparison with an unobserved counterfactual effect, which must be predicted. Several of the approaches below also use prediction as part of estimation, even if they go further.

While the problems emphasized by this camp are certainly interesting, they do a poorer job when explaining why the buck must stop here. 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'. While this is certainly a valid concern - not all machine learning algorithms will provide consistent effects - there is no reason to dismiss these properties out of hand. Rather, we should analyse which estimators might actually be useful. The authors list several concerns. First, machine learning traditionally does not provide standard errors. This is a concern, but it is quite possible we could derive them. Second, regularization may mean that the influence of some variables is diminished or even removed. This is certainly a problem if true effects are masked or biased - however, note that if regularization is used properly, it is used to ensure the algorithm reflects the underlying process. In this sense, regularization may actually improve estimation. Finally, the main concern is that due to the great representational capacity of algorithms, the choice between very different models may come down to 'the flip of a coin' (Mullainathan & Spiess, 2017, p. 98). It is true that many algorithms may stop in various local minima, rather than the true optimum. However, again, there is no assurance that such minima would be worse than the optimal solution of a model which cannot reflect the underlying structure. All in all, the concerns listed are serious, and should be considered. However, we should analyse whether the algorithms are subject to them, and whether we can adjust them if they are, rather than simply dismiss their use out of hand.

[5] Mullainathan & Spiess (2017) provides a brief survey of this literature.

### 2.1.2. Gettin' busy: Adapting machine learning for causal inference

Some econometricians agree with this view.[6] Optimists, one might call them, who believe machine learning offers tools which can be adapted for our purposes, even if they are not ready out of the box. There is already a burgeoning literature which seeks to 'harness the strength of [machine learning] algorithms to solve causal inference problems' (Athey, 2017, p. 10).

Tools are being adapted to estimate *average treatment effects*. There is a large literature on estimation of such under versions of the unconfoundedness assumption I introduce later (Athey & Imbens, 2017). There has long been a tradition in this literature for the use of semiparametric methods, which do not impose functional form assumptions (Athey, 2017). This makes it easy to allow for the introduction of flexible algorithms. One approach is to use sparse, regularized regression methods (e.g. LASSO) to handle a large number of covariates (see Belloni *et al.* (2014), Chernozhukov *et al.* (2015) and Athey *et al.* (2016a)). Another recent contribution is that of Chernozhukov *et al.* (2017), who propose 'double machine learning'. The core method follows an approach by Robinson (1988), but allows replacement of kernel regression with any consistent learning algorithm which converges at rate $n^{\frac{1}{4}}$. While such contributions are certainly interesting and should be explored, it should be noted that if causal variables are low dimension (binary, or few treatment values), existing estimators based on matching or the propensity score already do a good job of estimating average treatment effects (Athey & Imbens, 2017).

A more promising venue is on *heterogeneous treatment effects*. Machine learning may allow one to better investigate whether effects vary depending on covariate values without specifying the form of heterogeneity (Athey, 2017). There has been recent work uncovering relatively simple shapes of heterogeneity through 'causal trees', which estimates treatments in partitions of the covariate space provided by decision trees, which optimize with regard to revealing heterogeneity in effects (Athey & Imbens, 2016). The method has also been extended to leverage ensemble methods through 'causal forests' (Wager & Athey, 2017). These estimators may provide insights into heterogeneity, but they do not provide fully fledged nonparametric estimator of heterogeneity. Athey *et al.* (2016b) suggests a local estimator called 'generalized random forests', a method which uses decision trees to substitute the kernel weighting function in traditional local linear regression. Similar methods can be applied anywhere a kernel function might be applied (Athey, 2017). Hartford *et al.* (2016) suggests a two-stage procedure for instrumental variables based on neural networks, where networks are first used for treatment prediction, and then a network is estimated where the loss function is altered to integrate over the conditional treatment distribution.

This paper places itself in extension of the last literature, with the belief that machine learning is well-placed to provide fully nonparametric estimators. Compared to Hartford *et al.* (2016), I start from neural networks far more 'off the shelf'. My view is that we should first check whether the basic networks work well, before we alter their underlying objectives. Compared to the literature in general, I focus less on the case of low-dimensional treatment effects, and instead investigate continuous variables. If results hold for them, it should be straightforward to extend them to discrete variables. The next sections sets up the theoretical background for such an endeavour, simultaneously providing arguments why machine learning is a relevant approach.

[6] For an argument along these lines, and a survey of recent studies, see Athey (2017).

## 2.2. The irresistible allure of the conditional expectation function

The end goal of applied econometrics and machine learning is not identical, but may overlap more than it appears at first glance. We are usually interested in the the identification and estimation of causal effects, whereas data scientists seek to allow an automated program to improve performance on some task by learning from data. Below, I argue that optimal answers to either of these questions relies on estimation of the conditional expectation function (CEF). Causal identification then relies on theoretical arguments, which are equally valid for any technique as long as it obtains the CEF. This provides a theoretical justification for the use of (some) learning algorithms, in line with the view of Athey (2017), who believes machine learning will provide new estimators, but 'no fundamental changes to the theory of identification of causal effects'.

### 2.2.1. Questions may be causal, but estimation is just statistics

Econometrics can be many things, but many econometricians believe the end goal is to answer causal questions.[7] Descriptive and predictive statistics may certainly be relevant, as outlined above, but strong answers to them will inevitably lead to new causal questions: we may be interested in predicting which students are prone to overdrinking for targeted responses (or beer ads), but a natural next endeavour would be to figure out *why* they do so.

The literature on policy evaluation within applied microeconometrics arguably provide the best causal estimates economics can boast of.[8] The influence of the literature expanded greatly from the late eighties and onward, leading to 'a credibility revolution, with a consequent increase in policy relevance and scientific impact' (Angrist & Pischke, 2010, 4). The key notion is that causal estimation is difficult, and we should be realistic about what we can accomplish in a given research scenario. The literature focus on strong *research designs* through explicit *identification strategies* for estimation of causal effects (Athey & Imbens, 2017). Such strategies typically rely on a *experimental ideal*, a fictitious research design which would certainly identify the causal effects (Angrist & Pischke, 2008, ch. 2). However, due to the more or less unique challenges of econometrics, these ideals are typically unobtainable due to financial and/or moral concerns, or even impossible (e.g. if key variables are inherently unobservable). Researchers then look for observational data which can support assumptions and estimation which achieve causal identification outside the ideal. The focus on designing and defending identification strategies fosters both strong answers, as well as transparency around which assumptions results rely on.

The literature typically defines causality using the *potential outcomes* framework[9] (also called the *Rubin causal model*).[10] Consider an outcome of interest $y_i \in \mathcal{Y}$ (for observation $i$), which could be wages ($\mathcal{Y} = \mathbb{R}_+$) or whether or not a college student blacks out ($\mathcal{Y} = \{0, 1\}$). A variable $x_i \in \mathcal{X}$ - which could be education or number of beers drunk - has a causal effect on $y_i$ if an *exogenous* change only in $x_i$ induces a subsequent change in $y_i$. This follows an old notion typically stated as 'no causation without manipulation' (Holland *et al.*, 1985; Rubin, 1975), which stresses that

[7] For an eloquent argument along these lines, see Angrist & Pischke (2008).

[8] 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).

[9] Pearl (2000) provides a graph-based approach to causality instead. The approaches are not decidedly in conflict, nor do they completely overlap. I will focus on the more popular approach in econometrics.

[10] Angrist & Pischke (2008) provides a brief introduction. Imbens & Rubin (2015) provides a thorough discussion.

causal effects are well-defined if it is hypothetically possible to manipulate $x_i$ (Imbens & Rubin, 2015, 21). Define a a *potential outcome* $y_i' = f_i(x_i')$ for any possible $x_i' \in \mathcal{X}$. A causal effect implies that a change from $x_i'$ to different value $x_i''$ tends to imply a different outcome $y_i' \neq y_i''$. We typically summarize such relationships using conditional expectations (Angrist & Pischke, 2008). The causal effect $\gamma_i$ of any specific exogenous change $x_i' \to x_i''$ may then be written as

$$\gamma_i(x_i', x_i'') = \mathrm{E}[f_i(x_i'') - f_i(x_i')] = \mathrm{E}[y_i|x_i''] - \mathrm{E}[y_i|x_i']. \tag{2.1}$$

The literature has largely focused on the case of 'treatment' variables, which are either binary (treated or untreated) or takes relatively few values (Athey & Imbens, 2017). In such cases, it possible to work with finite differences of the sort above. If $x_i$ is continuous, the natural extension is to instead consider *marginal causal effects*.

**Definition 2.1.** The *marginal causal effect* on $y_i \in \mathcal{Y}$ of $x_i \in \mathcal{X}$ is the change in expectation induced by an infinitesimal *exogenous* change in $x_i$ (e.g. by direct manipulation):

$$\gamma_i(x_i) = \frac{\partial \,\mathrm{E}[f_i(x_i)]}{\partial x_i} = \frac{\partial \,\mathrm{E}[y_i|x_i]}{\partial x_i}. \tag{2.2}$$

Note the explicit mention of exogenous variation. Conditional expectations by themselves only model statistical dependencies, which may or may not be causal. This is where causality parts way from mere prediction - for instance, while wage may help predict one's age, it could never cause it. The classical way to ensure exogenous variation is to manipulate it within an experiment. If this is not possible, one typically looks for exogenous variation by using natural experiments, or through clever research designs (e.g. instrumental variables). Usually, pure exogenous variation cannot be obtained. Instead, we look to isolate exogenous variation by controlling for covariates. Consider a vector of variables $\boldsymbol{x}_i \in \boldsymbol{\mathcal{X}}$ where we are interested in the causal effect of $x_{i,p} \in \boldsymbol{x}_i$. If we can ensure that $x_{i,p}$ is assigned independently from potential outcomes conditional on covariates $\boldsymbol{x}_{i,-p} = \boldsymbol{x}_i \setminus x_{i,p}$, then the remaining variation is exogenous (Angrist & Pischke, 2008, 43). This is ensured by *unconfoundedness*, defined for continuous cases by Imbens (2000).

**Definition 2.2.** Assignment of variable $x_{i,p}$ given covariates $\boldsymbol{x}_{i,-p}$ is *weakly unconfounded* if

$$\forall x_{i,p} \in \mathcal{X}_p : \quad D(x_{i,p}) \perp\!\!\!\perp f_i(x_i) | \, \boldsymbol{x}_{i,-p}, \qquad D(x) = 1\{x_{i,p} = x\}. \tag{2.3}$$

Violation of this implies that observed expectation differ from causal ones due to an effect typically called *selection bias*. Consider observing that people who participate in a job training programme are more likely to find work. One would then be tempted to conclude a positive effect of the programme. However, it may be that people willing to participate in the programme are those already looking for a job. The difference in job probability may then either be ascribed to the programme, or to fact that the participants were always more likely to find work. However, if we could assign participation independently of whether they are currently job seeking, this selection bias evaporates. Generally, consider excluding a variable $x_{i,j} \in \boldsymbol{x}_{i,-p}$ which affects $y_i$ and has $\mathrm{Cov}(x_{i,p}, x_{i,j}) \neq 0$. This violates unconfoundedness because an increase in $x_{i,j}$ is associated with an increase in both $x_{i,p}$ and $f_i(x_{i,p})$. It also invalidates causal interpretation, since the observed increase in $f_i(x_{i,p})$ is not caused by the increase in $x_{i,p}$.

**Proposition 2.1.** Assume that $x_{i,p} | \boldsymbol{x}_{i,-p}$ is weakly unconfounded (as in definition 2.2). The marginal causal effect $\gamma_i(x_{i,p})$ of $x_{i,p}$ on $y_i$ (as in definition 2.1) is then identified by the change in conditional expectation induced by an infinitesimal change in $x_{i,p}$

$$\gamma_i(x_{i,p}) = \frac{\partial \mathrm{E}[f_i(x_{i,p})]}{\partial x_{i,p}} = \frac{\partial \mathrm{E}[y_i | x_{i,p}, \boldsymbol{x}_{i,-p}]}{\partial x_{i,p}}. \tag{2.4}$$

*Proof.* A brief proof is provided in appendix V.1. □

The key point here is that causal identification is theoretical - once it is achieved, the statistical exercise is just to model the dependency through conditional expectations. Then we interpret this in a causal manner, hopefully justified by either direct or natural manipulation. This is not unique to program evaluation - most econometric exercises model conditional expectations, and assign causality through theory and research designs. Observational data simply cannot tell us whether an effect is causal - it can only tell us whether outcomes are dependent.

### 2.2.2. Machines may want to learn, but what they do is just statistics

Machine learning is a varied field, with many different techniques and goals.[11] Informally, one is interested in the construction of algorithms which are able to perform some task by automatically learning from data.[12] Although such a definition does not preclude the search for causality, data scientist have not focused much on it. Nor is it likely, given the importance of theoretical considerations for deducing causality indicated above, that a program can *automatically* deduce causality from observational data[13]. Instead, the tasks solved usually consist of *prediction* problems, which only require modelling statistical dependencies.

The subfield of machine learning most easily portable to econometrics is *supervised learning*. It is concerned with predicting a random variable $y_i$ based on a set of variables $\boldsymbol{x}_i$. Learning is supervised because the algorithms learn by being trained on a data set where both $y_i$ and $\boldsymbol{x}_i$ are provided (Goodfellow *et al.*, 2016). This should sound familiar to econometricians, but there are two critical differences. First, the focus is solely on optimal prediction. Second, performance is measured on *unseen* data, typically because the intended use is to suggest decisions when $y_i$ is not provided. Before I discuss optimal prediction, I may note that some consideration of out-of-sample performance might not be misguided for econometricians. The reason algorithms perform poorly out of sample is often that they *overfit* the data, learning to replicate noise rather than the underlying process. Econometricians certainly have no wish to replicate noise - indeed, every time an instructor warns not to add variables just to increase $R^2$, she is implicitly warning against overfitting. The likely reason this has not been an explicit concern is that our methods have not been prone to overfit due to low representational capacity. Indeed, with linear models, we have been more likely to *underfit*. However, if we import methods with greater capacity, we should take care not to overfit the data if we are interested in uncovering the true relationships.

[11] For introductions to the field, see Goodfellow *et al.* (2016) or Abu-Mostafa *et al.* (2012a).

[12] Formally: 'A program is said to learn from experience $E$ with respect to some class of tasks $T$ and performance measure $P$, if its performance at tasks in $T$, as measured by $P$, improves with experience $E$.' (Mitchell, 1997).

[13] Some algorithms are implicitly causal. Consider the algorithm built by by Google DeepMind to compete in Go (Silver *et al.*, 2016). This requires identifying moves which *cause* one to win. However, the setting here is implicitly experimental (as any 'move' implies direct manipulation), so few theoretical considerations are needed.

There is a theoretical reason optimal prediction and estimation of effects may directly overlap: The optimal predictor is often the CEF. The typical approach in machine learning is to minimize a *loss function*, which measures the prediction error in expectation, to stress out-of-sample performance. Common choices lead directly to the CEF.

**Definition 2.3.** The objective of supervised machine learning for a *regression* problem $y_i \in \mathbb{R}$ is to produce predictions $\hat{y}_i \in \mathcal{Y}$ which minimize the *mean-squared error* (MSE) loss

$$L_R(y_i, \hat{y}_i) = \mathrm{E}\left[(y_i - \hat{y}_i)^2\right]. \tag{2.5}$$

MSE provides a common basic loss function for continuous problems, although there are often loss functions more appropriate for particular cases. Its popularity stems partly from mathematical convenience, and partly from some nice theoretical properties (see section 5.1).

**Proposition 2.2.** The optimal predictor of $y_i$ given $\boldsymbol{x}_i$ given the objective in definition 2.3 is the conditional expectation. Let $m(\boldsymbol{x}_i)$ be any function of $\boldsymbol{x}_i$. Then

$$\mathrm{E}\left[y_i | \, \boldsymbol{x}_i\right] = \arg\min_{m(\boldsymbol{x}_i)} \mathrm{E}\left[(y_i - m(\boldsymbol{x}_i))^2\right]. \tag{2.6}$$

*Proof.* A proof is provided in appendix V.2, courtesy of Wasserman (2016). □

**Definition 2.4.** The objective of supervised machine learning for a *classification* problem $y_i \in \{0, 1\}$ is to produce predictions $\hat{y}_i \in \{0, 1\}$ which miminize the *classification error* loss

$$L_C(y_i, \hat{y}_i) = \mathrm{P}\left(y_i \neq \hat{y}_i\right). \tag{2.7}$$

For binary problems ($y_i \in \{0, 1\}$), the basic goal is typically to classify observations correctly. The basic measure of this is *0-1 loss*, where misspecified observations yield a loss (leading to simple accuracy as a measure). Measures which weigh errors differently (such as precision or recall) may be more fitting if the sample is unbalanced, but have similar goals overall.

**Proposition 2.3.** The optimal predictor of $y_i$ given $\boldsymbol{x}_i$ given the objective in definition 2.4 is the conditional expectation. Let $m(\boldsymbol{x}_i)$ be any function of $\boldsymbol{x}_i$. Then a classifier $1\left(m(\boldsymbol{x}_i) \geq 0.5\right)$ obtains the Bayes error rate if and only if it is based on the CEF:

$$1\left(E\left[y_i | \, \boldsymbol{x}_i\right] \geq 0.5\right) = \arg\min_{m(\boldsymbol{x}_i)} \mathrm{P}\left(y_i \neq 1\left(m(\boldsymbol{x}_i) \geq 0.5\right)\right). \tag{2.8}$$

*Proof.* A proof is provided in appendix V.3, courtesy of Wasserman (2016). □

This illustrates that, in typical cases, the CEF provides the theoretically optimal solution for prediction. This does not mean all learning algorithms are concerned with it[14], but many algorithms directly model the CEF, and may therefore be relevant for econometrics. One main caveat remains however: We are really interested in the marginal effects. Therefore, we should carefully check whether these are also well-behaved in the models.

[14]Support vector machines, for instance, look to do classify data points by achieving linear separation when dimensions are fictiously increased, with no direct probability estimate.

## 2.3. Improving estimation through universal approximation

How does one go about estimating the conditional expectation? The first section below reviews a few assumptions which ensures that it is possible to estimate the objects of theoretical interest to both econometrics and machine learning. The second section presents a formal framework for judging estimators. It differs slightly from the standard exposition, but the fundamental goals are the same. First, it explicitly focuses on estimation of *effects* rather than *parameters*. Second, it explicitly allows for estimation of *approximate* models, rather than nailing down the exact data-generating process (DGP). The final section presents an argument for why neural networks may perform well within this framework by presenting formal *universal approximation* theorems.

### 2.3.1. If people aren't similar, data is of no use

Variation is the lifeblood of statistics, and without untestable assumptions, we observe none. Consider again the potential outcome $y_i = f_i(x_{i,p})$ which (possibly) depends on the value of a causal variable $x_{i,p}$. We only observe one such value for any observation and without further assumptions it carries no information on other observations. This is the 'fundamental problem of causal inference' (Holland *et al.*, 1985, 959). Yet there is large number of unknowns implicit to this outcome. Previously, they were allowed implicitly through observation-specific outcome function $f_i(\cdot)$, but they can be made explicit. The causal effect of $x_{i,p}$ in a non-linear setting could easily depend on existing level of the variable itself, or of other covariates $\boldsymbol{x}_{i,-p}$. It could also plausible depend on the covariates of other observations, $\boldsymbol{x}_{-i}$, or even their outcomes $\boldsymbol{y}_{-i}$ (either potential or realized). Even accounting for such influences, the outcome is likely to remain stochastic, which can be made explicit through an error term $u_i$.

**Definition 2.5.** A linearly separable *data generating process* (DGP) may be written as

$$y_i = f_i\left(\boldsymbol{x}_i, \boldsymbol{x}_{-i}, \boldsymbol{y}_{-i},\right) + u_i = \mathrm{E}[y_i|\, \boldsymbol{x}_i, \boldsymbol{x}_{-i}, \boldsymbol{y}_{-i}] + u_i, \qquad \mathrm{E}[u_i|\, \boldsymbol{x}_i, \boldsymbol{x}_{-i}, \boldsymbol{y}_{-i}] = 0. \tag{2.9}$$

This definition imposes hardly more structure than nature.[15] Particularly, there is no assumption that it covers all variables relevant for $y_i$. Rather, it holds for any set of regressors $\boldsymbol{x}_i$, and states that we can express the part of $y_i$ which can be explained by these. This may range from completely stochastic if all regressors are independent of $\boldsymbol{x}_i$ ($y_i = u_i$) to completely deterministic if we can account for all features relevant for $y_i$. Furthermore it may or may not be causal for elements of $\boldsymbol{x}_i$, following the previous section.

The literature has focused on identifying average effects through the stable unit treatment value assumption (Rubin, 1980). This states that the potential outcomes for any one observation must not depend on the assignment of causal variables to other units, and that there are no unobservable differences in the causal variable which affects potential outcomes (Imbens & Rubin, 2015, p. 10). I invoke stronger assumptions, since these underlie the use of neural networks (and most learning algorithms). This may seem theoretically unappealing but I would argue that the assumptions are not that restrictive, and that they are often implicitly assumed in econometrics.

[15] It only imposes that errors are linearly separable in the remaining features, a representation which is implicitly common in econometrics, and which is not crucial for the results.

**Definition 2.6.** Outcomes are *independent* given covariates if

$$\forall i: \quad \mathrm{E}\left[y_i \middle| \boldsymbol{x}_i, \boldsymbol{x}_{-i}, \boldsymbol{y}_{-i}\right] = \mathrm{E}\left[y_i \middle| \boldsymbol{x}_i\right]. \tag{2.10}$$

This is fulfilled if individual observations are independent, a common assumption in both microeconometrics and supervised learning. However, it is not always true in practice, and should therefore be considered critically. Note however that any estimation technique which does not allow models to depend on other observations implicitly assume it.

**Definition 2.7.** Outcomes have a *common form* given covariates if

$$\forall i: \quad \mathrm{E}\left[y_i \middle| \boldsymbol{x}_i, \boldsymbol{x}_{-i}, \boldsymbol{y}_{-i}\right] = f_i\left(\boldsymbol{x}_i, \boldsymbol{x}_{-i}, \boldsymbol{y}_{-i}\right) = f\left(\boldsymbol{x}_i, \boldsymbol{x}_{-i}, \boldsymbol{y}_{-i}\right). \tag{2.11}$$

This does not imply that everyone responds identically to the causal variable of interest - it just states that if they do, this should be reflected through the covariates. If this is not so, then we have no hope of modelling the differences. The assumption may seem theoretically harsh but in practice, it typically represents the best we can do. Once again, any estimation technique which estimates the same model for everyone implicitly assumes this.

**Proposition 2.4.** If outcomes are independent given covariates with a common form, the conditional expectation function is identified by

$$\mathrm{E}\left[y_i \middle| \boldsymbol{x}_i\right] = f\left(\boldsymbol{x}_i\right). \tag{2.12}$$

The, if $x_{i,p} \middle| \boldsymbol{x}_{i,-p}$ is weakly unconfounded, the marginal causal effect $\gamma$ of $x_{i,p}$ is identified by

$$\gamma_{i,p}(x_{i,p}) = \gamma_p(\boldsymbol{x}_i) = \frac{\partial f(\boldsymbol{x}_i)}{\partial x_{i,p}}. \tag{2.13}$$

*Proof.* Equation (2.12) follows directly from definition 2.5, 2.6 and 2.7. Equation (2.13) follows directly from this and proposition 2.1. □

This provides the object of main interest: an underlying function $f(\boldsymbol{x}_i)$ which relates covariates to outcomes. It also makes quite clear the assumptions needed (or at least sufficient) to ensure that such a function is well-defined, and provides causal effects rather than mere prediction.

### 2.3.2. A simple framework for evaluating approximate estimators

Estimation is straightforward if the functional form $f(\cdot)$ is known. In reality, functional forms are rarely known with certainty, but this does not necessarily mean they are unwise. I will show this by defining a framework for judging estimators. My view of econometrics is inspired by Angrist & Pischke (2008)[16]

> : '[O]ur view of of empirical work [is] an effort to describe the essential features of statistical relationships without necessarily trying to pin them down exactly'.

The authors use this to argue that OLS may be good enough for analysis, even if the underlying process is non-linear, because it provides the best linear approximation:

[16] Some economists disagree. For a comparison in favour of a *structural* approach, see Heckman (2010).

**Proposition 2.5.** The function $\boldsymbol{x}_i \boldsymbol{\beta}$ with $\boldsymbol{\beta} = \mathrm{E}[\boldsymbol{x}_i \boldsymbol{x}_i']^{-1} \mathrm{E}[\boldsymbol{x}_i y_i]$ provides the best linear approximation to $\mathrm{E}[y_i | \boldsymbol{x}_i]$ in a minimum mean square error sense:

$$\boldsymbol{\beta} = \arg\min_{\boldsymbol{b}} \mathrm{E}\left[ (\mathrm{E}[y_i | \boldsymbol{x}_i] - \boldsymbol{x}_i \boldsymbol{b})^2 \right]. \tag{2.14}$$

*Proof.* A proof is provided in appendix V.4, inspired by Angrist & Pischke (2008). □

Even though I disagree linear methods are acceptable, this inspired me to define a general framework for evaluating approximate estimators. MSE provides a good error measure because it is fairly intuitive, mathematically convenient, and implies consistency if it goes to zero (more on this in section 5.1). I define estimators in a *non-parametric*, *frequentist* manner: Assume there exists a true value $\theta(\boldsymbol{x}_i)$, which we estimate using an estimator $\hat{\theta}(\boldsymbol{x}_i)$, a random variable hopefully somewhat related to the true value. $\theta(\boldsymbol{x}_i)$ directly reflects some effect we are interested in (either the expectation $f(\boldsymbol{x}_i)$ or the marginal effect $\gamma_p(\boldsymbol{x}_i)$)), rather than a parameter in the DGP. This is in the non-parametric spirit, but the approach can also judge parametric estimators - I just skip to the part where effects are calculated from parameters.

**Definition 2.8.** The *estimation objective* for an estimator $\hat{\theta}(\boldsymbol{x}_i)$ of a true value function $\theta(\boldsymbol{x}_i)$ is to obtain the best estimator in a minimum mean square error sense:

$$\hat{\theta}^{OPT}(\boldsymbol{x}_i) = \arg\min_{\hat{\theta}(\boldsymbol{x}_i)} \mathrm{E}\left[ \left( \theta(\boldsymbol{x}_i) - \hat{\theta}(\boldsymbol{x}_i) \right)^2 \right]. \tag{2.15}$$

This provides a meaningful guideline both asymptotically and for finite samples. Asymptotically, we are interested in consistent estimators, where the error goes to zero. However, in finite samples (also called 'reality') we are interested in the estimators which perform best for the given data $\boldsymbol{x}_i$, which may not be the asymptotically optimal estimators. One may see this by decomposing the objective into two error sources. First, I will account for the fact that we do not know the true DGP, by distinguishing between the true value function $\theta(\boldsymbol{x}_i)$, and the statistical model we construct for it $\tilde{\theta}(\boldsymbol{x}_i)$, which is the quantity we are actually estimating.

**Definition 2.9.** The *approximation error* of statistical model $\tilde{\theta}(\boldsymbol{x}_i)$ vis a vis the true model $\theta(\boldsymbol{x}_i)$ for a given $\boldsymbol{x}_i$ is $\epsilon_A(\boldsymbol{x}_i) = \theta(\boldsymbol{x}_i) - \tilde{\theta}(\boldsymbol{x}_i)$.

One may claim to have found the true model $\tilde{\theta}(\boldsymbol{x}_i) = \theta(\boldsymbol{x}_i)$, as old school statistics often do, which obviously allows zero error $\epsilon_A = 0$. However, as the quote above indicates, modern econometricians are often honest about that the fact that their statistical model has some $\epsilon_A(\boldsymbol{x}_I) \neq 0$. One part of the estimator puzzle is then to find an estimator with low enough approximation error to be useful. The second part reflects that statistics is uncertain, and we cannot obtain optimal models within finite samples:

**Definition 2.10.** The *estimation error* of an estimate $\hat{\theta}(\boldsymbol{x}_i)$ vis a vis the optimal value of the statistical model it estimates $\tilde{\theta}(\boldsymbol{x}_i)$ for a given $\boldsymbol{x}_i$ is $\epsilon_E(\boldsymbol{x}_i) = \tilde{\theta}(\boldsymbol{x}_i) - \hat{\theta}(\boldsymbol{x}_i)$.

Given a consistent estimator, this error should go to zero asymptotically (I return to this in section 5.1), but in finite samples it should not be ignored. The relation between these two provides an overall guide for choosing estimators:

**Lemma 2.1.** The MSE of an estimate $\hat{\theta}(\boldsymbol{x}_i)$ vis a vis the true model $\theta(\boldsymbol{x}_i)$ can be summarized

$$\mathrm{E}\left[\left(f(\boldsymbol{x}_i) - \hat{f}(\boldsymbol{x}_i)\right)^2\right] = E\left[\left(\epsilon_A(\boldsymbol{x}_i) + \epsilon_E(\boldsymbol{x}_i)\right)^2\right]. \tag{2.16}$$

*Proof.* $\mathrm{E}\left[\left(\theta(\boldsymbol{x}_i) - \hat{\theta}(\boldsymbol{x}_i)\right)^2\right] = \mathrm{E}\left[\left((\theta(\boldsymbol{x}_i) - \tilde{\theta}(\boldsymbol{x}_i)) + (\tilde{\theta}(\boldsymbol{x}_i) - \hat{\theta}(\boldsymbol{x}_i))\right)^2\right]$. Insert definition 2.9 and 2.10. □

This provides a simple framework which explicitly accounts for the fact that we can make errors both through estimation and model choice, and weighs all errors equally. Fully parametric estimators may perform well if assumptions hold (maximum likelihood for instance will have zero approximation error and minimum estimation error), but approximation error may be high if they do not, which is likely if the true DGP is uncertain (as it typically is). Fully non-parametric estimators (such as kernel regression and other local estimators) may allow no approximation error under mild assumptions, but will typically exhibit large estimation error (particularly in high dimensions). It therefore provides formal justification for choosing *approximate* estimators, which can achieve low approximation error even if they can never fully recover the DGP, and which impose enough structure to limit estimation error.

**Proposition 2.6.** The *optimal estimator* for an estimation objective following definition 2.8 minimizes the expected sum of approximation and estimation error given available data.

*Proof.* This follows directly from definition 2.8 and lemma 2.1. □

This finalizes the theoretical framework for the thesis. The next section (finally) introduces and motivates neural networks as an estimator which may achieve this estimation goal, given the underlying assumptions for causal inference from the previous section.

### 2.3.3. Join the data side, we've got universal approximators

Neural networks may be a good candidate for an estimator which satisfies the goals outlined above. The next section presents neural networks as an estimator. Here, I will motivate their use. The main theoretical justification for neural networks is that they are *universal approximators*, which are capable of replicating any underlying function as precisely as one would like. This may offer significant improvements compared to parametric models if the underlying process exhibits non-linearity (of unknown form).

**Theorem 2.1** (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 3.1, which is explained below)[17] with output $\tilde{f}(\boldsymbol{x}_i)$ such that

$$\forall\, \boldsymbol{x}_i \in \mathcal{X}: \qquad \left|\tilde{f}(\boldsymbol{x}_i) - f(\boldsymbol{x}_i)\right| < \epsilon. \tag{2.17}$$

*Proof.* Versions of the theorem were separately obtained for various narrowly defined 'squashing' activation functions (see section 3.2.1) by Hornik *et al.* (1989), Cybenko (1989) and Funahashi (1989). The allowed functions were then extended by Hornik (1991) and Leshno *et al.* (1993). □

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

This is a powerful result. Given that we make the neural network itself complex enough, we can approximate continuous functions arbitrarily well. This implies, in the framework above, that we can achieve an arbitrarily low approximation error $\epsilon_A(\boldsymbol{x}_i)$ (which is exactly what is shown in the theorem). The result is easily extendable to the binary case.

**Corollary 2.1.** For any continuous $f(\boldsymbol{x}_i) \in [0,1]$ on a compact subset $\mathcal{X} \subseteq \mathbb{R}^k$, and any error $\epsilon > 0$, there exists a neural network with one hidden layer[18] with output $\tilde{f}(\boldsymbol{x}_i)$ such that:

$$\forall\, \boldsymbol{x}_i \in \mathcal{X} : \qquad \left|\tilde{f}(\boldsymbol{x}_i) - f(\boldsymbol{x}_i)\right| < \epsilon. \tag{2.18}$$

*Proof.* Castro *et al.* (2000) prove the corollary from the results of Hornik *et al.* (1989) and Funahashi (1989). They use that with a strictly increasing, continous $\sigma$, there exists a continuous funtion $\sigma^{-1}\left(f(\boldsymbol{x})\right) \in \mathbb{R}$ which can be approximated just as in theorem 2.1. Hence, their proof also applies to the more general activation functions. □

But wait! One might object that we are not only interested in $f(\boldsymbol{x}_i) = \mathrm{E}[y_i|\, \boldsymbol{x}_i]$. Our purpose is estimation of marginal effects rather than prediction after all. Fortunately, the theoreticians have got us covered. Hornik *et al.* (1990) showed that neural networks are capable of simultaneously providing an arbitrarily close approximation (in the sense of the theorems above) to an arbitrary function *and its derivatives*. This provides a justification for their use for our purposes, something the authors were quite aware of:

> '[A] potential application area is economics, where theoretical considerations lead to hypotheses about the derivative properties ... of certain functions. ... Our results establish neural network models as providing an alternative framework for studying the theory of the firm and of the consumer'. (Hornik *et al.* (1990))

The results implies that we can use neural networks to model arbitrary non-linearity as precisely as we'd like. However, recall that approximation error is only part of the picture: We also have to estimate the model. White (1990)[19] showed 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 of estimates of the marginal effects as well. These results are important because they establish the possibility of consistent learning. However, modern neural networks are not trained using sieve estimation. Below, I will show that (with-in my simulation framework), neural networks still provide consistent estimates of the marginal effects. I also show that neural networks converge fast enough to be useful with data sets of realistic size, even with several regressors (where classical non-parametric estimators, such as kernel regression, often succumb to the curse of dimensionality). First though, I will explain how neural networks manage these extraordinary approximation capabilities, and how one works with them in practice.

[18] Formally, for a strictly increasing, continuous *squashing* output function $\sigma : \mathbb{R} \to [0,1]$ with $\lim_{s\to\infty} \sigma(s) = 1$ and $\lim_{s\to-\infty} \sigma(s) = 0$ and activation function $\eta(\cdot)$ as in theorem 2.1, there exists $J \in \mathbb{N}$, $\boldsymbol{\beta}^{(1)} \in \mathbb{R}^{(k+1)\times J}$ and $\boldsymbol{\beta}^{(2)} \in \mathbb{R}^{(J+1)\times 1}$ such that the result holds.

[19] Fun fact: This is the same Halbert White who came up with robust standard errors (White, 1980). He also participated in showing the universal approximation theorem along with Kurt Hornik & Maxwell Stinchcombe.

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

'Any sufficiently advanced technology is indistinguishable from magic'.

*– Arthur C. Clarke (1973)*

Neural networks may seem daunting at first, but econometricians will find plenty familiar features within them. In short, they combine a number of different underlying 'regression-like' models in an automated manner, which is found using a maximum likelihood framework. The approximation capacity is determined by the number of underlying models. The remainder of this section shows how this work. First, I illustrate the structure of networks and their capacity rather informally, which should hopefully motivate their use even further. Second, I present the framework formally for the basic version of a neural network, with a focus on elements relevant for econometricians. I also present the main pitfalls one should be aware of when designing and training networks. However, I argue that some of the more technical aspects (such as efficient gradient calculation) can reliably be left to the computer scientists. The final section extents the formal framework to 'deep' models. This is not critical for disinterested readers, as such models function much like the simpler case, but does provide the final estimator of marginal effects.

### 3.1. Neurons for dummies: An informal introduction

Neural networks are are often motivated by a loose inspiration from neuroscience, but it may be more fruitful to think of them solely as a statistical technique, or 'efficient nonlinear function approximators' (Goodfellow *et al.*, 2016, 217). It is true that the earliest networks intended to model biological learning by replicating features of the brain, but current networks are neither a very good description of the brain, nor intended to be one. Even though the basic idea of multi-layered learning through firing neurons remains, modern networks are more a product of mathematics and engineering than biology. In this sense, networks are inspired by the brain much in the same way that submarines could be inspired by sharks.[20] All in all,

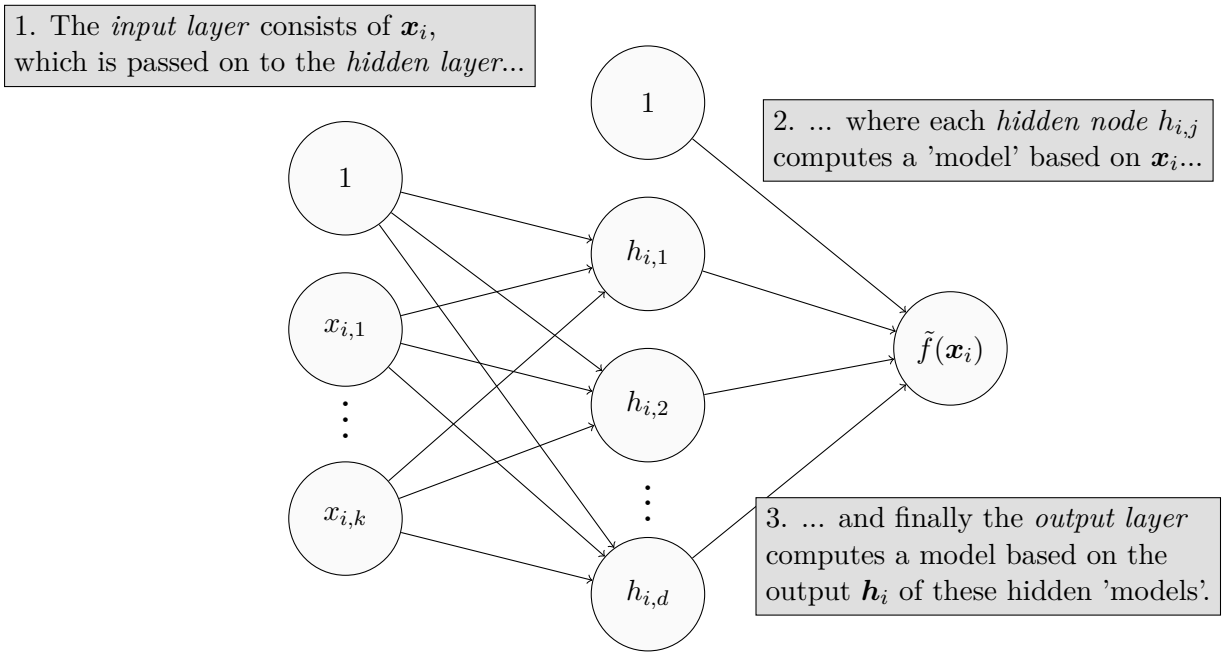
> It is best to think of feedforward networks as function approximation machines that are designed to achieve statistical generalization, occasionally drawing some insights from what we know about the brain, rather than as models of brain function. (Goodfellow *et al.* (2016), p. 164)

If they do not work like the brain, how do they work? Figure 3.1 illustrates a basic example of a neural network. I will go through it step by step. The basic set-up is that we divide the various operations of the network into different *nodes* (or neurons), which are in turn sorted into different *layers*. The basic neural network has three layers - an *input layer*, and *output layer* and a *hidden layer* in between. Later, I will introduce 'deep' networks with multiple hidden layers, but there is no reason to start there. The key feature is the existence of hidden layers.

The easiest way to grasp the concept of layers may be to note that any basic statistical model can be described as a two-layer model with an input and output layer. Consider, for instance,

[20] I do not, in fact, know whether submarines were inspired by sharks. However, they do replicate key features of sharks (such as being underwater predators, and terrifying to be inside), and the lingo overlaps (the cylindrical structure on top of submarines is called a 'fin'). Despite this, they are quite poor models of sharks all in all.

**Figure 3.1:** *A neural network with a single hidden layer*




a *logistic regression*. Setting aside optimization issues for now, a logistic regression computes a linear signal, and squashes it down to a probability using the sigmoid function:[21]

$$f(\boldsymbol{x}_i \,|\, \boldsymbol{\beta}) = \sigma\left(\beta_0 + \beta_1 x_{i,1} + ... + \beta_k x_{i,k}\right), \qquad \sigma(s) = \frac{1}{1+e^{-s}}. \tag{3.1}$$

This computation takes an input layer consisting of $\boldsymbol{x}_i = \begin{pmatrix} 1 & x_{i,1} & ... & x_{i,k} \end{pmatrix}$ and passes it onto an output layer in a *linear signal* $s_i = \beta_0 + \beta_1 x_{i,1} + ... + \beta_k x_{i,k}$, to which the final node then applies an *output function* $f(\boldsymbol{x}_i) = \sigma(s_i)$. I could illustrate this by omitting the the middle layer in figure 3.1. There is typically no reason to formulate this as two layers, but nor is there a reason one cannot. The layered representation does however ease the understanding of neural networks, because more happens when we add the hidden layer. We may think of it as adding more statistical 'models',[22] which the networks then combines together in a clever manner. In one version of the neural network, each hidden node actually applies the same computation as the logistic regression. However, the nodes do not perform individual logistic regression (if they did, they would likely arrive at the same answer). Instead, the models are designed to work together, so we achieve an optimal prediction when the final layer combines each of the hidden 'models' in a aggregate model, providing a single prediction based on them all.

An example may help illustrate how this works[23]. The left-most picture in figure 3.2 shows a simple, deterministic target function we might wish to approximate. The underlying function is a 'circle function', where observations inside the circle are positives, and observations outside are negative.[24] I could try to solve this using a logistic regression. However, logistic regression in

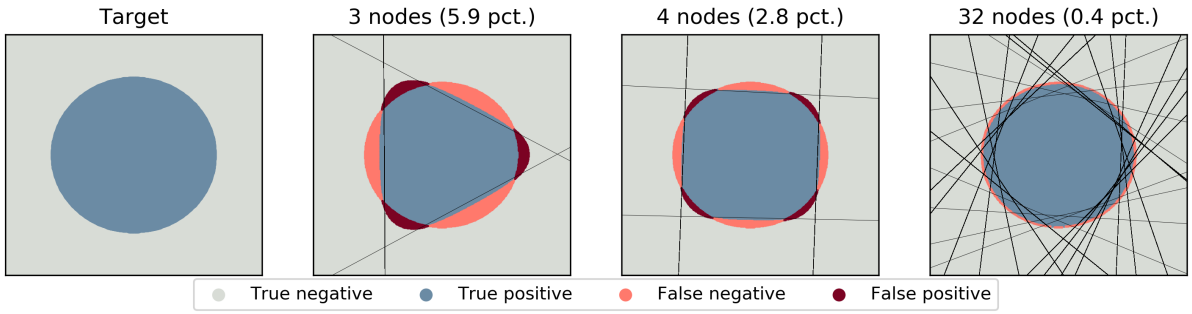
[21] Appendix I provides a general summary of the notation used in the paper, including variable dimensions.

[22] It is not strictly speaking individual models, because no individual optimization takes place. Rather, we solve the entire network for the optimal structure. However, the slight abuse of notation may aid understanding.

[23] This example largely follows one in Abu-Mostafa *et al.* (2012b), but directly uses my estimation set-up.

[24] Formally, $y_i = 1\{x_1^2 + x_2^2 < r^2\}$. I observe an entire 'population' $x_1, x_2 \in [-5, 5]$, and a radius of 2.

**Figure 3.2:** *Alas, witness the approximation capacity of the mighty neural network!*




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

two dimensions basically amounts to drawing a separating line, which cannot separate the circle properly.[25] Instead, I may apply a neural network. The remaining pictures does exactly that, using my estimation set-up defined below. Each of them implements a number of hidden nodes, which computes a 'logistic regression'. Since one logistic regression allows one linear separator, multiple models allow more.

The network implemented in the second picture has three hidden nodes, and hence allows three lines. The final prediction can then be based on the points placement vis a vis all three lines. This allows us to draw a triangle, and predict positively within it. A triangle cannot replicate the circle perfectly, but it does much better than a single line. Some points in the corners are inside the triangle although outside the circle (false positives), and some on the sides are outside the triangle even though the belong in the circle (false negatives). Despite this, the model can predict points with an error rate of 5.9 percent (compared to a baseline error of 28 pct. if I just guessed at the class there are most examples of).

The next pictures illustrate how the representational capacity of a neural network can be increased by adding more nodes to the hidden layer. If I add one more I can build a box instead of a triangle, decreasing the error even further. Adding more nodes (and thus lines) allows the network to draw more complex shapes. With the 32 nodes in the last picture, the lines can actually draw a pretty good circle. By adding more lines, I could decrease approximation error arbitrarily much, following the theorems above.

The example was for a binary classifier, but the intuition is much the same for a regression problem. For a continuous problem, the target will be some sort of line (or hyperplane in more dimensions). If it is non-linear, it may exhibit various quirks which a straight line cannot replicate. Instead, a neural network will construct a series of hidden 'models', each of which may be though of as providing a single line, and then combine these to approximate the target line. Again, the network 'figures out' when and how to use each of these provided lines. Therefore, given arbitrarily many lines, an infinitely close approximation can be formed to any target, no matter how quirky. Next, I will show how this is formally done.

[25] Strictly speaking, logistic regression computes a probability. However, implementing a classifier which predicts the high-probability outcome $1\{f(\boldsymbol{x}_i \mid \boldsymbol{\beta}) \geq 0.5\}$ corresponds to predicting one if the linear signal is positive, and zero if negative. This corresponds exactly to predicting differently on either side of a line.

## 3.2. Neurons for economists: A formal introduction

The following section goes through the workings of networks in a bit more detail.[26] First, I provide a formal framework for neural networks with a single hidden layer, and derive the marginal effects of regressors within them. Then, I describe the optimization procedure used for estimation of networks, with a particular focus on topics of relevance to econometric practitioners.

### 3.2.1. Forward pass: How does a network arrive at its conclusions?

Consider again a neural network as in figure 3.1. This is a *feed-forward* neural network, so named because information only flows forward with no feedback connections (Goodfellow *et al.*, 2016). I focus on these, which are both strong in and of themselves, and provide the building blocks for more complex networks.[27] If marginal effects are well-behaved in the basic networks, research could investigate more complex methods. I will go through the operations of a single-layer network step by step, but the terminology scales naturally to more hidden layers (see section 3.3).[28] For now, the network consists of an input layer, a hidden layer, and an output layer. The input layer merely passes the exogenous variables to the hidden layer, so I will start from there.

**Single hidden node**. Recall that each hidden node is intended to compute a small 'model'. These models are actually simple linear models, to which we apply a non-linear transformation. Each node computes a linear signal based on the $1 \times (k+1)$ regressors and intercept $\boldsymbol{x}_i$:

$$s_{i,j} = \beta_{0,j}^{(1)} + \beta_{j,1}^{(1)} x_{i,1} + ... + \beta_{j,k}^{(1)} x_{i,k} = \boldsymbol{x}_i \, \boldsymbol{\beta}_j^{(1)} \, . \tag{3.2}$$

The $(k+1) \times 1$ coefficient vector $\boldsymbol{\beta}_j^{(1)}$ is specific to the hidden node, so that each node can compute its own little 'model'. It is denoted by $j$ to specify the node, and by (1) to distinguish it from the output layer. Each $\beta_{j,p}$ corresponds to one of the lines in the figure above, as it connects a node in the input layer $x_{i,p}$ to the hidden node $j$. The node then applies a non-linear *activation* function to this signal. This step is crucial. Without it, we would simply be combining linear models, which could be combined to a single linear model (i.e. OLS/logit). The activation functions (inspired by the 'firing' of neurons) is what provides the approximation capacity:

$$h_{i,j} = \eta(s_{i,j}). \tag{3.3}$$

The form of $\eta(\cdot)$ is not set in stone, and may be case dependent. Above, I used the sigmoid function, which worked well for the circle example. While this is still relatively common, the default for modern networks is the *rectified linear unit* (ReLU) function, $\eta(s) = \max\{0, s\}$. This works well because linear units are easy to optimize, but it is still non-linear enough for approximation because it only 'fires' some times (Goodfellow *et al.*, 2016, p. 187).

**All hidden units**. The steps are performed for all hidden units. The output of each of these non-linearly transformed signals is combined in a vector, which is then passed on the output layer. One way to think of this is as *feature learning*, where the algorithm automatically learns

[26] For good introductions to neural networks, see Abu-Mostafa *et al.* (2012b) or Goodfellow *et al.* (2016).

[27] The two main such are *convolutional* networks, which have performed well for e.g. image recognition and time series, and *recurrent* networks, which specialize in sequential data, such as e.g. speech recognition.

[28] The notation used largely follows Abu-Mostafa *et al.* (2012b), although there are minor adjustments

an alternative version of the regressors (much in the samme manner as you might manually add squared terms to a regression). In this sense, the hidden layer just creates a new set of variables. Once again adding a constant node, this leads to a $1 \times (J+1)$ hidden output vector $\boldsymbol{h}_i(\boldsymbol{x}_i)$.

**Output layer.** The final node computes a linear signal from this, and applies an *output function*. The purpose of this function is just to ensure the output belongs to the desired space. If the problem is binary, we use the sigmoid function $\sigma(s) = 1/(1+e^{-s})$ to obtain a probability. If $y_i$ is continuous, we just use *linear output* $\sigma(s) = s$, where we basically do nothing:

$$\tilde{f}(\boldsymbol{x}_i \,|\, \boldsymbol{\beta}) = \sigma\left(\beta_0^{(2)} + \beta_1^{(2)} h_{i,1} + ... + \beta_J^{(2)} h_{i,J}\right) = \sigma\left(\boldsymbol{h}_i \, \boldsymbol{\beta}^{(2)}\right). \tag{3.4}$$

The $(J+1) \times 1$ coefficient vector $\boldsymbol{\beta}^{(2)}$ is specific to the output layer. This calculation produces a single estimate for the underlying function (i.e. the conditional expectation in our set-up).

**Definition 3.1.** A *neural network* with a *single* hidden layer with $J$ nodes, output function $\sigma(\cdot)$, activation function $\eta(\cdot)$, $1 \times (k+1)$ feature and intercept vector $\boldsymbol{x}_i$, $(k+1) \times J$ coefficient matrix $\boldsymbol{\beta}^{(1)}$ for the hidden layer, and $(J+1) \times 1$ coefficient vector $\boldsymbol{\beta}^{(2)}$ for the output layer estimates

$$\begin{aligned} \tilde{f}(\boldsymbol{x}_i \,|\, \boldsymbol{\beta}) &= \sigma\left(\beta_0^{(2)} + \sum_{j=1}^{J} \beta_j^{(2)} \eta\left(\beta_{0,j}^{(1)} + \sum_{p=1}^{k} \beta_{j,p}^{(1)} x_{i,k}\right)\right) \\ &= \sigma\left(\tilde{\eta}\left(\boldsymbol{x}_i \, \boldsymbol{\beta}^{(1)}\right) \boldsymbol{\beta}^{(2)}\right), \end{aligned} \tag{3.5}$$

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

A nice feature of neural networks is that the entire representation is continuous, and may even be written in elegantly condensed matrix notation, as we all know econometricians love to do. Continuity allows us to derive the marginal effects directly from this expression. This is well-known in the texts on machine learning, but the expressions are rarely used for analytical purposes.[29] Derivation is done through successive application of the chain rule, since the network implements a sequence of functions on the linear signals of the previous layer.

**Proposition 3.1.** The *marginal effect* of a regressor (feature) $x_p$ evaluated at $\boldsymbol{x}_i$, in a neural network with a single hidden layer (as in definition 3.1) is the partial derivative

$$\begin{aligned} \gamma_p(\boldsymbol{x}_i) = \frac{\partial \tilde{f}(\boldsymbol{x}_i)}{\partial x_{i,p}} &= \sigma'\left(s_i^{(2)}\right)\left(\sum_{j=1}^{J} \beta_j^{(2)} \beta_{j,p}^{(1)} \eta'\left(s_{i,j}^{(1)}\right)\right) \\ &= \sigma'\left(s_i^{(2)}\right)\left(\eta'\left(\boldsymbol{s}_i^{(1)}\right) \operatorname{diag}\left(\boldsymbol{\beta}_p^{(1)}\right) \boldsymbol{\beta}_{1:}^{(2)}\right), \end{aligned} \tag{3.6}$$

with $\sigma'(s) = \partial\sigma(s)/\partial s$, $\eta'(s) = \partial\eta(s)/\partial s$, $\boldsymbol{s}_i^{(1)} = \boldsymbol{x}_i \, \boldsymbol{\beta}^{(1)}$, $s_i^{(2)} = \eta(s_i^{(1)}) \, \boldsymbol{\beta}^{(2)}$, $\operatorname{diag}\left(\boldsymbol{\beta}_p^{(1)}\right)$ as a $J \times J$ diagonal matrix based on row $p$ of $\boldsymbol{\beta}^{(1)}$, and $\boldsymbol{\beta}_{1:}^{(2)}$ is $\boldsymbol{\beta}^{(2)}$ excluding intercept $\beta_0^{(2)}$.

*Proof.* The expression is derived in appendix V.5. □

The key feature of neural networks is that changes in input variables do not affect the outcome directly - rather, they affect each of the hidden nodes, which in turn will affect the outcome.

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

This does not mean that marginal effects are not well-defined. Recall that the hidden nodes basically perform feature engineering, finding features which approximate the function. As a comparison, when we add quadratic terms to a regression, marginal effects remain well-defined but must account for these terms. In a similar sense, the above expression capture how changes in $x_{i,p}$ flow through the nodes. This also does away with assigning meaning to each node, which is typically difficult. Focus is solely on the overall effect of changes in $x_{i,p}$, which is enough for the econometric goal of causal estimation.

#### 3.2.2. Backwards pass: How do we find optimal neural networks?

Estimation of the optimal parameters is based on numerical optimization of an objective function, a concept familiar to econometricians[30]. Indeed, the typical objective functions are inspired by the principle of maximum likelihood, although for the approximation provided by the neural network rather than an assumed true model.[31] Let $\hat{\boldsymbol{\beta}}$ denote the entire set of parameters used in a network of the form in definition 3.1, and let $\hat{f}(\boldsymbol{x}_i\,|\hat{\boldsymbol{\beta}})$ be the corresponding prediction. The goal is to minimize these population objectives (sample analogs are provided in appendix III.1):

$$\begin{aligned} L_C(\hat{\boldsymbol{\beta}}|\,\boldsymbol{y},\boldsymbol{x}) &= -\,\mathrm{E}\left[-y_i \ln \hat{f}\left(\boldsymbol{x}_i\,|\hat{\boldsymbol{\beta}}\right) - (1-y_i)\ln\left(1-\hat{f}\left(\boldsymbol{x}_i\,|\hat{\boldsymbol{\beta}}\right)\right)\right], \\ L_R(\hat{\boldsymbol{\beta}}|\,\boldsymbol{y},\boldsymbol{x}) &= \frac{1}{2}\,\mathrm{E}\left[\left(\hat{f}\left(\boldsymbol{x}_i\,|\hat{\boldsymbol{\beta}}\right)-y_i\right)^2\right]. \end{aligned} \tag{3.7}$$

The objective for classification $L_C$ corresponds exactly to maximum likelihood based on a Bernoulli distribution. The continuous case $L_R$ similarly corresponds to regression with Gaussian noise (see appendix III.2 for derivations). The basic approach to optimizing this is also familiar, since it employs versions of gradient descent.[32] However, computing the entire gradient from scratch is computationally expensive due to the complexity of networks, and the number of parameters. Instead neural networks rely on the *back-propagation algorithm* (Rumelhart *et al.*, 1986), which takes advantage of the chain-like structure of neural networks.[33] The basic idea is to first compute the derivative of the final layer, and then 'propagate' these backwards. The chain rule implies that these derivatives summarize all relevant information for the hidden layer. The same holds for any given layer in the deeper models. This allows us to only calculate relevant information once, providing an efficient calculation of the gradient. The literature on how to do this quickly is quite technical, and I would argue that it is not necessary for practitioners to understand all details. However, it is important to be somewhat more vigilant than one may be used to, since estimation is more likely to fail for neural networks than for many traditional econometric estimators (see below). Practioneers should know how to watch out for pitfalls, and implement basic alterations to the networks to account for specific troubles.

**Optimization failure**[34]. Econometricians are often used to minimizing convex loss functions with analytical gradients, ensuring easy convergence to a global minimum. Neural networks are

[30] For thorough discussion of estimation goals for neural networks, see Goodfellow *et al.* (2016)

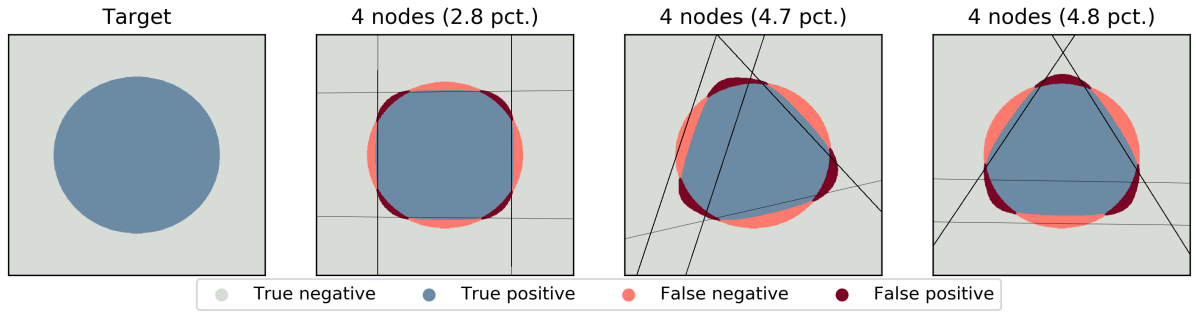
[31] Data scientists may refer to this as '*cross-entropy* between the training data and the model's predictions' (Goodfellow *et al.*, 2016, 172), but they really just mean minimizing the log-likelihood.

[32] Gradient descent is an iterative optimization approach which updates the weights based on the current gradient, until a local extremum is found. For a general overview, see Cameron & Trivedi (2005).

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

[34] For a thorough discussion of numerical optimization of neural networks, see ch. 8 in Goodfellow *et al.* (2016)

**Figure 3.3:** *Examples of convergence failure.*




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

less fortunate. They are typically non-convex, implying a danger of ending in local minima. The algorithmic approach to gradient updating may also spell trouble, as gradients can be inexact, saturated, exploding, or likely exhibit one of a myriad other problems, which may hinder convergence. Figure 3.3 shows an example of convergence failure by revisiting the circle target example. The same network is refitted with different starting values, leading to optimization failure in the last two cases (where a triangle is found instead of the optimal box). An econometrician might plausibly suggest trying a range of starting values to ensure one has found a (likely) global optimum (Cameron & Trivedi, 2005). Data scientists often highlight that reaching a local minimum with low cost may be 'good enough', and that most local minima satisfy this (Goodfellow *et al.*, 2016, 178). My results, where I could not pay close attention to convergence in each simulation, indicates that networks typically provide decent solutions, also for marginal effects. However, in practice, econometricians venturing into learning should carefully consider whether optimization appears to have been successful.

**Overfitting**.[35] The goal of modelling is to reflect the underlying process. However, with methods such as neural networks which are capable of approximating complex functions, there is always a danger of *overfitting*, where algorithms learns to replicate the noise in observed data, rather than the underlying process. Overfitting is also likely to compromise our estimates of marginal effects - If our model replicates noise, it is likely this also affects observed marginal effects. Fortunately, strong tools have been developed to combat overfitting. Two main ones deserve mention. First, the literature on machine learning focuses on the partition of tasks into *training*, *validation* and *testing* of algorithms, which stress that algorithms should be tested on data it has not seen before.[36] If the neural net approximates the true function, it should work about as well on this data as on the data it was trained on. In practice, there will typically be some divergence, as it is hard not to replicate a little noise. If there is a large divergence, the practioneer should regard the results with healthy scepticism. Fortunately there is a large literature on how to avoid this by employing *regularization*, which is defined as 'any modification we make to a learning algorithm that is intended to reduce its generalization error but not its training error' (Goodfellow *et al.*, 2016, 117). A typical approach is to employ versions of

[35]For a good introduction to overfitting, see Abu-Mostafa *et al.* (2012a). For a thorough discussion in the context of neural networks, see Goodfellow *et al.* (2016)

[36]There are strong theoretical results underlying this result, e.g. VC dimensions (Abu-Mostafa *et al.*, 2012a).

*weight decay*, which discourages algorithms from fitting too closely to the data by adding a term $\Omega(\boldsymbol{\beta})$ to the objetive function, which punishes large coefficients.[37] Some econometricians are sceptical about the use of regularization because it 'encourages the choice of less complex, but wrong models' (Mullainathan & Spiess, 2017). However, note that neural networks always fit approximations. The key argument should be whether regularization gets us closer to the target. Since it discourages overfitting (which is also a wrong model), this seems likely. Although more research on its effects is probably wise, I suggest employing it for now.

**Network architecture**. A final, but crucial, issue is the choice of which network to estimate. The theoretical results state that a network exists with arbitrarily close approximation, but does not say which it is. The network architecture (number of nodes and hidden layers) determines the model's *effective capacity*, the level of functional complexity it can replicate. There are two key concerns which should guide the choice of architecture. The first is the complexity of the question at hand. If the function is believed to be complex, we would ideally like a large network to reflect this. However, if the problem is simple, a complex network is likely to overfit. The second concern is the data at hand. If data is limited, it may be unlikely that we can model the full complexity with any degree of certainty, and a simpler approximation may do better. This implies that model complexity should match 'the quantity and quality of the data we have' (Abu-Mostafa *et al.*, 2012a, 122). A pragmatic approach to figuring out what the data can sustain is to use a *grid search* to evaluate networks of various size, and then choose the one which performs best in a validation approach (Goodfellow *et al.*, 2016).

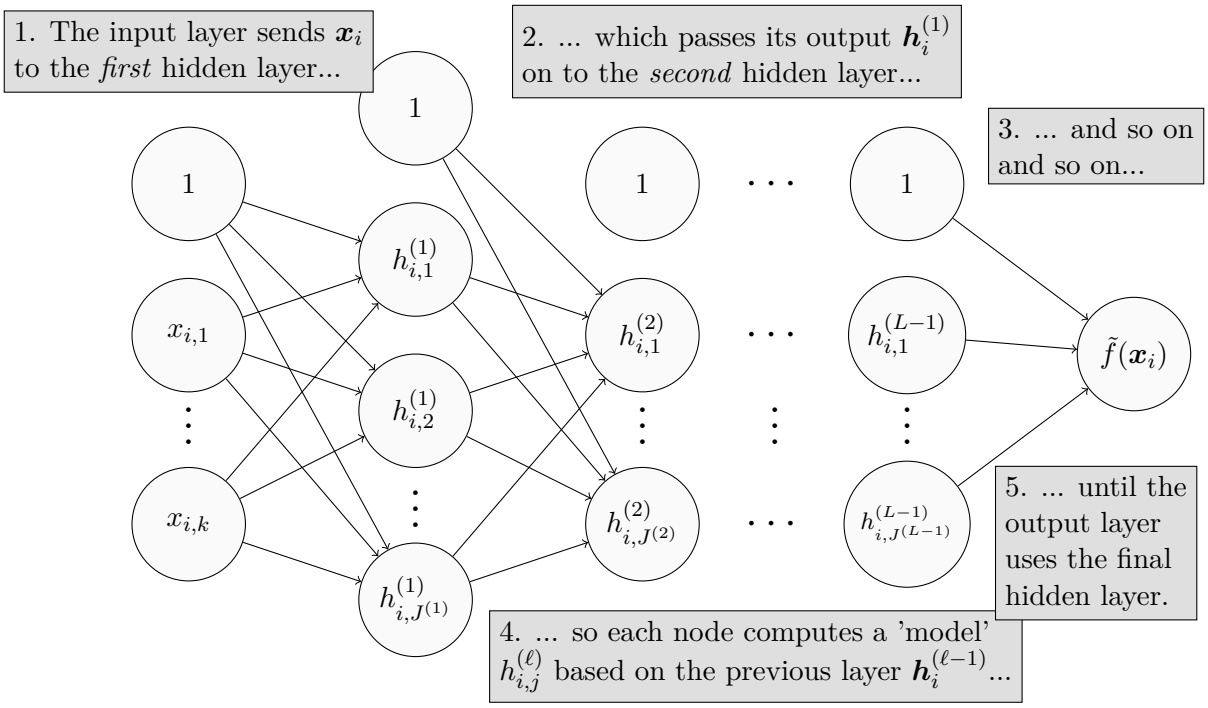
### 3.3. Neurons for nerds: Extension to deep learning

The success of neural networks has been driven by *deep learning*, where adding multiple hidden layers allows learning of 'representations of data with multiple levels of abstraction' (LeCun *et al.*, 2015). There are two ways to motivative deep learning. One is that it may provide a more natural approach to learning, which corresponds to how humans learn. A second, more pragmatic approach is that it may simply allow the algorithm to learn the best approximation more efficiently. Whatever your preference, the key takeaway is that although a network with one layer can reach an arbitrarily level of precision, a much smaller network with multiple layers may exist which reaches the same precision (Abu-Mostafa *et al.*, 2012b). Training this network may be more efficient and thus allows us to reduce our estimation error without sacrificing approximation error. This section retraces the steps of the first section, once again describing a full network. However, now I do so with an arbitrary number of hidden layers.

**Network set-up**.[38] A network with layers $\ell = 0, 1, 2..., L$ is illustrated in figure 3.1. The input layer is denoted by $\ell = 0$, and the output layer is $\ell = L$. In between, we have $L-1$ hidden layers, each with $J^{(\ell)} + 1$ hidden nodes. The basic mechanics of the model are unchanged: Each layer passes its output onto the next layer, where each node computes a linear signal based on these, and applies a non-linear function. The hidden layers now take their input from the previous hidden layers, and the output layer observes the last hidden layer.

[37] More advanced forms of regularization also exists, such as *dropout*. See ch. 7 in (Goodfellow *et al.*, 2016).
[38] Once again the notation largely follows Abu-Mostafa *et al.* (2012b), but there are some adjustments.

**Figure 3.4:** *A deep neural network with multiple hidden layers.*




**Hidden layer**. Each hidden layer $0 < \ell < L$ observes the output of the previous layer, $\boldsymbol{h}_i^{(\ell-1)}$. This is easiest to see by starting from the first hidden layer. The output of the input layer is simply the regressors, so define $\boldsymbol{h}_i^{(0)} = \boldsymbol{x}_i = \begin{pmatrix} 1 & x_{i,1} & \dots & x_{i,k} \end{pmatrix}$. A given node $j$ in the first layer computes a linear signal $s_{i,k}^{(1)} = \boldsymbol{x}_i \boldsymbol{\beta}_j^{(1)} = \boldsymbol{h}_i^{(0)} \boldsymbol{\beta}_j^{(1)}$, and then applies an activation function to produce its output $h_{i,j}^{(1)} = \eta\left(s_{i,k}^{(1)}\right)$. The layer passes a vector of these outputs (and a constant term) $\boldsymbol{h}_i^{(1)}$ to the next hidden layer. This layer then perform the exact same steps. We can write this neatly in matrix notation, for an arbitrary layer. Each will compute a linear signal:

$$\boldsymbol{s}_i^{(\ell)} = \boldsymbol{h}_i^{(\ell-1)} \boldsymbol{\beta}^{(\ell)}, \qquad 0 < \ell \leq L. \tag{3.8}$$

This corresponds to the linear signal computed above, except that we do it for every node in one go (by using $\boldsymbol{\beta}^{(\ell)}$ instead of $\boldsymbol{\beta}_j^{(\ell)}$). The resulting matrix is $1 \times J^{(\ell)}$, with an entry for every hidden node in the layer. We can then simply apply then activation function element-wise:

$$\boldsymbol{h}_i^{(\ell)} = \begin{cases} \boldsymbol{x}_i, & \ell = 0. \\ \begin{pmatrix} 1 & \eta\left(\boldsymbol{s}_i^{(\ell)}\right) \end{pmatrix}, & 0 < \ell < L. \end{cases} \tag{3.9}$$

It can be a bit hard to get a feel for this at first. Folllowing Abu-Mostafa *et al.* (2012b), it may be easier to visualize this in a chain. Starting from the input layer, we compute a linear signal based on the previous layer, apply the activation function, and pass it on to the next layer. Every signal is based on unique coefficients. Rinse and repeat:

$$\boldsymbol{x}_i = \boldsymbol{h}_i^{(0)} \overset{\beta^{(1)}}{\to} \boldsymbol{s}_i^{(1)} \overset{\eta(\cdot)}{\to} \boldsymbol{h}_i^{(1)} \overset{\beta^{(2)}}{\to} \boldsymbol{s}_i^{(2)} \overset{\eta(\cdot)}{\to} \boldsymbol{h}_i^{(2)} \overset{\beta^{(3)}}{\to} \dots \overset{\eta(\cdot)}{\to} \boldsymbol{h}_i^{(L-1)} \overset{\beta^{(L)}}{\to} s_i^{(L)} \overset{\sigma(\cdot)}{\to} \tilde{f}(\boldsymbol{x}_i) \tag{3.10}$$

**Output layer**. At the end of the chain, the final node works exactly like with a single layer. It observes the output of the final hidden layer $\boldsymbol{h}_i^{(L-1)}$, and calculates a linear signal $\boldsymbol{s}_i^{(L)} = \boldsymbol{h}_i^{(L-1)}\boldsymbol{\beta}^{(L)}$. The output function then either outputs this for continuous variables, or squashes the result to a probability. The strength of the network once again comes through the automatic feature learning, where the hidden nodes are used to obtain hidden 'variables' which better approximate the target function. Since this is a form of feature engineering, it is tempting to try to assign meaning to any node. This is not necessarily useful. Consider the disc target example: Many nodes were able to recreate a circle by being combined. However, each single node just provides a line, with no individual interpretation. Sometimes nodes are only meaningful in combination. This may be especially true with multiple layers.

**Definition 3.2.** A *feedforward neural network* with a $L-1$ hidden layers indexed by $\ell = 0, ..., L$, with $J^{(\ell)}$ nodes in a given hidden layer, output function $\sigma(\cdot)$, activation function $\eta(\cdot)$, a $1\times(k+1)$ feature 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

$$\begin{aligned}\tilde{f}(\boldsymbol{x}_i) &= \sigma\left(\beta_0^{(L)} + \sum_{j_{L-1}=1}^{J^{(L-1)}} \beta_{j_{L-1}}^{(L)}\eta\left(\dots\eta\left(\beta_{j_2,0}^{(2)} + \sum_{j_1=1}^{J^{(1)}}\beta_{j_2,j_1}^{(2)}\eta\left(\beta_{j_1,0}^{(1)} + \sum_{p=1}^{k}\beta_{j_1,p}^{(1)}x_{i,k}\right)\right)\right)\right) \\ &= \sigma\left(\tilde{\eta}\left(\dots\tilde{\eta}\left(\tilde{\eta}\left(\boldsymbol{x}_i\,\boldsymbol{\beta}^{(1)}\right)\boldsymbol{\beta}^{(2)}\right)\dots\right)\boldsymbol{\beta}^{(L)}\right),\end{aligned} \tag{3.11}$$

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

**Proposition 3.2.** The *marginal effect* of a feature $x_p$ evaluated at $\boldsymbol{x}_i$, in a neural network with a multiple hidden layers (as in definition 3.2) is the partial derivative

$$\begin{aligned}\gamma_p(\boldsymbol{x}_i) &= \sigma'\left(s_i^{(L)}\right)\left(\sum_{j_{L-1}=1}^{J^{(L-1)}}\beta_{j_{L-1}}^{(L)}\eta'\left(\boldsymbol{s}_i^{(L-1)}\right)\dots\eta'\left(\boldsymbol{s}_i^{(2)}\right)\left(\sum_{j_1=1}^{J^{(1)}}\beta_{j_2,j_1}^{(2)}\beta_{j_1,p}^{(1)}\eta'\left(s_{i,j_1}^{(1)}\right)\right)\right) \\ &= \sigma'\left(s_i^{(L)}\right)\eta'\left(\boldsymbol{s}_i^{(L-1)}\right)\operatorname{diag}\left(\dots\operatorname{diag}\left(\eta'\left(\boldsymbol{s}_i^{(1)}\right)\operatorname{diag}\left(\boldsymbol{\beta}_p^{(1)}\right)\boldsymbol{\beta}_{1:}^{(2)}\right)\dots\right)\boldsymbol{\beta}_{1:}^{(L)},\end{aligned} \tag{3.12}$$

with $\sigma'(s) = \partial\sigma(s)/\partial s$, $\eta'(s) = \partial\eta(s)/\partial s$, $s_i^{(\ell)}$ as defined through equation (3.8) and (3.9), $\operatorname{diag}\left(\boldsymbol{\beta}_p^{(1)}\right)$ 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 V.6. □

The diagonal matrices correspond to element-wise multiplication, which supplies multiplication with inner derivatives following the chain rule. Once again, we observe how initial changes in $x_{i,p}$ first affect the initial hidden layer through $\boldsymbol{\beta}_p^{(1)}$, and the subsequent changes the affect the next hidden layer, flowing through the network until it finally affects the output layer.

Proposition 3.2 provides a general estimator for the marginal effect of $\boldsymbol{x}_p$ for a neural network of any size. The proposition is not well-suited for direct implementation since it only holds generally for a single $\boldsymbol{x}_i$ (the full $n\times k$ $\boldsymbol{x}$ leads to a dimensional mismatch), but appendix III.1 provides an algorithm I designed which calculates the effects in an equivalent manner for an entire $n\times 1$ $\boldsymbol{x}_p$, at a time, providing a relatively efficient implementation. This is the estimator I will investigate in the paper

## 4. Methodology: Let's get this simulation started!

'Science is hard — really fucking hard'.

– *Christie Aschwanden (2015)*

The following section explains the methodical approach taken in this study, leaving the technical stuff to the appendices. First, I motivate the use of simulation to shed light on estimator properties. Second, I define the Monte Carlo set-up I implemented, and introduce the various data-generating processes I analyse. Afterwards, I briefly review my implementation of neural networks, and a few basic econometric estimators I implemented as comparison.

### 4.1. First the why...

Analytically derived results remain the gold standard in econometrics. Despite this, I have chosen to implement a simulation study rather than derive properties. The main motivation for this, which I present in the first section below, is that it may be quite hard to derive general properties for neural networks, since they impose few assumptions on the DGP. Instead Monte Carlo simulation, as I argue next, presents an easier approach to obtain valid properties for the estimator, even if they only hold within observed cases.

#### 4.1.1. Analytical properties may be quite difficult to derive for neural networks

Neural networks, at their core, are an *m-estimator* since they minimize an average of subfunctions over observations $Q_N(\boldsymbol{\beta}) = \frac{1}{n}\sum_{i=1}^{n} q(y_i, \boldsymbol{x}, \boldsymbol{\beta})$, particularly the negative log-likelihood (see appendix III.1 for formulas).[39] This provides a framework for deriving properties, particularly consistency and asymptotic normality, since general theorems exist for such extremum estimators (Cameron & Trivedi, 2005; Amemiya, 1985). However, several problems present themselves. First, these theorems depend on properties of $Q_0(\boldsymbol{\beta})$, the optimal objective function for the neural network given the underlying data-generating process. The typical approach to establishing consistency is to impose assumptions on the DGP, and then apply a law of large numbers to the estimator, and show that it converges in probability to the optimum (Cameron & Trivedi, 2005, p. 131). Since neural networks impose few assumptions on the DGP, this venue is difficult.

Even if one manages to do this, further problems present themselves. The general consistency theorems of Amemiya (1985) indicate that, given the multiple local minima of interesting networks, one will likely only be able to show that the estimator is consistent for at least one of these local minima. It would then be necessary to judge the performance of these local minima which, as reeviewed in section 3.2.2, there is no guarantee provide meaningful results.

Finally, note that neural networks are an m-estimator not for the true process, but rather for an approximative model. Even if the neural network is a consistent estimator for the optimal network given the chosen architecture, there is no guarantee that this architecture allows low enough approximation error for meaningful analysis. Any analytical solutions for this will necessarily depend on assumptions on the DGP. The key point here is not that analytical results are unobtainable, but rather that they are difficult to find, and may not resolve all relevant questions without substantial assumptions.

[39] For a general review of the properties of m-estimators, see section 5.2. in Cameron & Trivedi (2005).

#### 4.1.2. Fortunately, Monte Carlo simulation provides a reliable alternative

I have chosen an alternative route which I am both more capable of reliably executing, and which I would argue elegantly handles some of the troubles of an analytical approach. Monte Carlo studies of estimator properties relies on repeatedly sampling data from a fictional DGP, and then analysing the results of applying the estimator to each of these samples.[40]

**Definition 4.1.** A Monte Carlo study of the properties of an estimator $\hat{\theta}$ proceeds:

1. Define a DGP which specifies a probability distribution $p(y_i, \boldsymbol{x}_i)$.
2. For simulation $m = 1, ..., M$:
   2.1. Draw $\boldsymbol{x}_i^{(m)}, y_i^{(m)} \sim p(y_i, \boldsymbol{x}_i)$.
   2.2. Estimate $\hat{\theta}^{(m)}$ based on $\boldsymbol{x}_i^{(m)}, y_i^{(m)}$.
3. Calculate summary statistics for $\hat{\theta}^{(1)}, ..., \hat{\theta}^{(M)}$.

The validity of the summaries is ensured by independent draws within each simulations, which ensures that various $\hat{\theta}^{(m)}$ are independent, and hence allows application of a law of large numbers:

**Theorem 4.1** (Kolmogorov's law of large numbers)**.** If the stochastic variables $\hat{\theta}^{(1)}, ..., \hat{\theta}^{(M)}$ are independent and identically distributed and if (and only if) $\mathrm{E}[\hat{\theta}]$ exists and $\mathrm{E}[|\hat{\theta}|] < \infty$ then

$$\frac{1}{M} \sum_{m=1}^{M} \hat{\theta}^{(m)} \underset{M \to \infty}{\to} \mathrm{E}\left[\hat{\theta}\right]. \tag{4.1}$$

*Proof.* A proof is provided in (Rao, 1973, p. 114). □

Traditional uses often focus on parameter estimates, but the approach is equally valid for estimates of effects following section 2.3.2, such as expectations and marginal effects. It also allows us to estimate the MSE, which in turn allows us to establish consistency. Hence it allows me to analyse the properties of main interest within the confines of the defined DGP.

This also presents the main draw-back of the method: Results are only valid for the simulated probability distribution $p(y_i, \boldsymbol{x}_i)$. If this distribution depends on some parameters $\boldsymbol{\beta}$, they are only valid for the covered parameters. While this is not ideal, note that the discussion above indicated that formal proofs would likely also entail assumptions on the DGP. Below, I try to provide as general results as possible by simulating different processes, with variable parameters.

There are two main advantages to the simulation approach. First, it is relatively easy to implement, since it only requires designing a procedure for simulation, as well as coding up the estimators of interest. Secondly, it allows direct comparison between true values $\theta$, and estimates $\hat{\theta}$. Thus, even if neural networks tend to converge to local minima, or if optimal solutions exhibit large approximation error, the approach can shed light on whether solutions are good enough.

Somewhat less formally, I would also argue that simulation results are easily trust-worthy. Rigorous proofs requires that readers meticulously check the math. This approach, on the other hand, requires estimation of models on actual data. If the theory or methods are wrong, then this is directly observable from the results, since estimation will fail.

[40] See Cameron & Trivedi (2005), Davidson & MacKinnon (1993) or Hendry (1984).

### 4.2. ... and then the how.

So, how did I actually design the simulation study? Given the limitations of simulation studies, the main goal is to investigate a sufficiently broad class of scenarios to allow for somewhat general claims about properties. Below, I first define a broad theoretical scenario, and then provide the actual DGP's I investigated. I then briefly review my implementation of neural networks and a few econometric estimators, leaving details for the appendices.

#### 4.2.1. Simulation set-up

The simulation set-up used will feel quite familiar to econometricians, with only a single main change introducing non-linearity. I will consider analysis of a $n \times 1$ vector $\boldsymbol{y}$ of dependent variables (regressands, labels), which depends on a $n \times k$ matrix $\boldsymbol{x}$ of independent variables (regressors, features). In the basic set-up, it is computed in accordance with the theoretical scenario outlined in proposition 2.6, so that observations follow an independent, common form $y_i = f(\boldsymbol{x}_i)$, and all regressors are weakly unconfounded, leading to clean causal interpretations. I relax the latter assumption later. $f(\boldsymbol{x}_i)$ consists of the expectation $\mathrm{E}[y_i|\,\boldsymbol{x}_i] = g(\boldsymbol{x}_i, \boldsymbol{\beta})$ as well as linearly seperable stochastic noise $u_i$. I investigate various specifications of the deterministic part $g(\boldsymbol{x}_i, \boldsymbol{\beta})$ as outlined in the next section.

**Definition 4.2.** The *data-generating process* (DGP) for *regression* scenarios consist of

$$\begin{aligned} y_i &= g(\boldsymbol{x}_i, \boldsymbol{\beta}) + u_i, \\ \boldsymbol{x}_i \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Omega}), &\quad \boldsymbol{\mu} \sim \mathcal{N}(0, \boldsymbol{I}_k), \quad \mathrm{diag}(\boldsymbol{\Omega}) = \mathbf{1}, \\ \boldsymbol{\beta} \sim \mathcal{N}(\mu_\beta, \boldsymbol{I}_k), &\quad u_i \sim \mathcal{N}(0, \sigma_u^2). \end{aligned} \tag{4.2}$$

$\boldsymbol{x}_i$ is drawn from a multivariate normal distribution with means drawn from a standard normal distribution. I ensured unit variance while allowing non-zero correlation by drawing a symmetric, positive-definite matrix and imposing unit variance across the diagonal. Finally, $g(\boldsymbol{x}_i, \boldsymbol{\beta})$ depends on various parametrizations, which is allowed through a normal column vector $\boldsymbol{\beta}$.

**Proposition 4.1.** The true causal effect of $x_{i,p}$ in a regression scenario (as in definition 4.2) is

$$\gamma_p(\boldsymbol{x}_i) = \frac{\partial\, \mathrm{E}[y_i|\,\boldsymbol{x}_i]}{\partial x_{i,p}} = \frac{\partial g\left(\boldsymbol{x}_i, \boldsymbol{\beta}\right)}{\partial x_{i,p}}. \tag{4.3}$$

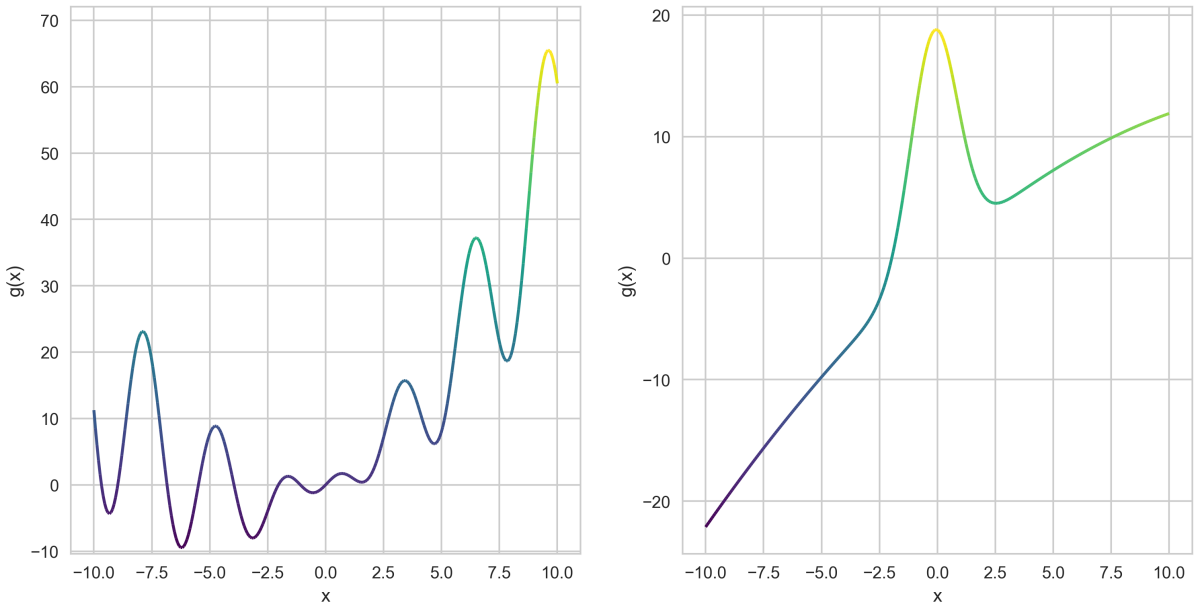
*Proof.* The result follows directly from definition 4.2. □

**Definition 4.3.** The *data-generating process* (DGP) for *classification* scenarios consist of:

$$\begin{aligned} y_i = 1\{y_i^* \geq 0\}, &\quad y_i^* = g(\boldsymbol{x}_i, \boldsymbol{\beta}) + u_i, \\ \boldsymbol{x}_i \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Omega}), &\quad \boldsymbol{\mu} \sim \mathcal{N}(0, \boldsymbol{I}_k), \quad \mathrm{diag}(\boldsymbol{\Omega}) = \mathbf{1}, \\ \boldsymbol{\beta} \sim \mathcal{N}(\mu_\beta, \boldsymbol{I}_k), &\quad u_i \sim LOG(0, 1). \end{aligned} \tag{4.4}$$

The binary case is based on a *latent variable* representation, where $y_i^*$ corresponds to the continuous case above, but we instead observe a variable with a cut-off (arbitrarily) set at zero. The expectation is now $\mathrm{E}[y_i|\,\boldsymbol{x}_i] = F_u\left(g(\boldsymbol{x}_i, \boldsymbol{\beta})\right)$ where $F_u(\cdot)$ is the cumulative distribution function (CDF) for the error term, which here is the sigmoid function.

**Figure 4.1:** *Wiggly and pointy non-linear functions*




*Notes*: Examples of two non-linear functions analysed (see appendix IV for details).

**Proposition 4.2.** The true causal effect of $x_{i,p}$ in a classification scenario (as in 4.3) is

$$\gamma_p(\boldsymbol{x}_i) = \frac{\partial \, \mathrm{E}[y_i|\, \boldsymbol{x}_i]}{\partial x_{i,p}} = F_u' \left( g \left( \boldsymbol{x}_i, \boldsymbol{\beta} \right) \right) \frac{\partial g \left( \boldsymbol{x}_i, \boldsymbol{\beta} \right)}{\partial x_{i,p}} \tag{4.5}$$

*Proof.* The result follows directly from definition 4.3. □

The analysed cases are intended to show quite general regression and classification scenarios, and I would expect results to hold for different distributions for variables and error terms. However, due to the limitations of the chosen method, I can only check so much.

### 4.2.2. Scenarios of increasing non-linearity

The set-up allows analysis of any CEF by defining the underlying DGP function $g(\boldsymbol{x}_i, \boldsymbol{\beta})$. I mainly investigate nine different scenarios, with various levels of non-linearity. The idea is to investigate a relatively large number functions, to minimize the risk that results are specific to the analysed cases. I describe the investigated scenarios here. Formal derivations of marginal effects and gradients are relegated to appendix IV. There, one may also find visualizations of all functions with one or two explanatory regressors (since more strains the dimensional sensibilities). The scenarios are summarized in table 4.1. The first two scenarios should be quite familiar to any econometrician. The first considers a linear signal $g(\boldsymbol{x}_i, \boldsymbol{\beta}) = \boldsymbol{x}_i \, \boldsymbol{\beta}$, leading to OLS/logit as the optimal (MLE) solution. The second DGP augments this with a vector of polynomial terms, a classical way to account for increasing/decreasing marginal effects. The third one adds further a set of cubic terms, which is seen less often but still easily possible. These cases highlights the idea that some non-linearities may be easy to handle through simple feature transformations. However, they also investigate whether neural networks provide a plausible alternative.

**Table 4.1:** *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 IV.

The next three scenarios are based on functions which still present straightforward non-linearities, which could conceivable have economic relevance, but which economists are unlikely to think of (or have good ways to augment models for). The first is a function which follows a global trend, but tends to walk about in a random fashion ('wiggling' around the trend). An example is shown in figure 4.1. Versions of such a function could, for example, reflect the gains from years of education, where there will likely be large differences in marginal effects in years where one typically finishes a degree, but an overall positive effect. The second one (also shown in figure 4.1) again shows a clear global trend, but one clear departure from it (a 'point'). In Hartford *et al.* (2016) such a function is used to show price sensitivities which differ throughout the day. Finally, I have also shown a trigonometric polynomial which shows that effects may be *periodic*, where marginal effects tend to be positive or negative at different times, rather than globally monotonous. These cases investigate how networks perform in scenarios of relatively simple non-linearity, which we may not know how to model. They also present traps from continuing to model only average marginal effects, when there may be significant heterogeneity.

The final three scenarios are highly non-linear. They are inspired by similarly named functions which are used for testing algorithms for numerical optimization (Surjanovic & Bingham, 2017). Hence, they are multimodal with many local minima, which presents traps for such algorithms. It also implies that the marginal effects are very much not monotone, and indeed prone to large shifts from small changes in regressors. The idea is not that such functions are directly applicable to economics, but rather to present worst-case scenarios for analysis, and check whether neural networks can still recover the underlying marginal effects.

#### 4.2.3. Implementation of neural networks

All neural networks in this paper are implemented using the Python module *Scikit-learn* (Pedregosa *et al.*, 2011). It provides tools for training neural networks which follows the approach outlined in the previous sections. I describe the implementation in more technical detail in appendix III.1. Particularly, I describe my choices for hyperparameters for optimization algorithms, regularization and network architecture. However, the disinterested reader may just note that I generally follow default choices in the current literature on neural networks.

**Table 4.2:** *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 estimation based on true $g(\boldsymbol{x}_i, \boldsymbol{\beta})$ |
| **OLS (I)** | Ordinary least squares regression. |
| **OLS (II)** | Second-order polynomial expansion of $\boldsymbol{x}_i$. |
| **Logit (I)** | Logistic regression. |
| **Logit (II)** | Second-order polynomial expansion of $\boldsymbol{x}_i$. |

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

There is no support for the computation of marginal effects within the module, since it is designed for data scientists and hence focused on prediction. Instead, I designed and implemented an algorithm that provides an equivalent computation to the propositions of section 3, in a decently efficient manner. It is also provided in appendix III.1.

#### 4.2.4. Comparison with basic econometric estimators

I also implemented a few standard econometric estimators as comparison. The implementation is described in more detail in appendix III. They present an econometric baseline in the form of a first-best estimator (MLE), and the most basic linear methods. This is by no means an exhaustive comparison, nor are they necessarily the most relevant in each case. Further research comparing neural networks to non- and semi-parametric estimators (e.g. kernel regression) in particular would be very relevant. Note though that the main argument here is that neural networks are well-behaved, which does not directly depend on comparison.

I implemented *maximum likelihood estimation* MLE) as a first-best estimator. It often shows the best we could possibly do if we knew the underlying DGP, since it is the most efficient of all consistent, asymptotically normal estimators (Cameron & Trivedi, 2005). I formally derive the estimator in appendix III. The estimators follows the standard structure, but depend on $g(\boldsymbol{x}_i, \boldsymbol{\beta})$ and its gradient $\partial g(\boldsymbol{x}_i, \boldsymbol{\beta})/\partial \boldsymbol{\beta}$. These are provided in appendix IV.[41] This also highlights the issue with MLE: It requires precise knowledge of the DGP, which is often implausible.

I also implemented basic, well-known linear methods as a baseline comparison. Ordinary least squares (OLS) remains popular in undergraduate courses and practice, particularly because it provides the best linear approximation even if the underlying process is non-linear. Comparisons with it allow us to judge how much better we can do by allowing for better approximation. In the binary case, OLS is often discarded because it is not confined to probabilities, and logistic regression is used instead. While this is no doubt an improvement, logistic regression also provides a solely linear classifier. All the implemented estimators are summarized in table 4.2.

All in all, the simulation study ended up depending on a somewhat sprawling code module, which is available on GitHub, and described in detail in appendix II. The results should be understandable without consulting it, but it may provide further insights.

[41] For some DGP's the MLE is not well-behaved, either due to frequent failure of optimization or lack of identification in $g(\boldsymbol{x}_i, \boldsymbol{\beta})$. I have not prioritized adapting MLE to fit these cases, as it only serves as a comparison.

## 5. Results I: Neural networks provide a well-behaved big data estimator...

'If you can't get it right as $n$ goes to infinity, you shouldn't be in this business'.

– *Clive W.J. Granger (n.d.)*[42]

This section will show that neural networks provide a well-behaved estimator of conditional marginal effects. First, I will analyse its basic properties, and show that neural networks provide a consistent estimator in all investigated scenarios which, although less efficient than the MLE, seem to allow for analysis with realistic sample sizes. Second, I will argue that that although linear methods may be good enough for estimation of average effects, neural networks allow us to investigate whether such averages miss heterogeneity. Finally, the third section discusses how to test significance, and introduces a bootstrapping algorithm for this purpose. All in all, this provides evidence that neural networks can provide a useful tool for analysis of marginal effects.

### 5.1. Estimates are consistent, but not terribly efficient

If you ask a theoretical econometrician to choose among a set of estimators, she likely offers a quick answer: I would like the most efficient among consistent estimators, please. This does not bode well for neural networks, which can at best be consistent for an error rate $\epsilon > 0$ (see section 2.3.3). Applied econometricians may hesitate, since they know consistent estimators may rely on strong assumptions (e.g. MLE) or be quite inefficient in practice (e.g. kernel regression). They may instead implicitly rely on a framework like the one I introduced in section 2.3.2, where estimators are judged on the MSE for a given sample size. I will show that neural networks may do well then. First, I show a simple example where neural networks provide the preferred plausible estimator in non-linear scenarios. Second I will show that this is because neural networks provide a consistent estimator for a low enough error to be useful. Finally, I investigate the convergence rate, arguing that although bias goes to zero relatively fast, estimates have a high variance, which is why the estimator is mainly relevant with relatively big data.

#### 5.1.1. A first look: We might just prefer neural networks

Consider first a simple example with five regressors $\boldsymbol{x}_i$, and 100.000 observations.[43] In my framework, the preferred estimator minimizes the MSE between estimates $\hat{\gamma}$ and the true marginal effects in the DGP $\gamma$. The simulation provides a direct estimate of this since both are observed. Averaging across all $k$ regressors $x_{i,p} \in \boldsymbol{x}_i$ allows us to observe all estimator simultaneously.

$$\text{Mean MSE}\left(\hat{\gamma}\right) = \frac{1}{k}\sum_{p=1}^{k}\frac{1}{n}\sum_{i=1}^{n}\left(\hat{\gamma}_p(\boldsymbol{x}_i) - \gamma_p(\boldsymbol{x}_i)\right)^2. \tag{5.1}$$

To get an estimate of the expectation of this quantity, I consider the average across $M$ simulations. Table 5.1 shows this quantity for the nine investigated scenarios. The MLE is presented as a first-best alternative, but since the DGP is unknown, it is typically not plausible. The preferred estimator among the plausible ones is marked in bold. In the linear case, OLS is preferred. If the

[42] Wooldridge (2015) notes that Mr. Granger is known to have once remarked it, but no source is provided.

[43] A note on the scenarios in general: Dimensions of the various variables may seem slightly arbitrary, which they honestly are. The key notion is to test out different specifications, to check whether results rely heavily on any one. Note that I have tested many more combinations in my analysis, but can only present some results.

**Table 5.1:** *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 | 0.00 | 0.01 |
| | (0.000) | (0.000) | (0.000) | (0.000) | (0.001) | (0.002) |
| **Polynomial (2)** | 0.00 | 0.00 | 3.25 | **0.00** | 0.16 | 0.13 |
| | (0.000) | (0.000) | (0.013) | (0.000) | (0.020) | (0.013) |
| **Polynomial (3)** | 0.00 | 0.00 | 36.20 | 13.21 | 3.82 | **2.50** |
| | (0.000) | (0.000) | (0.189) | (0.096) | (0.340) | (0.402) |
| **Pointy** | 0.00 | 10.66 | 60.51 | 34.15 | 2.41 | **1.65** |
| | (0.000) | (6.850) | (0.112) | (0.154) | (0.458) | (0.171) |
| **Wiggly** | 0.00 | 0.00 | 8.27 | 7.71 | 0.85 | **0.51** |
| | (0.000) | (0.000) | (0.028) | (0.032) | (0.180) | (0.051) |
| **Trig. pol (3)** | 0.00 | 0.00 | 0.37 | 0.09 | **0.02** | 0.04 |
| | (0.000) | (0.000) | (0.001) | (0.001) | (0.002) | (0.023) |
| **Ackley** | 0.00 | 2.79 | 2.89 | 2.85 | 1.76 | **0.80** |
| | (0.000) | (0.130) | (0.007) | (0.007) | (0.865) | (0.066) |
| **Rastrigin** | 0.00 | 0.00 | 44.64 | 44.65 | 6.71 | **5.50** |
| | (0.000) | (0.001) | (0.083) | (0.083) | (0.837) | (0.968) |
| **Drop-Wave** | 0.00 | ... | 8.04 | 7.88 | 3.97 | **0.75** |
| | (0.000) | (–) | (0.017) | (0.016) | (0.153) | (0.076) |

*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 4.1 and 4.2. Estimators were trained on 100,000 observations. Reported values are from a test set of equal size.

non-linearity can be accounted for by quadratic terms, the expanded linear method is preferred. However, for the remaining cases, neural networks provide a far closer estimate. In all cases but one this is best achieved through the network with two hidden layers. It is possible that a different network might perform even better. All in all, the table presents a picture of neural networks as decent estimators.

Table 5.2 presents a classification scenario with two regressors. Recall that it is basically the same scenario, except $g(\boldsymbol{x}_i, \boldsymbol{\beta})$ functions as a latent variable, and we instead observe only zeroes and ones. Overall, the same picture presents itself. For most of the non-linear cases, neural networks soundly beat both the simple linear probability model (i.e. OLS), and the slightly more sophisticated, but still linear, logistic regression.

This suggests that neural networks are usable, but it is not enough to conclude that they provide a good estimator. After all, in most non-linear cases the estimator still makes large errors, and often much larger than maximum likelihood estimation. I should probably investigate the estimator further. A perhaps wiser way is to build up a case for its properties by considering how the estimator depends on one key parameter: Sample size.

**Table 5.2:** *Binary response or no, neural networks still work*

| | DGP | MLE | OLS (I) | OLS (II) | Logit (I) | Logit (II) | NN (I) | NN (II) |
|---|---|---|---|---|---|---|---|---|
| **Linear** | 0.00 | 0.00 | 0.00 | 0.00 | **0.00** | 0.00 | 0.00 | 0.00 |
| | (0.000) | (0.000) | (0.000) | (0.000) | (0.000) | (0.000) | (0.000) | (0.000) |
| **Polynomial (2)** | 0.00 | 0.00 | 0.03 | 0.01 | 0.02 | **0.00** | 0.00 | 0.00 |
| | (0.000) | (0.000) | (0.000) | (0.000) | (0.000) | (0.000) | (0.000) | (0.000) |
| **Polynomial (3)** | 0.00 | 0.00 | 0.10 | 0.09 | 0.08 | 0.06 | **0.01** | 0.01 |
| | (0.000) | (0.000) | (0.001) | (0.001) | (0.001) | (0.000) | (0.002) | (0.001) |
| **Pointy** | 0.00 | 0.15 | 0.06 | 0.05 | 0.04 | **0.01** | 0.01 | 0.01 |
| | (0.000) | (0.038) | (0.002) | (0.001) | (0.001) | (0.001) | (0.002) | (0.007) |
| **Wiggly** | 0.00 | 0.00 | 0.30 | 0.28 | 0.29 | 0.24 | **0.02** | 0.02 |
| | (0.000) | (0.000) | (0.003) | (0.002) | (0.002) | (0.002) | (0.003) | (0.005) |
| **Ackley** | 0.00 | 0.32 | 0.35 | 0.35 | 0.35 | 0.35 | 0.14 | **0.08** |
| | (0.000) | (0.005) | (0.001) | (0.001) | (0.001) | (0.001) | (0.014) | (0.014) |
| **Rastrigin** | 0.00 | 0.00 | 0.99 | 0.99 | 0.99 | 0.99 | 0.41 | **0.18** |
| | (0.000) | (0.000) | (0.002) | (0.002) | (0.002) | (0.002) | (0.052) | (0.034) |
| **Drop-Wave** | 0.00 | 0.03 | 0.62 | 0.62 | 0.62 | 0.62 | 0.09 | **0.03** |
| | (0.000) | (0.095) | (0.002) | (0.002) | (0.002) | (0.002) | (0.004) | (0.004) |

*Notes:* The table shows MSE between estimated and true marginal effects, averaged across all regressors. Each row represents 100 simulations with 2 regressors and binary output. Cells show averages across simulations, with standard errors in parentheses. DGPs (rows) and estimators (columns) are described in table 4.1 and 4.2. Estimators were trained on 100.000 observations. Reported values are from a test set of equal size.

### 5.1.2. That's nice, but is it consistent?

This leads naturally to that favourite of econometricians, *consistency*. Informally, this means that if we observed an infinite sample, the estimate should provide the true value. Formally, we say an estimator is (weakly) consistent if it converges in probability to the true value, $\text{plim}(\hat{\theta}) = \theta$ (Cameron & Trivedi, 2005). However, a slight adjustment is necessary. Recall from the universal approximation theorem (see section 2.3.3) that neural networks can only approximate the true function up to an error $\epsilon > 0$ (which can be made arbitrarily small by increasing network size). This seem to directly preclude consistency, since the error will remain even asymptotically. However, we may instead look at whether the estimate is consistent for that of the optimal neural network $\tilde{\theta}$, rather than the true value $\theta$. Although this may make theoreticians shudder, such an estimator may be preferable with finite samples if the approximation error is low.

**Definition 5.1.** An estimator $\hat{\theta}$ is *consistent* for the value $\tilde{\theta}$ if, for any $\epsilon > 0$ :

$$\lim_{n\to\infty} \mathrm{P}\left(\left|\hat{\theta} - \tilde{\theta}\right| > \epsilon\right) = 0 \tag{5.2}$$

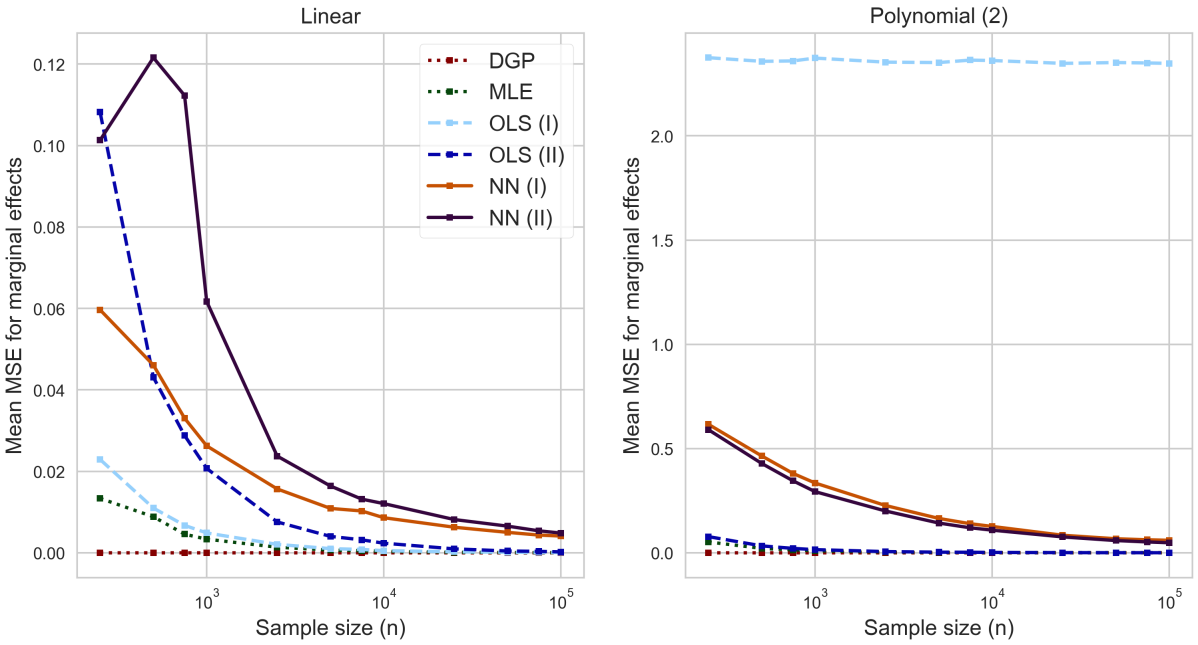
This definition is not easy to work with directly, since we do not observe the probability that we miss the target by $\epsilon$. Instead, we may look at whether MSE converges to zero (which is one of the reasons minimizing MSE provides a nice estimation objective)

**Proposition 5.1.** An estimator is consistent if it converges in mean square:

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

*Proof.* A brief proof, courtesy of Rao (1973), is provided in appendix V.7. □

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



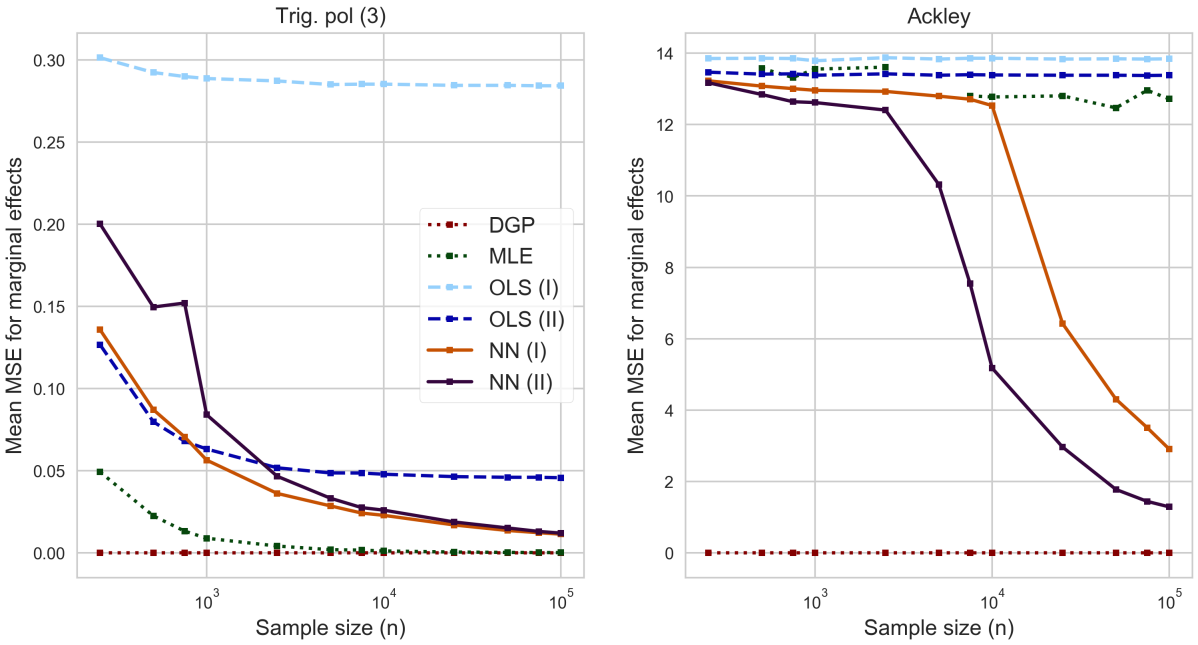

*Notes*: The figure shows MSE between estimated and true marginal effects, averaged across all regressors. Each figure represents a simulation study for the given DGP with 2 regressors and continuous output. In each study, 100 simulations were repeated for each sample size indicated by a marker. Sample 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 4.1 and 4.2.

Two problems present themselves. First, the optimal neural network is generally unknown, so I do not observe $\tilde{\theta}$. However, if neural networks are to be useful, they need both consistency and low approximation error. In practice, it should suffice to check whether overall error converges to a low value. Secondly, I cannot let $n \to \infty$ in practice. However, I can investigate progressively larger samples, and see whether MSE continues to decrease within observable bounds. If the error comes close to zero, or at least continues to decrease, I judge that consistency for an acceptable error level is likely. Fortunately, this is what I observe.

I repeated simulation studies starting from 250 observations and progressively increased the sample up to 100.000 observations. Each provides an estimate of the MSE for the given sample size, so I can observe whether it converges towards zero. Convergence is typically not linear, so I show the observations using a logarithmic scale. Figure 5.1 shows this for the two basic scenarios considered. Both neural networks (with one and two hidden layers respectively) seem consistent: Their square errors decrease as samples increase, and do not seem to flatten out.

However, neural networks are clearly not the preferred estimators at any point. In the basic linear scenario, OLS outperforms them both with and without quadratic terms. In the second case, OLS is clearly biased, but adding quadratic terms eliminates this. These are the cases where assumptions for these estimators hold, so the results are not surprising. Despite this, it serves us well to remember that, for smaller samples in particular, neural networks may be suboptimal. One reason is that linear methods converge to their optimal solutions much faster. This is a general result, even with more non-linearity.

**Figure 5.2:** *Neural networks remain consistent for complex scenarios*




*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 2 regressors and continuous output. 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 4.1 and 4.2.

Despite this, linear methods are not very useful for more complex models. Even though they provide the best linear approximations, these may be a far way off from the true values. Figure 5.2 shows convergence for two of the more non-linear scenarios. For a trigonometric polynomial of third order, neural networks are clearly superior for larger samples, and appears consistent for a low error level. In the highly multimodal Ackley scenario, this is even more clear: For small samples, no estimators can deduce the underlying structure. But as samples increase, the consistency of neural networks become apparent. Indeed, we here see indications of the power of deep learning, as the two-layered neural networks converges faster than the simpler one.

### 5.1.3. Estimates appear to have high variance, so we may need big data

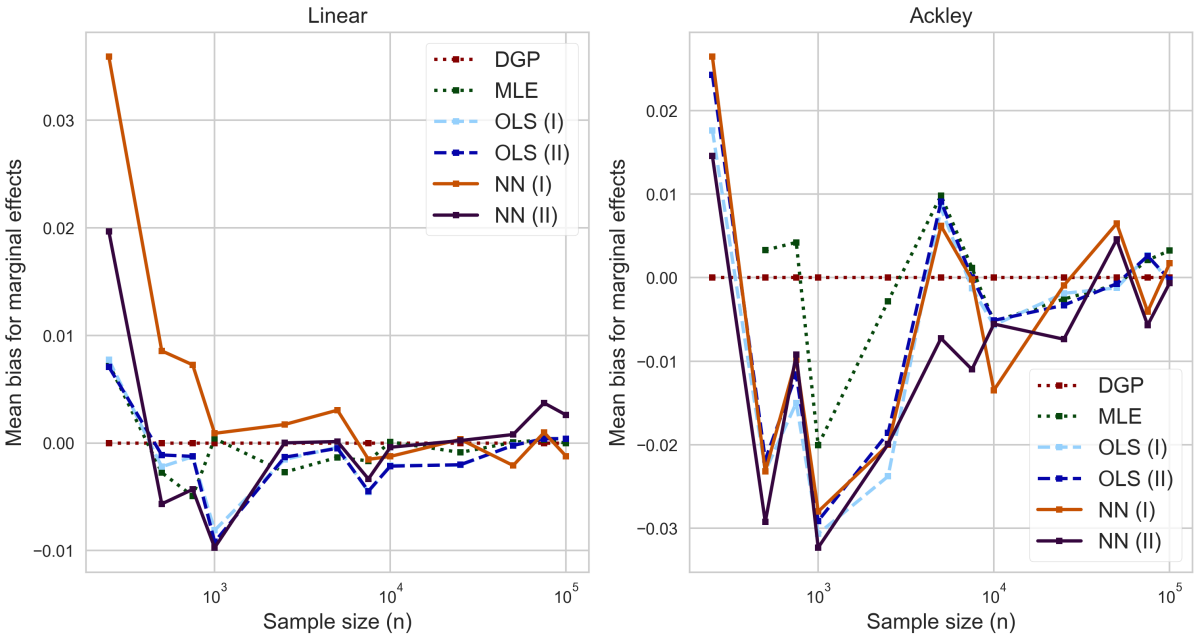
One obvious conclusion from the results above is that, although consistent, estimates converge somewhat slowly. I may dive slightly deeper into why that is the case. Particularly, I use the fact that MSE can be decomposed into the variance and the bias of the estimator:

**Proposition 5.2.** Consider an estimator $\hat{\theta}(\boldsymbol{x}_i)$ which estimates a model $\tilde{\theta}(\boldsymbol{x}_i)$ which approximates a true model $\theta(\boldsymbol{x}_i)$, as in section 2.3.2. The MSE can be decomposed as

$$\begin{aligned} \mathrm{E}\left[\left(\hat{\theta}-\theta\right)^2\right] &= \mathrm{E}\left[\mathrm{V}(\hat{\theta}|\,\boldsymbol{x}_i)\right] + \mathrm{E}\left[\mathrm{Bias}(\hat{\theta},\tilde{\theta}|\,\boldsymbol{x}_i)^2\right] + \mathrm{E}\left[\tilde{\epsilon}_A(\boldsymbol{x}_i)\right] \\ &= \text{Variance} + \text{Bias}^2 + \text{Approximation error}. \end{aligned} \tag{5.4}$$

*Proof.* The expression is derived in appendix V.8. It also elaborates in the individual terms, particularly how $\tilde{\epsilon}_A$ depends on the direct approximation error $\epsilon_A$. □

**Figure 5.3:** *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 2 regressors and continuous output. 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 4.1 and 4.2.

This directly accounts for the fact that we are estimating an approximate model by adding an irreducible term (which I derived under simple assumptions). Consistency for the approximate model will imply that MSE converges to this level. Beyond this term, the proposition provides the traditional decomposition of MSE into bias and variance (as in e.g. Friedman *et al.* (2009)).

The decomposition allows us to judge what is holding back convergence for neural networks: Either bias converges slowly, or the estimator has a large variance (approximation error does not change). The bias of an estimator is defined as $\text{Bias}(\hat{\theta}) = \text{E}[\hat{\theta}] - \tilde{\theta}$. I can estimate the bias directly, by looking at average difference between estimates and true marginal effects[44]

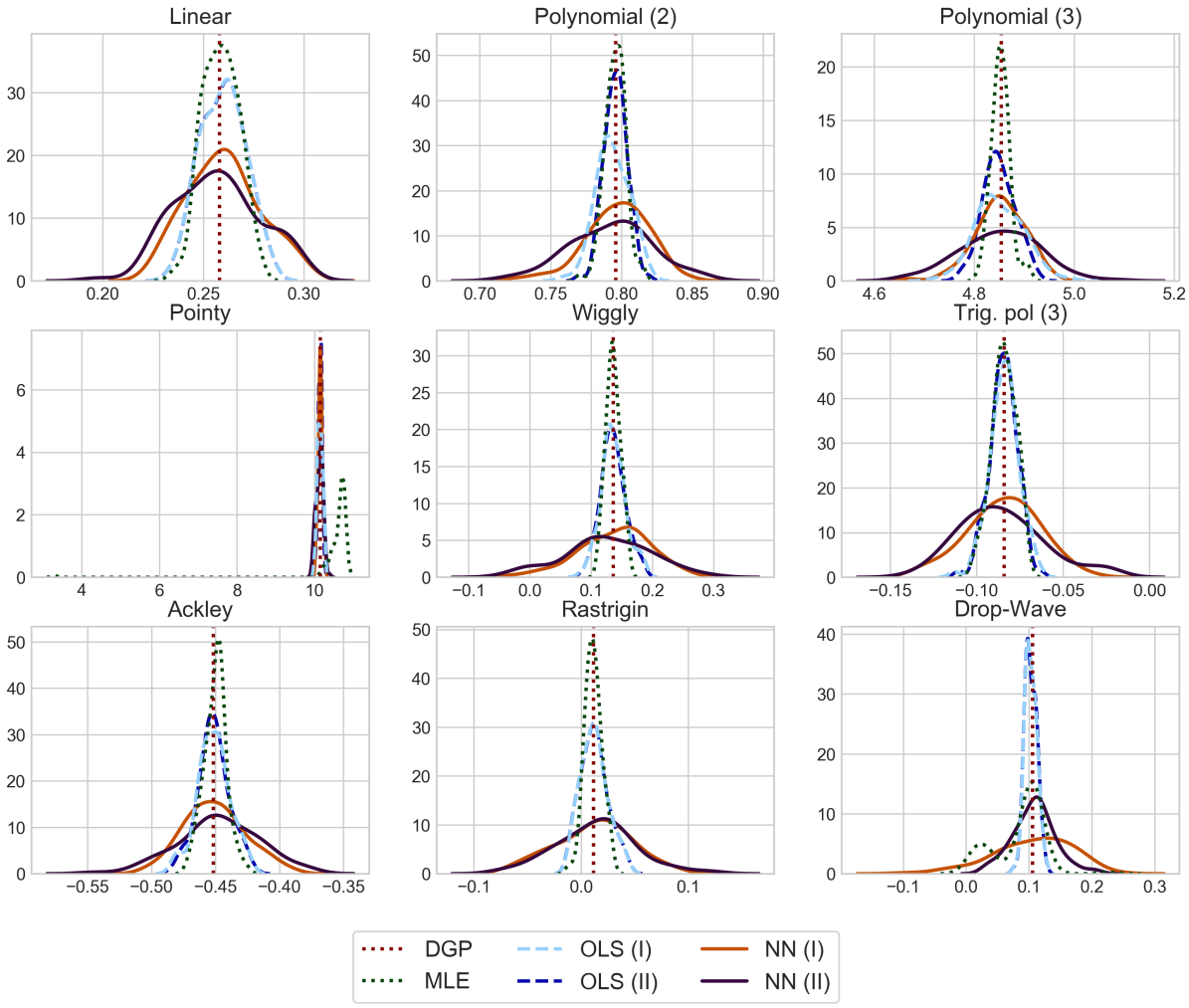
$$\text{ME}(\hat{\gamma}_p) = \frac{1}{n}\sum_{i=1}^{n}\left(\hat{\gamma}_p(\boldsymbol{x}_i) - \gamma_p(\boldsymbol{x}_i)\right). \tag{5.5}$$

Figure 5.3 shows the mean error for a single regressor for two of the scenarios considered. Other scenarios present similar pictures. The neural network is clearly biased for small samples in both cases. In the linear case, the bias goes to zero quite quickly. The case is less clear for the more complex Ackley function, but the bias also appears to go near zero for samples around 100.000. This indicates that the remaining error is largely due to high variance.

Compared to maximum likelihood, neural networks provide an estimator which converges relatively slowly. Given the above, this is due to the fact that neural networks provide a volatile estimator, with high variance, not because they are heavily biased (for medium-sized samples).

[44]It is unwise to look at averages over the $k$ regressors here. If some are positively biased, and some negatively, the aggregate effects may cancel out, falsely signalling unbiasedness. Hence I look at just one independent variable.

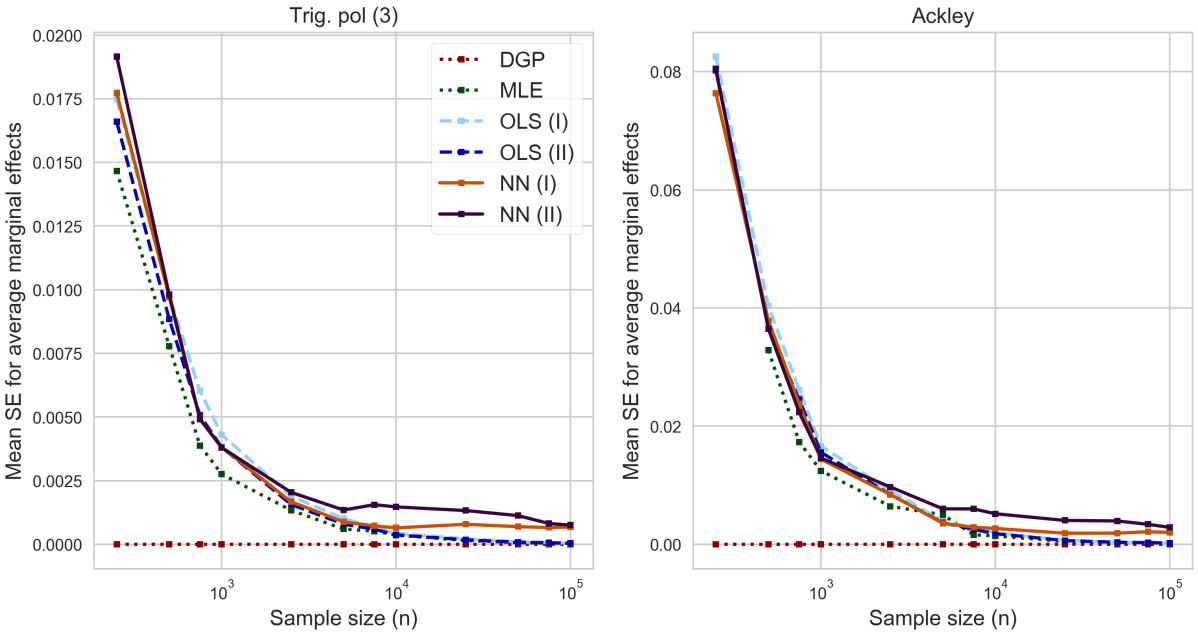
**Figure 5.4:** *Estimates of average marginal effects are nicely centered on true effects.*




*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 and continuous output. DGPs (panels) and estimators (lines) are described in table 4.1 and 4.2. Estimators were trained on 100,000 observations. Reported values are from a test set of equal size.

This makes sense, given what we know about neural networks. First, they have great representational capacity, since they can reflect many different functions. Intuitively, this would imply slower convergence to optimal solutions, as more candidates present themselves. Neural networks are also plagued by local minima, which means they can end up in various solutions which may differ slightly. This volatility does not invalidate neural networks as an estimator – Indeed, the large representational capacity is the main motivation for them – but practioneers should keep them in mind. Particularly, neural networks do not appear wise for small datasets. Even though linear methods may not yield strong optimal solutions, they converge fast to the best linear approximation. Still, as seen above, if the process is non-linear, neural networks provide a better estimate for even medium-sized samples.

**Figure 5.5:** *Squared error of estimated averages goes to zero quickly for all estimators*




*Notes*: The figure shows squared error between estimated and true average marginal effects, averaged across all regressors. Each figure represents a simulation study for the given DGP with 2 regressors and continuous output. 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 4.1 and 4.2.

## 5.2. Averages come easy, but tells not the full story

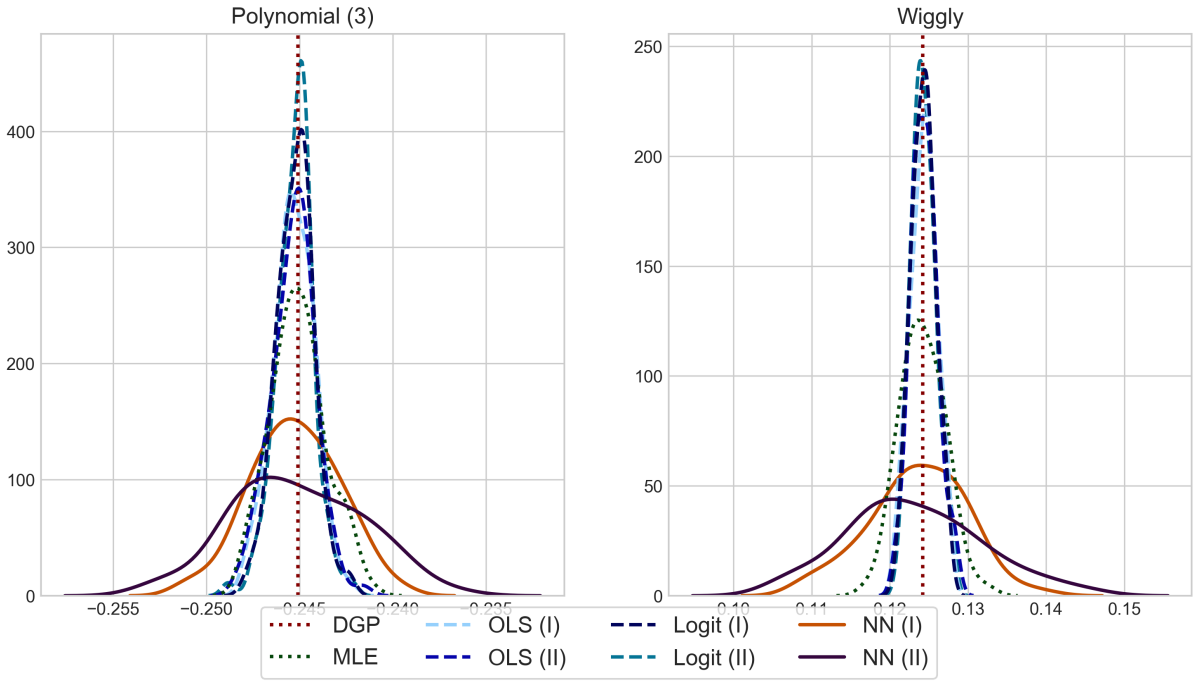
You might think 'But wait! If linear method are that bad, why haven't we noticed?' Economists have long used linear estimates with decent success. One could think this is because many economic relationships are in fact linear. More likely, it is because economists typically focus on *average* effects. This is partially path-dependent (in OLS, all marginal effects are equal to the average effect, so one could consider nothing else), partly practical (typically, we like to summarize findings in one number and averages are an obvious choice) and partly theoretical (economic models often focus on expectations). Whatever, the reason, we are often interested in estimating the average effect across the population, and it turns out linear methods do this quite well, even if the underlying process is non-linear. However, this may not be wise if there is significant heterogeneity, since we miss out on how effects may differ.

### 5.2.1. Linear methods estimates averages quite well...

The microeconometric literature often focus on identification of the *average treatment effect*, or the average treatment effect on the treated (see e.g. Angrist & Pischke (2008)). The challenge is similar to that for individual causal effect in that observed changes need not reflect causal changes. However, identification is simpler, since we do not necessarily need common form assumptions. Here, note that if individual effects are identified, then average effects are as well:

$$\bar{\gamma}_p = \int_{\mathcal{X}} \gamma_p(\boldsymbol{x}) \, \mathrm{d}\, \boldsymbol{x} \quad \underset{\infty \leftarrow n}{\leftarrow} \quad \frac{1}{n} \sum_{i=1}^{n} \gamma_p(\boldsymbol{x}_i). \tag{5.6}$$

**Figure 5.6:** *Even the linear probability model can recover average effects*




*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 2 regressors and binary output. DGPs (panels) and estimators (lines) are described in table 4.1 and 4.2. Estimators were trained on 100,000 observations. Reported values are from a test set of equal size.

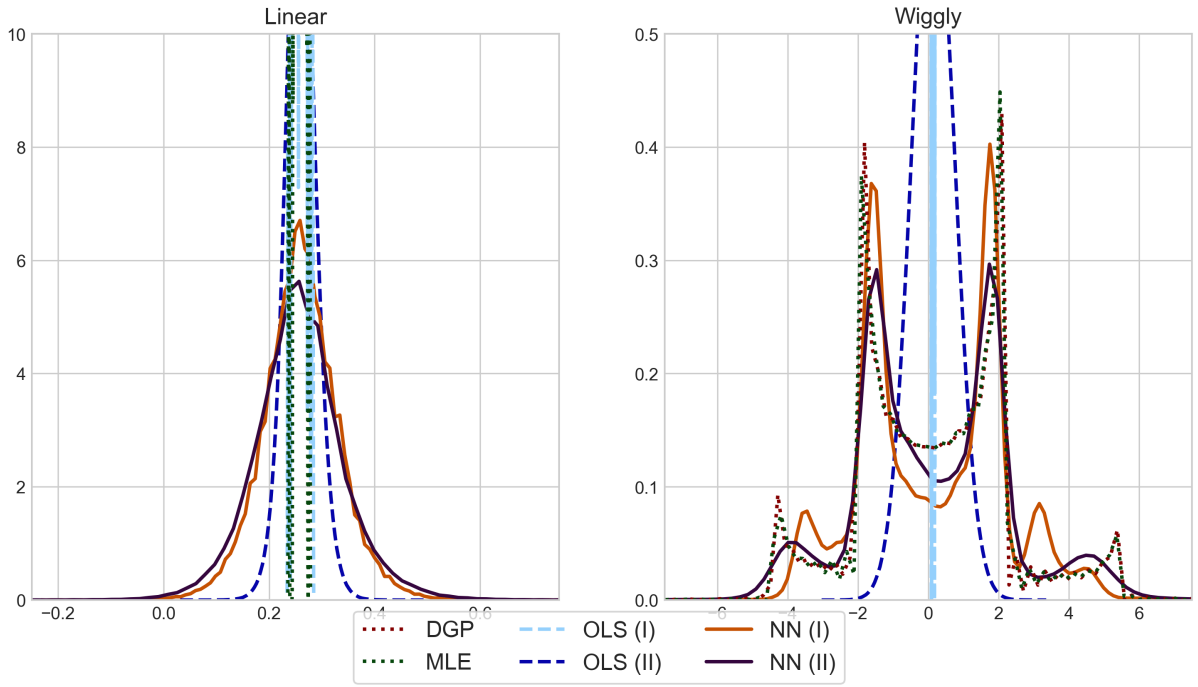
Figure 5.4 shows the distribution of the estimates of this average, $\hat{\bar{\gamma}}_p(\boldsymbol{x}_i) = \frac{1}{n}\sum_{i=1}^{n}\hat{\gamma}_p(\boldsymbol{x}_i)$ in given scenarios, as well as the true average effect for the DGP. There are two key takeaways. First, the estimated average effects are nicely centered on the true average effects for all estimators in all cases. Second, said distributions is markedly wider for neural networks in all cases, while linear estimators are more closely distributed around the true effect. This reflects the fact that linear models converge quickly to their optimal estimates as observations increase, leading to a less volatile estimate of the overall average. Consider again a MSE measure

$$\text{Mean SE}\left(\hat{\bar{\gamma}}\right) = \frac{1}{k}\sum_{p=1}^{k}\left(\hat{\bar{\gamma}}_p(\boldsymbol{x}_i) - \bar{\gamma}_p(\boldsymbol{x}_i)\right)^2. \tag{5.7}$$

Figure 5.5 shows that all estimators appear consistent for averages as I simulate increasing sample sizes. Linear methods converge nicely to the true value, and does so faster than the neural network. Even though linear estimates may be worse for individuals, they appear quite capable of estimating the average effect.

This can be seen even more clearly by again considering the binary case. Figure 5.6 shows the distribution estimated averages for two scenarios. Note again how the optimal linear estimator, logistic regression, is centered much more closely on the DGP. However, so is the linear probability model (i.e. OLS). This shows how linear methods converge easily to the average, even when they cannot uncover the true process: The linear probability model is not restricted to probabilities, and hence provides a poor model for individual probabilities (Cameron & Trivedi, 2005, 471).

**Figure 5.7:** *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 and binary output. DGPs (panels) and estimators (lines) are described in table 4.1 and 4.2. Estimators were trained on 100,000 observations. Reported values are from a test set of equal size

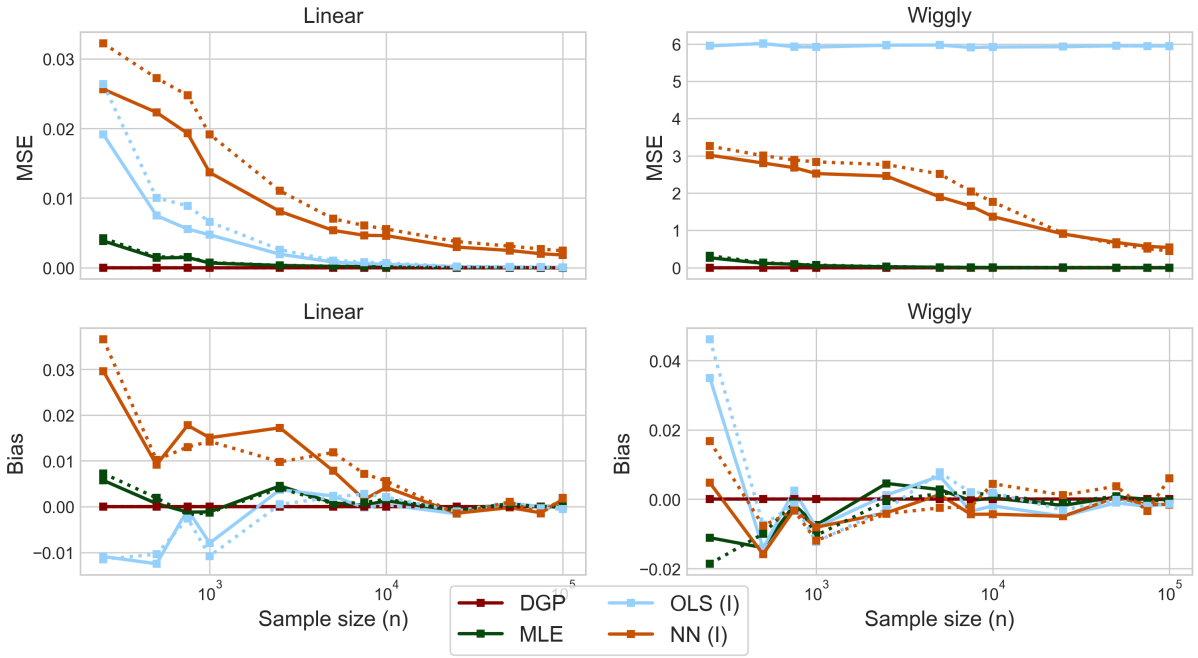
### 5.2.2. ... but networks allow for heterogeneous effects

Linear methods are worse at handling *heterogenous* effects, leading to the poor individual estimates in section 5.1. Quadratic terms and interactions does allow marginal effects to differ across observations, but does so through largely global effects. If effects exhibit less global heterogeneity, linear methods will tend to miss this. Although it does impede the estimation of average effects, it may mean that average effects are not very meaningful.

Figure 5.7 shows the distribution of marginal effects of a single regressor in two scenarios. In the first, the true DGP is linear, and all observations have homogeneous marginal effects, which all estimators are nicely centered on, although linear methods provide better estimates. The second scenario presents a more complex scenario, with truly heterogeneous effects, since some observations respond positively, and some negatively. Neural networks are capable of representing this, and their marginal effects follow distributions which look like the DGP, although with considerable noise.

The linear methods do what linear methods do, and offer the average effects for everyone. Adding quadratic terms allow some difference, but it still provides a unimodal distribution of effects. In this case, this is highly misleading. The average marginal effects are zero, which could lead us to believe that there is no effect of the variable. However, in reality there are clear marginal effects, they are just symmetric around zero. This is an extreme case, where the average is very misleading. However, even for smaller non-linearities, a focus on the average may miss the actual effects. In such cases, neural networks offer obvious possibilities neglected with linear methods.

**Figure 5.8:** *Irrelevant variables adds noise, but not bias.*




*Notes*: The upper figures show MSE between estimated and true marginal effects for a single regressor. The lower figures shows the corresponding average difference (i.e. bias). Each figure represents a simulation study for the given DGP with 1 regressors, 1 irrelevant variables and continuous output. 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 4.1 and 4.2. Solid lines show models trained only on true regressors. Dotted lines show models trained on data adding irrelevant variables.

## 5.3. Statistical inference is possible by the bootstraps

There are two key questions applied econometricians often ask (after ensuring identification): Is there an effect, and how strong is it? Above, I implicitly assumed all variables were relevant, and neural networks then provided consistent estimates of effect sizes. In practice, some variables may not be relevant for the true model. Fortunately, irrelevant variables only tend to reduce efficiency, but does not interfere with consistency. Figure 5.8 presents a straightforward extension of the simulation network where models are trained using regressors $\tilde{\boldsymbol{x}}_i = \begin{pmatrix} \boldsymbol{x}_i & \boldsymbol{v}_i \end{pmatrix}$, adding a set of irrelevant $\boldsymbol{v}_i$ not used to generate data so $\boldsymbol{v}_i \perp\!\!\!\perp y_i | \boldsymbol{x}_i$. There is no theoretical issue, as $\boldsymbol{x}_i$ remains unconfounded. The upper figures shows the MSE for estimates of the effect of a true regressor. Adding an irrelevant variable (dotted lines) confuses the model somewhat, but the error still goes to zero. The lower set of figures shows that the irrelevant variable does not add bias, so the interfere is mainly due to a more noisy estimate.

It is nice that we can still estimate actual effects, but what about the irrelevant variable? Can we also figure out that we should discard that one? Table 5.3 suggests that this is possible in expectation. The average estimates across simulations tend to be zero for the irrelevant $v_1$, and non-zero for $x_1$. However, there is a large degree of variation in estimates. The brackets below the average show a 95 pct. bound of the found estimates. These typically range over values which might indicate actual effects, often on both sides of the zero. Fortunately, econometricians are well-used to handling sampling variation of this sort.

**Table 5.3:** *Irrelevant regressors yield zero effects, but there is sampling variation*

| | **MLE** | | **OLS (I)** | | **NN (I)** | |
|---|---|---|---|---|---|---|
| | $v_1$ | $x_1$ | $v_1$ | $x_1$ | $v_1$ | $x_1$ |
| **Linear** | -0.00 | 1.71 | -0.00 | 1.71 | 0.03 | 1.67 |
| | [-0.04,0.03] | [1.67,1.76] | [-0.06,0.08] | [1.64,1.78] | [-0.03,0.11] | [1.60,1.75] |
| **Polynomial (2)** | 0.00 | -3.31 | -0.03 | -3.27 | 0.02 | -3.29 |
| | [-0.06,0.08] | [-3.51,-3.02] | [-0.23,0.19] | [-3.68,-2.89] | [-0.06,0.12] | [-3.56,-3.03] |
| **Polynomial (3)** | -0.00 | -9.95 | -0.02 | -9.89 | 0.03 | -9.75 |
| | [-0.05,0.06] | [-10.36,-9.38] | [-0.46,0.34] | [-10.57,-9.25] | [-0.14,0.16] | [-10.30,-9.14] |
| **Pointy** | -2.21 | 0.92 | -0.04 | -2.04 | -0.01 | -2.10 |
| | [-2.80,-0.03] | [-2.31,1.85] | [-0.39,0.34] | [-2.56,-1.43] | [-0.08,0.07] | [-2.52,-1.60] |
| **Wiggly** | -0.00 | 1.30 | -0.00 | 1.31 | -0.00 | 1.30 |
| | [-0.04,0.05] | [1.17,1.46] | [-0.12,0.09] | [1.14,1.51] | [-0.05,0.06] | [1.19,1.47] |
| **Trig. pol (3)** | -0.00 | 1.30 | -0.00 | 1.31 | 0.00 | 1.32 |
| | [-0.06,0.06] | [1.20,1.40] | [-0.10,0.10] | [1.16,1.45] | [-0.05,0.10] | [1.19,1.44] |
| **Ackley** | -0.01 | -1.08 | 0.02 | -1.21 | 0.02 | -1.22 |
| | [-0.17,0.15] | [-1.37,-0.37] | [-0.20,0.27] | [-1.45,-0.96] | [-0.20,0.27] | [-1.51,-0.92] |
| **Rastrigin** | 0.00 | -0.09 | 0.00 | -0.08 | 0.00 | -0.08 |
| | [-0.06,0.06] | [-0.77,1.02] | [-0.17,0.16] | [-0.24,0.07] | [-0.17,0.17] | [-0.27,0.11] |
| **Drop-Wave** | -0.05 | -0.54 | 0.00 | -0.16 | 0.01 | -0.14 |
| | [-0.49,0.09] | [-0.93,0.23] | [-0.14,0.14] | [-0.31,-0.02] | [-0.08,0.07] | [-0.40,0.19] |

*Notes:* The table shows estimated average marginal effects of the given regressor. Each row represents 100 simulations with 1 regressors, 1 irrelevant variables and continuous output. Cells show averages across simulations. Brackets show a 95 pct. confidence interval of this across the simulations. DGPs (rows) and estimators (columns) are described in table 4.1 and 4.2. Estimators were trained on 1,000 observations. Reported values are from a test set of equal size.

#### 5.3.1. Introducing the bootstrapper

The standard way to account for variation in estimates is by using analytically derived distributions for the estimator under a null hypothesis, typically based on asymptotic theory leading to normal limit distributions. This venue is not available for me, since I did not derive properties. However, I would argue that it is not necessarily fitting. Such results typically account only for *sampling variation*, where a different sample might change the estimates. However, neural networks introduce another sort of variation, since their estimates are generally volatile due to the challenges of finding optimal networks outlined in section 3.2.2. This both increases the influence of sampling variation (since neural networks may overfit the data), and adds *starting value variation* due to optimization issues (e.g. ending up in different local minima), where the initial conditions for the optimizer may affect the estimates.

While such concerns may be difficult to handle analytically, another tool incorporates them quite easily: *The bootstrap* (Efron, 1979, 1982).[45] Simple (nonparametric) bootstrapping simply assume that the empirical distribution of the data reflects the true DGP, and obtains fictitious

[45]For an introduction to bootstrapping methods, see ch. 11 in Cameron & Trivedi (2005).

samples of the same size by sampling from the empirical distribution with replacement. Models are then reestimated on these samples, and the spread of estimates may be observed. This will account not just for sampling variation but, given that I reestimate models with different starting values each time, also for the various solutions the neural network may obtain.

One concern may be computation feasibility. Single neural networks may already take a long time to train for large, high-dimensional data. Bootstrapping would require training a neural network of equal size for every iteration. However, do note that one should really only test inference once, after choosing the optimal model. Choosing models based on significance is bad practice, and leads to spurious results. Therefore it won't really be a problem the computer needs to think on it overnight a single time. Given the computation power typically available to deep learning practitioners, this is unlikely to be a serious problem.

I will redefine the bootstrap slightly for our purposes. Particularly, the bootstrap is typically used for parametric estimators, where we consider the difference in a given parameter estimate $\hat{\beta}$. However, it is unwise to use it in this manner, as there is no guarantee a particular coefficient is used similarly in different neural networks. Instead, I will bootstrap estimates of individual effects, which presents a small hurdle: In a given iteration, we only observe estimates for those observations which were sampled. This means that we actually observe fewer estimates than the number of iterations for every observation, which means we need more iterations for the same level of certainty. We will also observe more than one estimate for those sampled more than once within a single iteration. However, given that they are identical, effects will also be. Even though there are multiple estimates, they should only be counted once, as they provide information on just one estimated model. This presents a practical approach: For any iteration we save estimates for sampled observations once, and save nothing for those not sampled.

**Definition 5.2.** The *bootstrap replications* $\hat{\boldsymbol{\theta}}^*$ of estimates $\hat{\theta}_i$ for observations $i = 1, \dots, n$ with data $\boldsymbol{w}_i = (y_i, \boldsymbol{x}_i)$ is obtained by, for $b = 1, \dots, B$:

1. Draw (with replacement) a bootstrap sample of size $n$: $\boldsymbol{w}^{(b)} = \left(\boldsymbol{w}_1^{(b)}, \dots, \boldsymbol{w}_n^{(b)}\right)'$.
2. Estimate (retrain) statistical models based on $\boldsymbol{w}^{(b)}$.
3. Calculate $\hat{\theta}_{i,j}^{(b)}$ for bootstrap observations $j = 1, ..., n$.
4. Save bootstrap estimates $\hat{\boldsymbol{\theta}}^{(b)} = \left(\hat{\theta}_i^{(b)}, \dots, \hat{\theta}_n^{(b)}\right)'$ with $\hat{\theta}_i^{(b)}$ equal to any of the bootstrap estimates $\hat{\theta}_{i,j}^{(b)}$ if $i$ was sampled (e.g., pick the first one), and a 'missing value' if not.

Collect the estimates in a $n \times B$ matrix of bootstrap replications $\hat{\boldsymbol{\theta}}^* = \left(\hat{\boldsymbol{\theta}}^{(1)}, \dots, \hat{\boldsymbol{\theta}}^{(B)}\right)$.

#### 5.3.2. Standard errors, tests and confidence intervals

Relevant statistics can be obtained directly from the matrix of replications. Standard errors for estimates of average effects can be obtained by calculating the average effect within each replication, and then calculating the standard error of these.[46] Table 5.4 compares the average of such estimates with the actual sampling variation obtained across simulations. The bootstrapper is able to reflect the underlying variation quite well, particularly for the simpler processes and the irrelevant variable. However, in several cases the variation of the neural network seems somewhat volatile. More bootstrap replications would likely alleviate this.

[46] I will focus on average effects, but one could also consider the variation in individual effects.

**Table 5.4:** *Bootstrap can estimate standard errors, but may need more replications*

| | **MLE** | | **OLS (I)** | | **NN (I)** | |
|---|---|---|---|---|---|---|
| | $v_1$ | $x_1$ | $v_1$ | $x_1$ | $v_1$ | $x_1$ |
| **Linear** | 0.019 | 0.025 | 0.040 | 0.040 | 0.044 | 0.044 |
| | (0.019) | (0.023) | (0.037) | (0.038) | (0.041) | (0.042) |
| **Polynomial (2)** | 0.035 | 0.098 | 0.112 | 0.195 | 0.049 | 0.106 |
| | (0.035) | (0.137) | (0.104) | (0.187) | (0.045) | (0.141) |
| **Polynomial (3)** | 0.032 | 0.190 | 0.176 | 0.386 | 0.086 | 0.260 |
| | (0.029) | (0.270) | (0.185) | (0.361) | (0.077) | (0.317) |
| **Pointy** | 0.650 | 0.901 | 0.198 | 0.335 | 0.040 | 0.200 |
| | (0.714) | (1.032) | (0.178) | (0.304) | (0.038) | (0.287) |
| **Wiggly** | 0.028 | 0.063 | 0.052 | 0.095 | 0.035 | 0.061 |
| | (0.027) | (0.080) | (0.052) | (0.094) | (0.031) | (0.079) |
| **Trig. pol (3)** | 0.035 | 0.047 | 0.051 | 0.066 | 0.043 | 0.054 |
| | (0.034) | (0.050) | (0.049) | (0.075) | (0.042) | (0.061) |
| **Ackley** | ... | 0.529 | 0.137 | 0.141 | 0.137 | 0.150 |
| | (0.093) | (0.282) | (0.135) | (0.136) | (0.134) | (0.146) |
| **Rastrigin** | 0.033 | 0.330 | 0.096 | 0.095 | 0.098 | 0.107 |
| | (0.032) | (0.438) | (0.088) | (0.086) | (0.086) | (0.094) |
| **Drop-Wave** | ... | ... | 0.072 | 0.073 | 0.043 | 0.151 |
| | (0.497) | (0.926) | (0.077) | (0.076) | (0.042) | (0.166) |

*Notes:* The table shows standard errors of estimates of the average marginal effect of the given regressor. Each row represents 100 simulations with 1 regressors, 1 irrelevant variables and continuous output. Cells show average estimated standard error based on 99 bootstrap replications. Parentheses show the standard error of estimates across simulations. DGPs (rows) and estimators (columns) are described in table 4.1 and 4.2. Estimators were trained on 1,000 observations. Reported values are from a test set of equal size.

Econometricians often consider a simpler question: Is there an effect at all? Hypothesis tests against a null of zero effect can easily be done based on the bootstrap. I prefer to do so using the *percentile method* where the null is rejected (on an $\alpha$ significance level) if it falls outside lower and upper $\alpha/2$ quantiles of the bootstrap distribution (Cameron & Trivedi, 2005, 364).[47] Table 5.5 shows the share of simulations where the null was rejected on a 95 pct. significance level. Notice that for neural networks, the null is rejected for the true regressor 100 pct. of the time in all scenarios except the two most complicated ones (which, as seen above, probably cannot be estimated with this small a sample). For the irrelevant regressor, the null is rejected in less than 10 pct. of the cases, which is around the error level we would expect.

Econometricians may like tests too much. In recent years, there has been valid criticism of the reliance on tests and p-values (Aschwanden, 2015; Nuzzo, 2014)[48]. However, the criticism has

[47] The alternative is to assume a limit normal distribution, and perform typical t-tests. I prefer the percentile methods because it is assymmetric, and does not rely on assuming a normal distribution.

[48] In short, the criticism argues that p-values are a poor measure of the strength of hypotheses even if used correctly. Furthermore, the focus on them by journals leads to *p-hacking*, where practitioners try many approaches and report those which (by chance) clear the 5 pct. mark, making the statistics misleading.

**Table 5.5:** *Tests and confidence interval work just fine even with few observations*

| | **MLE** | | **OLS (I)** | | **NN (I)** | |
|---|---|---|---|---|---|---|
| | $v_1$ | $x_1$ | $v_1$ | $x_1$ | $v_1$ | $x_1$ |
| **Linear** | 0.06 | 1.00 | 0.05 | 1.00 | 0.07 | 1.00 |
| | [-0.04,0.04] | [1.67,1.76] | [-0.08,0.08] | [1.64,1.79] | [-0.05,0.11] | [1.59,1.76] |
| **Polynomial (2)** | 0.08 | 1.00 | 0.06 | 1.00 | 0.09 | 1.00 |
| | [-0.06,0.07] | [-3.49,-3.12] | [-0.24,0.18] | [-3.65,-2.91] | [-0.08,0.11] | [-3.49,-3.09] |
| **Polynomial (3)** | 0.04 | 1.00 | 0.09 | 1.00 | 0.07 | 1.00 |
| | [-0.06,0.06] | [-10.31,-9.59] | [-0.36,0.31] | [-10.63,-9.17] | [-0.16,0.17] | [-10.30,-9.30] |
| **Pointy** | 0.32 | 0.02 | 0.02 | 1.00 | 0.04 | 1.00 |
| | [...] | [...] | [-0.42,0.33] | [-2.67,-1.41] | [-0.08,0.07] | [-2.47,-1.71] |
| **Wiggly** | 0.05 | 1.00 | 0.10 | 1.00 | 0.05 | 1.00 |
| | [-0.05,0.05] | [1.18,1.42] | [-0.10,0.09] | [1.13,1.49] | [-0.07,0.06] | [1.19,1.42] |
| **Trig. pol (3)** | 0.04 | 1.00 | 0.05 | 1.00 | 0.04 | 1.00 |
| | [-0.07,0.07] | [1.21,1.39] | [-0.10,0.09] | [1.18,1.43] | [-0.08,0.08] | [1.22,1.42] |
| **Ackley** | 0.00 | 0.94 | 0.04 | 1.00 | 0.06 | 1.00 |
| | [-0.17,0.16] | [-1.59,-0.34] | [-0.23,0.28] | [-1.48,-0.94] | [-0.23,0.29] | [-1.51,-0.94] |
| **Rastrigin** | 0.08 | 0.17 | 0.08 | 0.10 | 0.07 | 0.12 |
| | [-0.06,0.07] | [-0.72,0.53] | [-0.18,0.18] | [-0.26,0.10] | [-0.18,0.19] | [-0.28,0.13] |
| **Drop-Wave** | 0.00 | 0.12 | 0.09 | 0.60 | 0.07 | 0.18 |
| | [-0.46,0.90] | [-1.18,0.73] | [-0.13,0.14] | [-0.30,-0.02] | [-0.08,0.08] | [-0.41,0.16] |

*Notes:* The table shows the results of hypothesis tests and confidence intervals of the average marginal effect of the given regressor. Each row represents 100 simulations with 1 regressors, 1 irrelevant variables and continuous output. Cells show the share of simulations where a null hypothesis of $\gamma = 0$ was rejected on a 95 pct. significance level. Brackets show 95 pct. confidence interval through the average upper and lower quantiles (across simulations). Both rely on a percentile method based on 99 bootstrap replications. DGPs (rows) and estimators (columns) are described in table 4.1 and 4.2. Estimators were trained on 1,000 observations. Reported values are from a test set of equal size.

been mainly aimed at our approach to statistical inference, not the basic goal of quantifying the robustness of our results. It just turns out that to do so, we need more than one largely arbitrary measure for whether an effect is likely. Authors have instead argued for an increased focus on practical effect sizes and confidence intervals (see e.g. Cumming (2014)). Fortunately, bootstrapping allows for basically any measure one can think of. Confidence intervals, for instance, are easily derived by simply reporting the upper and lower $\alpha/2$ quantiles of the bootstrap replications (Cameron & Trivedi, 2005, 365). Table 5.5 shows the average of these bootstrapped confidence intervals estimates. These are very similar to actual confidence bounds on sampling variation obtained in table 5.3, which indicates that I obtain good estimates of the relevant parts of the sampling variation. Thus, bootstrapping should allow robustness checks.

This finalises my basic review of the properties of neural networks as an estimator. They appear to provide a well-behaved estimator for individual marginal effects in the face of arbitrary non-linearity, which allows analysis of fully heterogeneous effects. Adding in the bootstrapping procedure validated here, we have all the tools we need for analysis.

## 6. Results II: ... but are invalidated (and saved) by the usual stuff

'Garbage in, garbage out'.

– *Scientific proverb*

Econometricians have spent the last century arguing that the world is not as rosy as assumed above, and meticulously deriving the implications for various estimators. However, many of these issues are really theoretical issues, since they imply violations of the assumptions for causal estimation (as reviewed in section 2.2.1 and 2.3.1), and their basic implications do not depend on the estimator at hand. I will show this by recreating two archetypical types of bias. First, I implement random measurement error on the regressors, and show this leads to attenuation bias for all included estimators (including neural networks). Second, I implement confounding variables, and show this leads to omitted variable bias. Fortunately, many of the solutions to such issues are similarly theoretically motivated. I illustrate this by adapting an instrumental variables (IV) approach to neural networks, and showing this yields a consistent estimator. This leads to an estimator which is actually useful in practice, which I show by replicating a few results on the returns to schooling from a well-known study by Angrist & Krueger (1991).

### 6.1. Measurement error leads to attenuation bias

Measurement errors is a classical topic in econometrics, likely because one can often expect to encounter it in practice.[49] The consequences are particularly well-studied for linear methods, because the implications are easy to derive. The classical case here is measurement error in $\boldsymbol{x}_i$ which is correlated with the regressors themselves, because this leads to inconsistency in the form of *attenuation bias*, where estimates are biased towards zero.[50] General properties are harder to derive for non-linear methods, leading many researchers to focus on properties of specific estimators (Cameron & Trivedi, 2005). However, the intuition for attenuation bias does not really depend on linearity. The key point is that an algorithm observes changes in $\boldsymbol{x}_i$ (caused by measurement error), which does not lead to changes in $y_i$, which is considered evidence that $y_i$ does not depend on $\boldsymbol{x}_i$, moving estimates towards zero. If errors depend on $\boldsymbol{x}_i$ they do not cancel out, leading to bias. Therefore, it is plausible that similar bias plagues neural networks.

I may investigate this with-in my simulation set-up. Assume that true regressors $\boldsymbol{x}_i^*$ exists, which is drawn just as the regressors $\boldsymbol{x}_i$ were in the clean set-up. Instead of observing the true regressor, I now observe a set of regressors with measurement error $\boldsymbol{e}$:

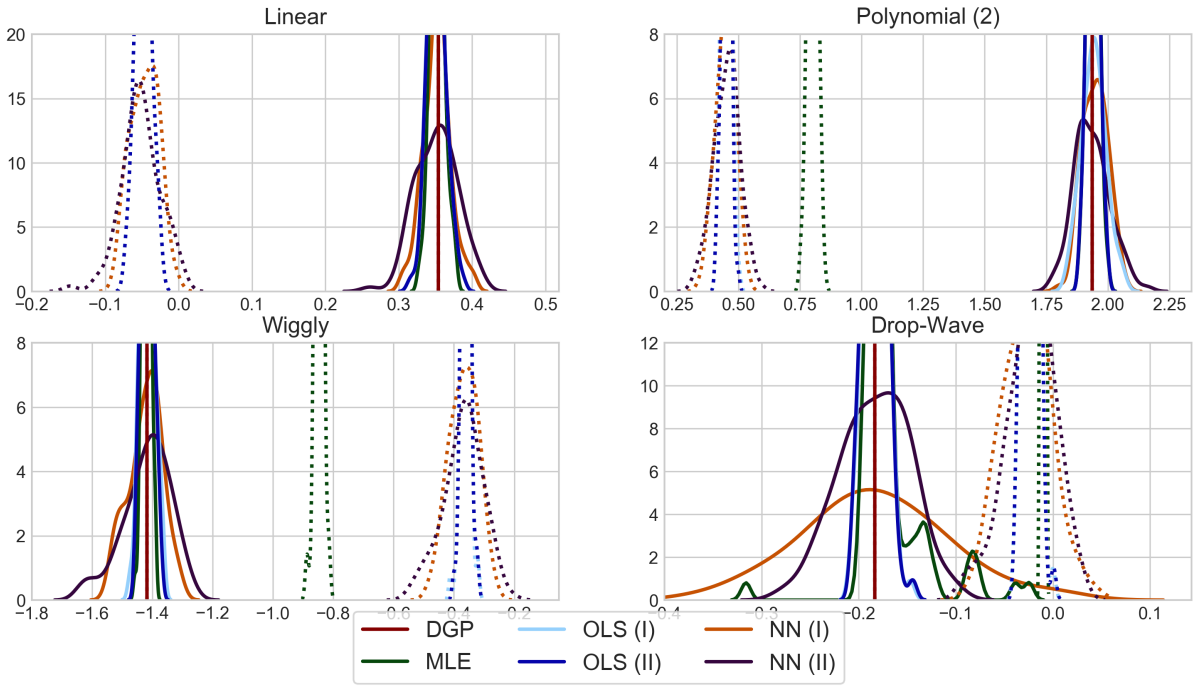
$$\boldsymbol{x}_i = \boldsymbol{x}_i^* + \boldsymbol{e}_i, \qquad \boldsymbol{e}_i \sim N\left(\boldsymbol{0}, \boldsymbol{\Omega}_{\boldsymbol{e}}\right), \quad \boldsymbol{\Omega}_{\boldsymbol{e}} \neq \boldsymbol{I}_k. \tag{6.1}$$

This fulfils the assumptions of a classical measurement error model (as seen in e.g. Cameron & Trivedi (2005)). Errors are mean zero and independent from $\boldsymbol{x}_i^*$ and $u_i$ so $\mathrm{E}[\boldsymbol{e}_i \,|\, \boldsymbol{x}_i^*] = \mathrm{E}[\boldsymbol{e}_i \,|u_i] = \boldsymbol{0}$. This naturally leads to correlation between $\boldsymbol{x}_i$ and $\boldsymbol{e}_i$. Furthermore, the covariance is designed so errors are correlated $\mathrm{Cov}(e_{i,p}, e_{i,j}) \neq 0$ leading to correlation between errors and other regressors as well $\mathrm{Cov}(x_{i,p}, e_{i,j}) \neq 0$.

[49] Cameron & Trivedi (2005) provides a review. For further information, see Wansbeek (2000) or Fuller (1987).

[50] The alternative is measurement error in either $y_i$ or $\boldsymbol{x}_i$ which is uncorrelated with $\boldsymbol{x}_i$. In both cases, such errors can be absorbed by the error term and therefore only reduces efficiency. However, implications may be different for non-linear cases (Cameron & Trivedi, 2005).

**Figure 6.1:** *Measurement error shifts average estimates towards zero*




*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 and continuous output. DGPs (panels) and estimators (lines) are described in table 4.1 and 4.2. Estimators were trained on 100,000 observations. Reported values are from a test set of equal size. Solid lines show models trained on actual data. Dotted lines show models trained on data with measurement error in regressors.

### 6.1.1. Yep, attenuation bias...

The induced bias can be seen clearly by the estimates of average effects obtained by the various estimators. Figure 6.1 show the distribution of such across simulations in four scenarios. Solid lines show models estimated on the actual data, while dotted lines show the results from training models on data with measurement error. These are all centered around different values than the true effect, showing clear bias. In all cases, the bias moves the estimates closer to zero, as predicted by attenuation bias. However, in some cases (e.g. the linear one shown), the new estimates appear to be pushed past zero. Rather than merely converging on a weaker effect, I observe an opposite effect. This may be an effect of the correlation between errors across regressors, so that errors disturb more than their own regressor. However, note that in both cases where this occurs seen here, the estimates are centered on values quite close to zero. They may therefore also be caused simply by very strong attenuation, leading to an estimated zero effect, and then sample quirks may account for the rest. However, some other cases I investigated suggested a small bias may arise in the opposite direction for some scenarios. All in all, the results suggests attenuation bias, although the estimates are possibly even further biased.

The individual estimates are likely to be biased in the same way. To see this I will consider a specific attenuation measure, since MSE does not provide that much information for a given sample. I could observe that measurement error increases the MSE, but this is largely a given since any increased volatility will increase variance. One cannot directly observe whether bias

**Table 6.1:** *Attenuation factors indicate measurement error biases towards zero*

| | MLE | | OLS (I) | | OLS (II) | | NN (I) | | NN (II) | |
|---|---|---|---|---|---|---|---|---|---|---|
| | **Act.** | **Obs.** | **Act.** | **Obs.** | **Act.** | **Obs.** | **Act.** | **Obs.** | **Act.** | **Obs.** |
| **Linear** | 1.00 | 0.28 | 1.00 | 0.52 | 1.00 | 0.52 | 1.00 | 0.52 | 1.00 | 0.52 |
| | (0.006) | (0.006) | (0.006) | (0.005) | (0.006) | (0.005) | (0.011) | (0.012) | (0.017) | (0.017) |
| **Polynomial (2)** | 1.00 | 0.87 | 0.80 | 0.27 | 1.00 | 0.61 | 0.82 | 0.52 | 1.24 | 0.58 |
| | (0.016) | (3.608) | (2.772) | (0.645) | (0.033) | (1.988) | (2.269) | (1.398) | (1.404) | (2.171) |
| **Polynomial (3)** | 1.00 | 0.78 | 2.33 | 0.68 | 0.74 | 0.57 | 1.26 | 0.52 | 1.17 | 0.41 |
| | (0.024) | (1.853) | (7.353) | (3.300) | (2.919) | (1.863) | (2.069) | (2.456) | (1.719) | (1.656) |
| **Pointy** | 0.76 | 0.60 | 0.76 | 0.67 | 0.83 | 0.50 | 1.20 | 1.33 | 1.14 | 1.16 |
| | (1.842) | (3.554) | (7.843) | (3.748) | (2.716) | (6.623) | (1.251) | (3.832) | (1.285) | (3.097) |
| **Wiggly** | 1.00 | 0.11 | 0.31 | 0.25 | 0.28 | 0.23 | 1.02 | 0.18 | 0.71 | 0.18 |
| | (0.032) | (1.829) | (2.041) | (1.654) | (2.604) | (1.385) | (1.989) | (1.064) | (1.779) | (1.295) |
| **Trig. pol (3)** | 0.99 | 1.07 | 0.33 | 0.15 | 0.70 | 0.42 | 1.16 | 0.48 | 1.02 | 0.40 |
| | (0.059) | (5.435) | (2.240) | (0.988) | (1.891) | (1.236) | (0.997) | (1.609) | (1.159) | (1.474) |
| **Ackley** | 0.13 | 0.01 | 0.07 | 0.02 | 0.03 | -0.04 | 1.13 | -0.02 | 0.47 | -0.08 |
| | (0.748) | (0.508) | (1.156) | (0.388) | (0.797) | (0.745) | (3.219) | (0.812) | (2.141) | (2.565) |
| **Rastrigin** | 1.00 | -0.04 | 0.02 | 0.01 | -0.01 | -0.02 | 0.98 | -0.00 | 0.28 | 1.22 |
| | (0.022) | (0.294) | (0.169) | (0.094) | (0.170) | (0.269) | (3.224) | (0.260) | (3.416) | (5.547) |
| **Drop-Wave** | ... | -0.00 | -0.04 | 0.01 | 0.22 | 0.09 | 0.96 | 0.22 | 1.04 | 0.67 |
| | (–) | (0.940) | (1.819) | (0.567) | (5.273) | (0.979) | (9.492) | (1.956) | (2.832) | (5.355) |

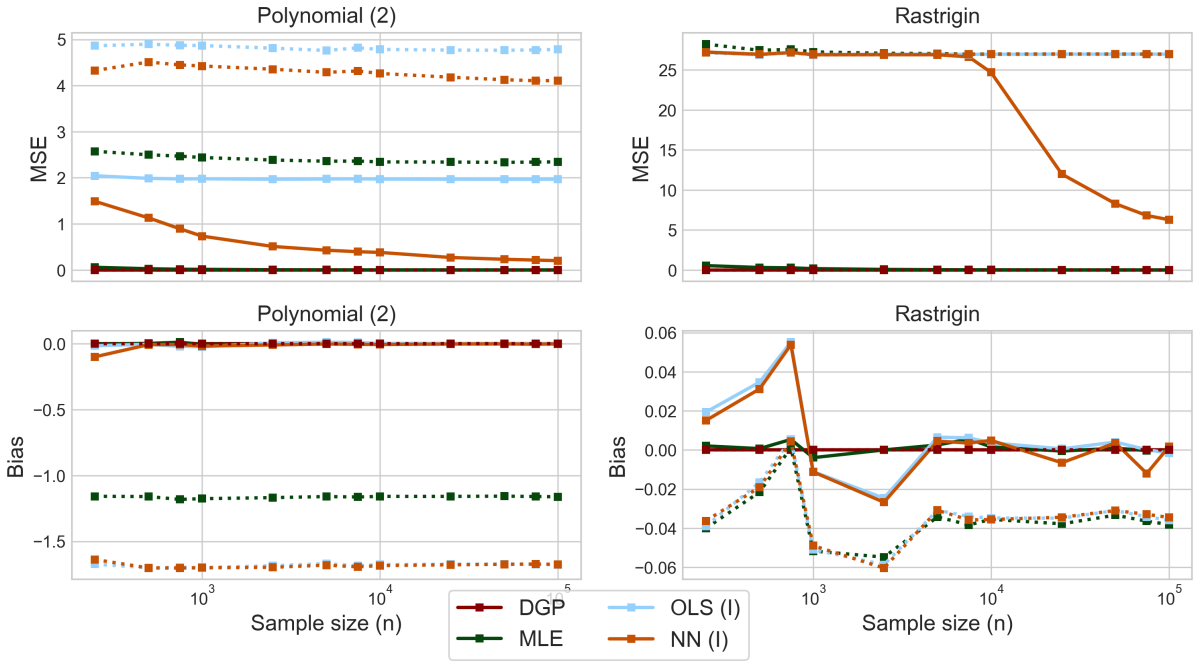
*Notes:* The table shows estimated attenuation factors between estimated and true marginal effects, averaged across all regressors. Each row represents 100 simulations with 5 regressors and continuous output. Cells show averages across simulations, with standard errors in parentheses. DGPs (rows) and estimators (columns) are described in table 4.1 and 4.2. Estimators were trained on 100,000 observations. Reported values are from a test set of equal size. Actual models are trained on the true regressors. Observed models are trained on data with measurement error.

is also induced. Instead, I will consider a improvised measure of the *attenuation factor* $\lambda$. In courses on OLS, one often shows that $\text{plim}\hat{\beta} = \lambda\beta$ where $\lambda \in (0, 1)$ (see e.g. Pischke (2007)). Any attenuation bias will function similarly, leading to a natural estimate: if estimates of marginal effects tend to be numerically smaller than the true values, then $\lambda = \hat{\gamma}(\boldsymbol{x}_i)/\gamma(\boldsymbol{x}_i)$ should tend to be between zero and one. If, on the other hand, it tends to be one, the estimates are unbiased. Averaging across the sample and the simulations provides an estimate of the expectation of this, allowing me to evaluate whether estimates are attenuated.[51]

Table 6.1 provides estimates of the attenuation factor. Consider first the linear case. In all 'actual' models, the mean attenuation for estimates of individual marginal effects is one, corresponding to no bias. In the observed case with measurement error, the factor is about a half for both linear and neural methods, indicating significant attenuation. The overall picture from the non-linear scenarios is similar. In all 'actual' cases where the MLE converges properly it obtains a $\lambda \approx 1$, and so does the neural network (although somewhat more noisely). The linear methods are more difficult to judge, because they also exhibit different bias. In the observed cases on the other hand, the estimated factor is lower, indicating that estimates are biased towards zero.

[51] The measure is not perfect. Estimates with the correct sign will pull the estimate in the right direction, but any estimates with opposite sign will tend to reduce it. Therefore, it more accuractely measures whether coefficients are biased in the same direction as zero, rather than towards zero itself.

**Figure 6.2:** *Attenuation bias leads to inconsistency*




*Notes*: The upper figures show MSE between estimated and true marginal effects for a single regressor. The lower figures shows the corresponding average difference (i.e. bias). Each figure represents a simulation study for the given DGP with 3 regressors and continuous output. 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 4.1 and 4.2. Solid lines show models trained on actual data. Dotted lines show models trained on data with measurement error in regressors.

### 6.1.2. ... leading to inconsistency

Attenuation bias is an asymptotic issue, and therefore implies inconsistency. Figure 6.2 shows this for two scenarios. The top panels show the MSE for estimates of individual marginal effects for various sample sizes.[52] The solid lines show models trained on the true regressors, where the neural network provides a nicely consistent estimator. The dotted lines instead show models trained on erroneous data, and there is no observable trend towards zero. Other scenarios present similar pictures. This indicates that individual effects are inconsistent.

The lower set of figures show directly that this occurs because the error models exhibit bias. The bias is again estimated by taking the average difference between estimates and true values. For actual models, the bias converges to zero for both scenarios, whereas it converges to a different level for the error models. I cannot directly observe whether the bias attenuates (i.e. induces a bias of opposite sign, which moves the estimate closer to zeron), but the section above indicated this. Hence, I conclude that attenuation bias is a general affliction which plagues neural networks with measurement error. Fortunately, attenuation does not irritate econometricians that much because it is a conservative error: It tends to encourage null results. Therefore, if we observe a strong effect, we may still trust it. The next section provides a graver danger.

[52] MSE is the fitting measure here, since proposition 5.1 uses it to establish consistency. Even if measurement error increase variance for a given sample, it should still go to zero asymptotically for consistency.

**Table 6.2:** *RMSE increases for a single regressor when it is confounded*

| | MLE | | OLS (I) | | OLS (II) | | NN (I) | | NN (II) | |
|---|---|---|---|---|---|---|---|---|---|---|
| | **Act.** | **Obs.** | **Act.** | **Obs.** | **Act.** | **Obs.** | **Act.** | **Obs.** | **Act.** | **Obs.** |
| **Linear** | 0.01 | 1.55 | 0.01 | 1.14 | 0.02 | 1.14 | 0.08 | 1.15 | 0.10 | 1.15 |
| | (0.005) | (0.016) | (0.004) | (0.010) | (0.006) | (0.010) | (0.015) | (0.036) | (0.017) | (0.053) |
| **Polynomial (2)** | 0.02 | 6.88 | 1.81 | 3.15 | 0.04 | 4.32 | 0.55 | 4.36 | 0.50 | 4.38 |
| | (0.009) | (0.030) | (0.004) | (0.023) | (0.012) | (0.028) | (0.041) | (0.054) | (0.043) | (0.104) |
| **Polynomial (3)** | 0.02 | 4.02 | 15.29 | 15.37 | 6.58 | 7.31 | 3.79 | 3.98 | 3.17 | 4.03 |
| | (0.008) | (0.117) | (0.061) | (0.066) | (0.040) | (0.067) | (0.461) | (0.159) | (0.364) | (0.214) |
| **Pointy** | ... | 18.40 | 12.99 | 17.91 | 12.98 | 17.96 | 3.61 | 13.08 | 2.98 | 12.76 |
| | (–) | (7.721) | (0.032) | (0.069) | (0.033) | (0.071) | (0.759) | (0.276) | (0.385) | (0.569) |
| **Wiggly** | 0.03 | 2.26 | 4.20 | 4.63 | 4.19 | 4.64 | 1.87 | 2.46 | 1.40 | 2.28 |
| | (0.012) | (0.021) | (0.011) | (0.016) | (0.011) | (0.018) | (0.113) | (0.114) | (0.099) | (0.098) |
| **Trig. pol (3)** | 0.02 | 1.92 | 0.61 | 1.27 | 0.26 | 1.66 | 0.19 | 1.80 | 0.21 | 1.81 |
| | (0.009) | (0.008) | (0.001) | (0.010) | (0.003) | (0.016) | (0.012) | (0.034) | (0.014) | (0.058) |
| **Ackley** | 1.76 | 1.85 | 1.81 | 1.81 | 1.79 | 1.80 | 1.53 | 1.21 | 1.25 | 0.97 |
| | (0.045) | (0.134) | (0.003) | (0.003) | (0.003) | (0.003) | (0.169) | (0.078) | (0.080) | (0.042) |
| **Rastrigin** | 0.04 | 0.10 | 6.71 | 6.71 | 6.71 | 6.71 | 3.19 | 3.10 | 2.77 | 2.16 |
| | (0.026) | (0.045) | (0.009) | (0.009) | (0.009) | (0.009) | (0.259) | (0.272) | (0.215) | (0.178) |
| **Drop-Wave** | 0.82 | 3.84 | 3.72 | 3.74 | 3.63 | 3.71 | 2.45 | 3.34 | 1.03 | 3.28 |
| | (1.477) | (1.440) | (0.007) | (0.007) | (0.006) | (0.007) | (0.078) | (0.015) | (0.062) | (0.019) |

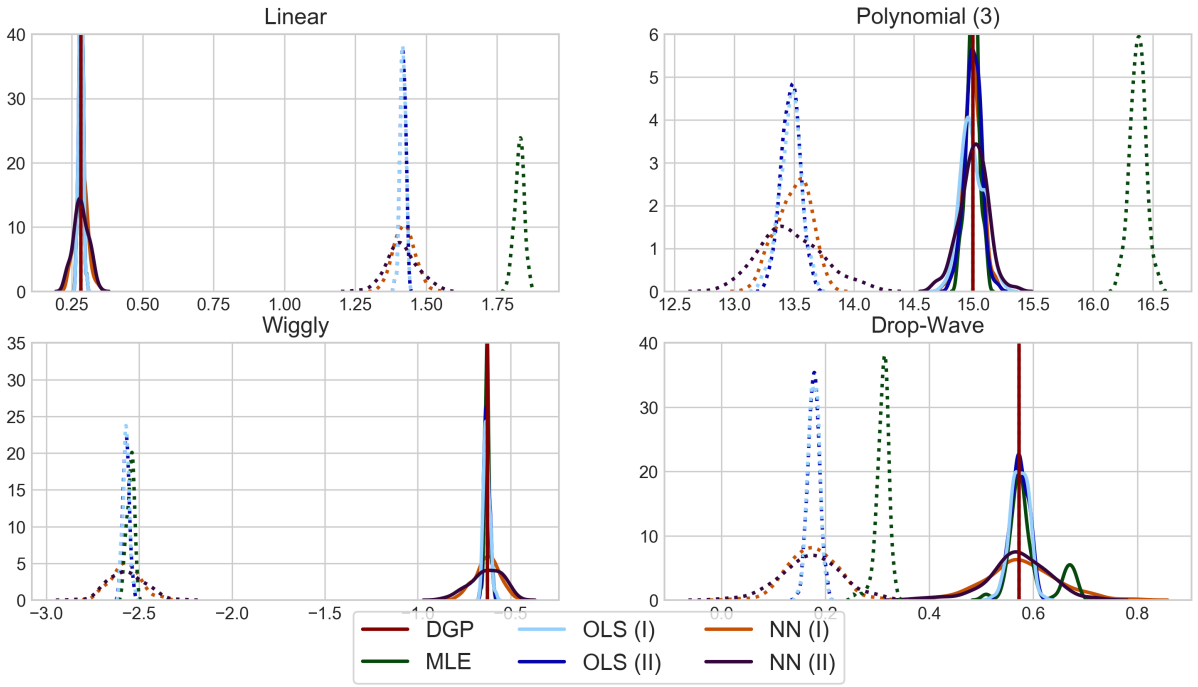
*Notes:* The table shows root MSE between estimated and true marginal effects for a single regressor. Each row represents 100 simulations with 1 regressors, 4 confounders and continuous output. Cells show averages across simulations, with standard errors in parentheses. DGPs (rows) and estimators (columns) are described in table 4.1 and 4.2. Estimators were trained on 100,000 observations. Reported values are from a test set of equal size. Actual models are trained on all variables. Observed models exclude the confounders.

### 6.2. Confounders yield omitted variable bias

Confounders arguably provide the greatest source of fear for applied econometricians, just ahead of being passed over for tenure and disgruntled sociologists. They do so for good reason, since missed confounding variables are likely one of the greatest sources of error in applied work. Most of us have derived formulas for omitted variable bias for OLS in undergraduate courses. However, the concept does not depend on the estimation procedure, but rather on the conditional expectation $\mathrm{E}[y_i|\,\boldsymbol{x}_i]$ itself. If we omit a variable which is relevant for $y_i$, then we will obtain a different expectation. If the variable is also correlated with the variable of interest $x_{i,p}$, then the observed effect of that variable will also differ from the one induced by manipulating it. Consider the archetypical example of ability bias: We exclude natural ability from estimation of the effect of education on wages, even though it is likely to be positively correlated with both. An unobserved increase (between observations) in ability would typically follow an observed increase in education, leading us to erroneously attribute this to an effect of education. Since these issues are entirely probability theoretic they should also be expected to plague neural networks.

I will show this by augmenting my simulation set-up slightly. Define the actual features used to generate the outcome of interest as $\tilde{\boldsymbol{x}}_i = \begin{pmatrix}\boldsymbol{x}_i & \boldsymbol{c}_i\end{pmatrix}$, which adds a set of *confounders* $\boldsymbol{c}_i$. These are extra variables drawn from the same distribution (so that $\tilde{\boldsymbol{x}}_i$ corresponds to the $\boldsymbol{x}_i$ used above), but unobserved in practice, where only $\boldsymbol{x}_i$ is known. This leads to two models:

**Figure 6.3:** *Estimates of average marginal effects are biased when confounded.*




*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 1 regressors, 4 confounders and continuous output. DGPs (panels) and estimators (lines) are described in table 4.1 and 4.2. Estimators were trained on 100,000 observations. Reported values are from a test set of equal size. Solid lines show models trained on actual data. Dotted lines show models trained on observable data, which excludes confounders.

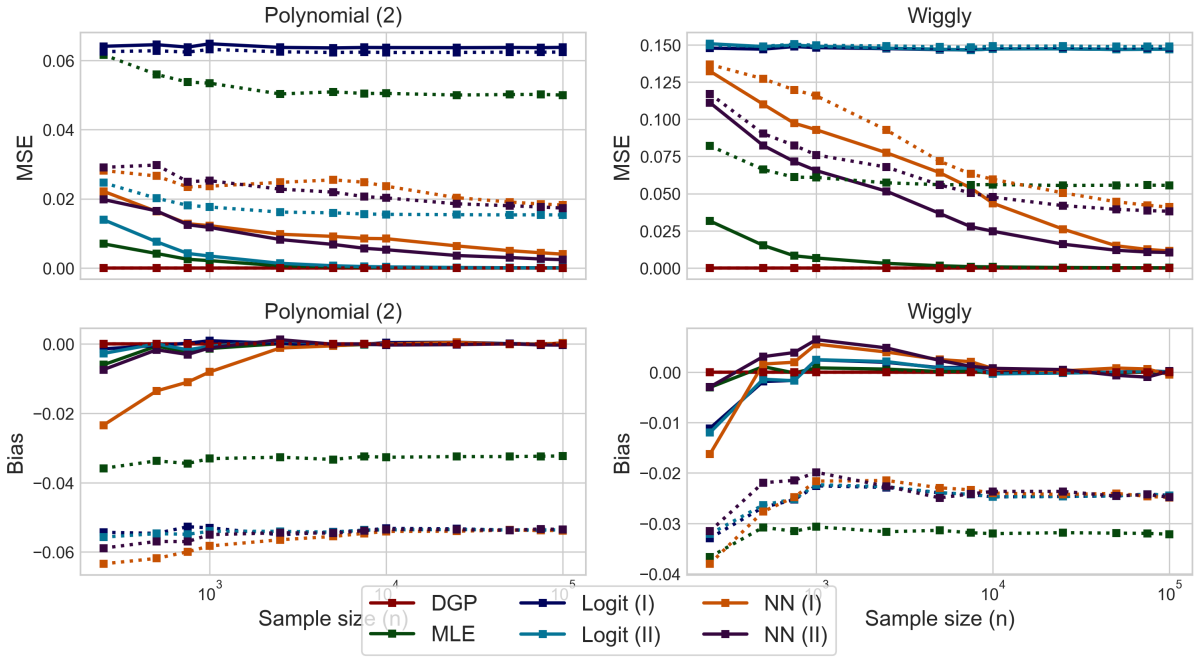
$$
\begin{aligned}
\tilde{\boldsymbol{x}}_i &= \begin{pmatrix} \boldsymbol{x}_i & \boldsymbol{c}_i \end{pmatrix} \sim \mathcal{N}(\tilde{\boldsymbol{\mu}}, \tilde{\boldsymbol{\Omega}}), \\
y_i &= g\left(\tilde{\boldsymbol{x}}_i, \tilde{\boldsymbol{\beta}}\right) + \tilde{u}_i \qquad \text{(Actual)} \\
&= g\left(\boldsymbol{x}_i, \boldsymbol{\beta}\right) + u_i. \qquad \text{(Observed)}
\end{aligned} \tag{6.2}
$$

The first specification corresponds to the clean case above, and should yield consistent estimates for causal effects. However, one would not expect the second one to do so, because $\boldsymbol{x}_i$ is confounded (see definition 2.2) by excluding $\boldsymbol{c}_i$ from the covariates. I can also state this in terms of the error term. In the actual specification, I have $\mathrm{E}[\tilde{u}_i|\tilde{\boldsymbol{x}}_i] = 0$. However, the same does not hold for the observed case in general $\mathrm{E}[u_i|\,\boldsymbol{x}_i] \neq 0$, since $\mathrm{Cov}(\boldsymbol{x}_i, \boldsymbol{c}_i) \neq 0$ and $\mathrm{Cov}(y_i, \boldsymbol{c}_i) \neq 0$. This should set the alarm bells ringing for econometricians.

Table 6.2 shows that confounders do indeed interfere with estimation. MSE provides a decent measure here, since one would not expect the volatility of estimates to increase when we estimate a simpler model by excluding variables. Therefore, increases in the MSE are likely to reflect bias. The table shows the mean root MSE for a single regressor $x_{i,p}$ observed in both cases (i.e. $x_{i,p} \in \boldsymbol{x}_i$). The result is the same for both maximum likelihood, OLS and the neural network across the observed processes: Unobservable confounders yield greater MSE in the observable case (where confounders are excluded) because the estimator becomes biased.

The effect can be seen even more clearly by focusing on average marginal effects. The distribution of estimates of the average effect of a single regressor across some of the simulations is visualized

**Figure 6.4:** *Confounders leads to bias leads to inconsistency.*




*Notes*: The upper figures show MSE between estimated and true marginal effects for a single regressor. The lower figures shows the corresponding average difference (i.e. bias). Each figure represents a simulation study for the given DGP with 1 regressors, 1 confounders and binary output. 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 4.1 and 4.2. Solid lines show models trained on actual data. Dotted lines show models trained on observable data, which excludes confounders.

in figure 6.3. The actual marginal effect of the DGP is visualized with a straight red line. As we've seen before, all estimators are able to center themselves on this value if they observe all relevant variables (solid lines). However, in the case where they do not observe confounders (dotted lines) their estimates are quite obviously biased. Indeed, it appears that estimators are biased in much the same manner,[53] which is consistent with the notation that omitted variable bias is a theoretical issue, rather than one of estimation.

Figure 6.4 shows how the effect of the confounding variables depend on the sample size for two scenarios. Other scenarios present similar pictures. The first two figures show the MSE for estimates. The solid lines show how, as shown above, neural estimates converge to the true value as the sample grows. The dotted lines indicates how omitted variable bias interferes with this, since estimators converge to an error level much greater than that obtained in the actual models.

The lower set of figures show that asymptotic bias is to blame here. The bias of the estimators (i.e. the average 'miss') clearly does not converge to zero. Rather, it converges to some other level of bias, and becomes stable there. This highlights the issue of omitted variables: The problem is not that we have less data (although this may induce noise). The problem is that the missing data biases our estimates with existing procedures. No amount of data will save us.

[53]The MLE often finds a different place to chill out. This is because my MLE does not allow an intercept, which could have offset some bias. This highlights ever further the danger of believing one knows the true DGP.

## 6.3. Fortunately, instrumental variables can still save us

Fortunately, econometricians have responded to this fear of confounders by developing methods of dealing with it. Indeed, much of the credibility revolution in microeconometrics (as reviewed in section 2.2.1) regards explicitly accounting for possible confounders. Since omitted variables are a theoretic problem, most identification strategies intended to do so can likely be extended to neural networks. I will illustrate this by adapting the *instrumental variables* (IV) approach.[54]

### 6.3.1. What can IV do?

Consider a situation akin to the section above. The variable $x_{i,p}$ has a causal interpretation in the underlying CEF, but due to the presence of some $\boldsymbol{c}_i \in \tilde{\boldsymbol{x}}_i$, $\boldsymbol{c}_i \notin \boldsymbol{x}_i$, the variable(s) of interest are confounded in the observed equation.

$$\begin{aligned} y_i &= \mathrm{E}\left[y_i|\tilde{\boldsymbol{x}}_i\right]) + \tilde{u}_i \quad \text{(Actual)} \\ &= \mathrm{E}\left[y_i|\, \boldsymbol{x}_i\right] + u_i. \quad \text{(Observed)} \end{aligned} \tag{6.3}$$

IV then exploits the (possible) existence of some instruments $\boldsymbol{z}_i$ which are relevant for the confounded (or endogenous) variables $\boldsymbol{z}_i \not\perp\!\!\!\perp x_{i,p}$, but independent from any other determinants of the outcome of interest $\boldsymbol{z}_i \perp\!\!\!\perp u_i$ (i.e. $\boldsymbol{z}_i \perp\!\!\!\perp \boldsymbol{c}_i$). Assuming the part of $\boldsymbol{x}_i$ associated with $z_i$ is linearly separable, this may be written as a classic triangular structure:

**Definition 6.1.** A linearly separable triangular system of $\boldsymbol{x}_i, \boldsymbol{z}_i$ and $y_i$ consists of:

$$\begin{aligned} y_i &= g(\boldsymbol{x}_i, \boldsymbol{\beta}) + u_i, & \boldsymbol{x}_i &\not\perp\!\!\!\perp u_i, \\ x_{i,p} &= g_p(\boldsymbol{z}_i, \boldsymbol{\beta}_{z,p}) + u_{i,x_p}, & \boldsymbol{z}_i &\perp\!\!\!\perp u_i. \end{aligned} \tag{6.4}$$

Such a structure allows causal estimates using a simple two-stage procedure. Some care is necessary however, since straightforward extension of the classical two-stage least squares (2SLS) estimator leads to inconsistent estimates. Such an approach would first estimate models for all endogenous $x_{i,p}$, leading to estimates $\hat{x}_{i,p} = \hat{\mathrm{E}}[x_{i,p}|\, \boldsymbol{z}_i]$, and then estimate $g(\hat{x}_{i,p}, \boldsymbol{\beta}_{\hat{x}})$. However, if $g(\cdot)$ is non-linear, we would actually need to estimate $\hat{g}(x_{i,p}, \beta)$ to obtain consistent estimates (Cameron & Trivedi, 2005, p. 199). Instead, it is wiser to implement a similarly motivated procedure which adds controls in the second stage, rather than replace endogenous variables (Kasy, 2011). Newey *et al.* (1999) suggests adding the residuals as variables in the second stage:[55]

$$\begin{aligned} x_{i,p} &= \mathrm{E}\left[x_{i,p}|\, \boldsymbol{z}_i\right] + u_{i,x_p}, & \text{(First stage)} \\ y_i &= \mathrm{E}\left[y_i|\, \boldsymbol{x}_i, \boldsymbol{u}_{i,x}\right]) + u_i. & \text{(Second stage)} \end{aligned} \tag{6.5}$$

The first stage splits $x_{i,p}$ into a part which is independent of $u_i$ (since it is a function of $\boldsymbol{z}_i$), as well as a residual which is still dependent on $u_i$. Adding the latter as a control in the second stage ensures that $x_{i,p} \perp\!\!\!\perp u_i | u_{i,x_p}$, so that we obtain valid estimates in the second stage. This follows the usual logic for 2SLS. Indeed, if $g(\cdot)$ is linear, we obtain identical estimates.

[54] For reviews of traditional IV methods, see Angrist & Pischke (2008) or Cameron & Trivedi (2005).

[55] The validity of this approach depends on the assumption of linear separability. If errors are not additive, it may be necessary to add the CDF of $x_{i,p}$ given $\boldsymbol{z}_i$ (Imbens & Newey, 2009). If errors are sufficiently complex however, it may be that no appropriate control function exist (Kasy, 2011).

**Table 6.3:** *Two-stage neural networks might just provide a decent RMSE*

| | OLS (I) | NN (I) | 2SLS | 2SNN | NN-OLS | OLS-NN |
|---|---|---|---|---|---|---|
| **Linear** | 1.45 | 1.45 | **0.01** | 0.13 | 0.05 | 0.11 |
| | (0.006) | (0.022) | (0.008) | (0.028) | (0.036) | (0.014) |
| **Polynomial (2)** | 2.45 | 6.26 | 3.72 | **0.57** | 3.81 | 37.84 |
| | (0.008) | (0.071) | (0.062) | (0.100) | (0.083) | (1.098) |
| **Polynomial (3)** | 158.56 | 35.42 | 94.58 | **26.94** | 162.64 | 102.21 |
| | (11.581) | (6.321) | (3.127) | (6.290) | (11.194) | (12.756) |
| **Pointy** | 14.67 | **1.38** | 15.04 | *1.55* | 14.68 | *1.55* |
| | (0.013) | (0.165) | (0.014) | (0.149) | (0.014) | (0.153) |
| **Wiggly** | 4.73 | **3.25** | 4.68 | *3.28* | 4.76 | *3.28* |
| | (0.021) | (0.171) | (0.021) | (0.211) | (0.021) | (0.167) |
| **Trig. pol (3)** | 1.10 | 0.88 | 1.09 | **0.22** | 1.11 | 2.44 |
| | (0.001) | (0.029) | (0.003) | (0.019) | (0.003) | (0.306) |
| **Ackley** | 2.11 | **2.02** | 2.11 | *2.06* | 2.12 | *2.06* |
| | (0.004) | (0.016) | (0.004) | (0.011) | (0.004) | (0.009) |
| **Rastrigin** | 5.19 | **3.44** | 5.19 | *3.50* | 5.19 | 3.67 |
| | (0.004) | (0.192) | (0.004) | (0.220) | (0.004) | (0.135) |
| **Drop-Wave** | 3.34 | 3.13 | 3.34 | **3.06** | 3.34 | *3.10* |
| | (0.004) | (0.056) | (0.004) | (0.021) | (0.004) | (0.044) |

*Notes:* The table shows root MSE between estimated and true marginal effects for a single regressor. Each row represents 100 simulations with 1 regressors, 3 confounders, 1 instruments and continuous output. Cells show averages across simulations, with standard errors in parentheses. Estimators trained on 200,000 observations. Reported values are from a test set. Values are from observed models, which exclude confounders.

The theoretical underpinnings are probability theoretic, and do not depend on the estimation procedure for the CEF, so I expect neural networks will do quite fine. They offer an alternative approach in both the first and second stage. A simple algorithm implementing the two-stage procedure for any two estimation techniques is provided in appendix III.4.[56]

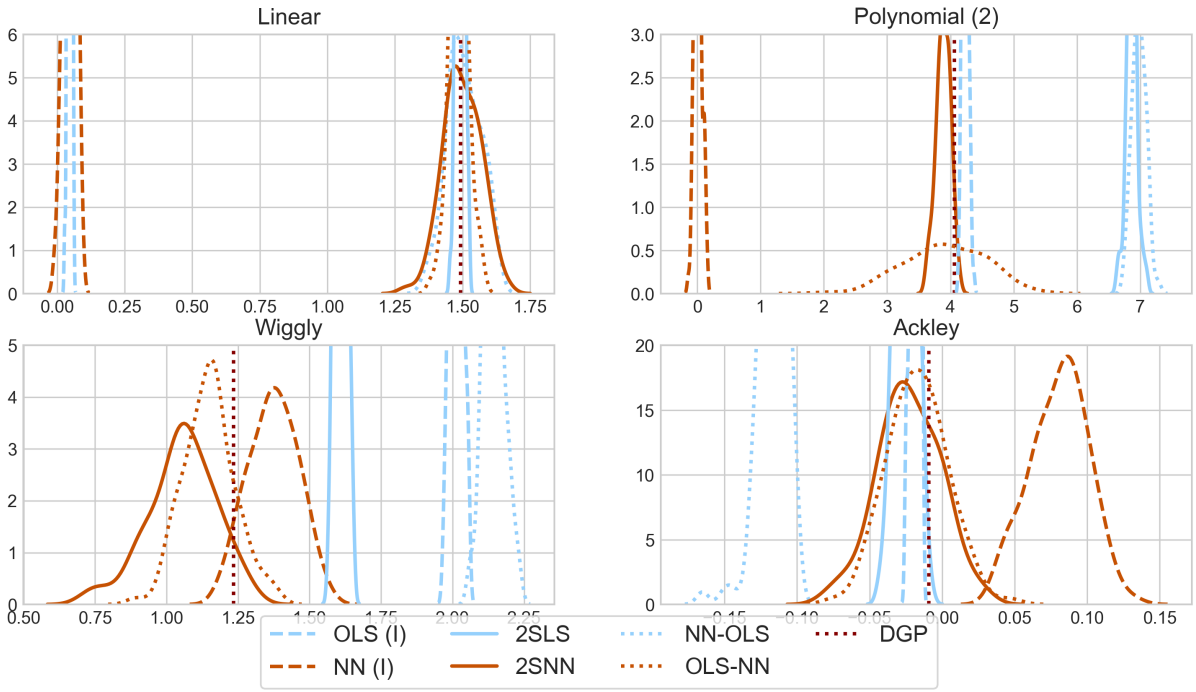
#### 6.3.2. Simulation results support IV efforts if data is plentiful

I implemented an exactly identified triangular structure, where every drawn $x_{i,p}$ is endogenous, but is also partly determined by an exogenous instrument $z_{i,p}$. I drew these instruments from the same distribution as $\boldsymbol{x}_i$ and $\boldsymbol{c}_i$, but imposed no dependency between instruments and confounders on the covariance matrix. To allow for non-linearity in the first stage, I simply used the same $g(\cdot)$ as in the second stage. I set all coefficients to one to ensure stable, relevant instruments.

$$\begin{aligned} x_{i,p} &= g(z_{i,p}, \boldsymbol{\beta}_{z,p}) + u_{i,x_p}, \qquad p = 1, ..., k, \\ \begin{pmatrix} \boldsymbol{u}_{i,x} & \boldsymbol{c}_i & \boldsymbol{z}_i \end{pmatrix} &\sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Omega}), \quad \boldsymbol{\mu} \sim \mathcal{N}(0, \boldsymbol{I}_k), \\ \text{diag}(\boldsymbol{\Omega}) = \mathbf{1}, &\quad \text{Cov}(\boldsymbol{c}_i, \boldsymbol{z}_i) = \mathbf{0}, \quad \forall p : \boldsymbol{\beta}_{z,p} = \mathbf{1}. \end{aligned} \tag{6.6}$$

[56] Above, I implicitly assumed all $x_{i,p} \in \boldsymbol{x}_i$ are endogenous. Exogenous covariates should not have their own first stage, but be added as controls in the first stage for endogenous regressors. The algorithm accounts for this.

**Figure 6.5:** *IV methods might just move estimates to the true values*

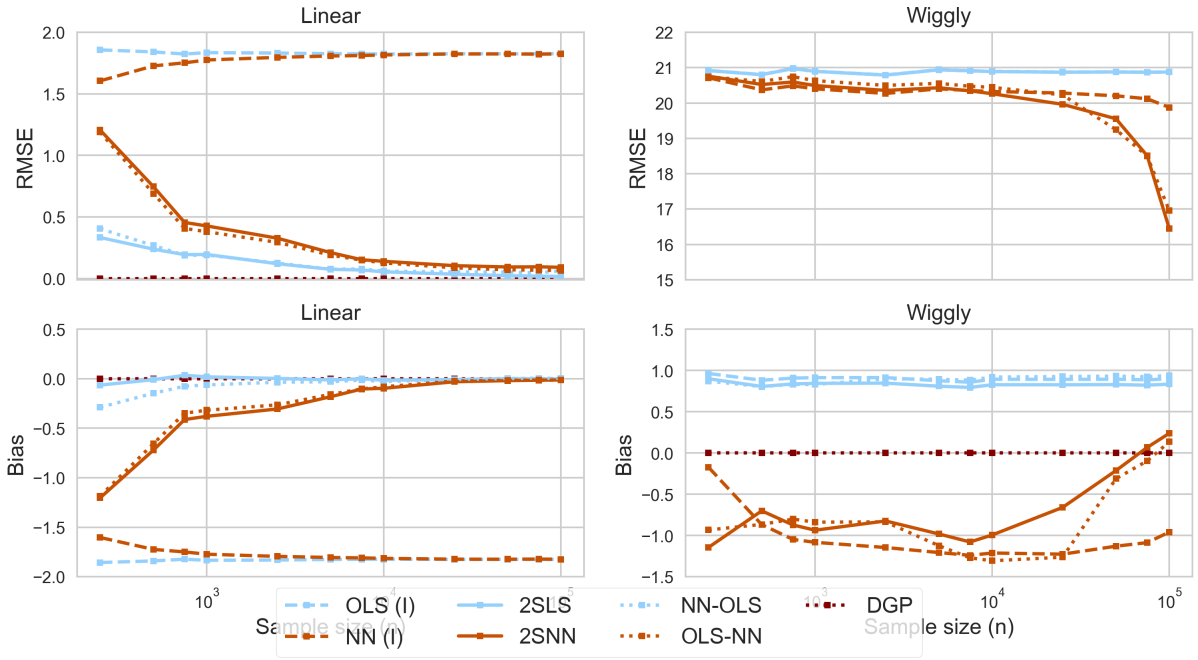



*Notes*: The figures shows the distribution of estimates of the average marginal effect for a single regressor. Each figure represents 100 simulations for the given DGP with 1 regressors, 3 confounders, 1 instruments and continuous output. Estimators were trained on 200,000 observations. Reported values are from a test set of equal size. Values are from observed models, which exclude confounders. Dashed line show original estimators. Solid lines show the main IV estimators. Dotted lines show hybrid estimators.

Table 6.3 shows the result of such a simulation. It provides the results of the basic estimators, as well as four IV estimators, based on all combinations of using OLS and IV in the first and second stage respectively. 2SLS implements classical two-stage least squares, which uses OLS in both stages. 2SNN instead uses neural networks in both stages, and the final two hybrid methods mix and match between the two. I look at root MSE of estimates in the observable case, where models do not observe confounders. The preferred model (marked in bold) is either 2SNN, or 2SNN obtains an error close to the preferred model (italics) in all scenarios. The likely reason the basic neural networks beats out 2SNN in these cases, even though it is biased, is that it is liable to converge to its optimal solution faster. Two-stage procedures are naturally less efficient, because they discard variation believed to be endogenous. Given more data, 2SNN would likely end up as the preferred estimator in all cases, because it is not asymptotically biased.

Figure 6.5 shows the latter claim by illustrating the elimination of bias. In all cases, the 2SNN estimator is either centered on the true marginal effect, or appears as the closest estimator, whereas the basic network is quite biased. In the latter cases, such as for the wiggly function in the figure, it is likely that more data would allow the estimator to center itself more elegantly on the true value. The linear combinations tend to produce biased results. 2SLS is clearly biased for the wiggly function, and NN-OLS is quite clearly biased for e.g. the quadratic function. OLS-NN does tend to be centered on the true value, but is far more inefficient (spread out) than 2SNN. All in all, 2SNN presents itself as the preferred estimator of averages.

**Figure 6.6:** *Consistency is restored! Hallelujah!*




*Notes*: The upper figures show MSE between estimated and true marginal effects for a single regressor. The lower figures shows the corresponding average difference (i.e. bias). Each figure represents a simulation study for the given DGP with 1 regressors, 2 confounders, 1 instruments and continuous output. 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. Values are from observed models, which exclude confounders. The second axis show averages across the simulations. Dashed line show original estimators. Solid lines show the main IV estimators. Dotted lines show hybrid estimators.

Figure 6.6 presents further evidence of asymptotic unbiasedness, and shows how this allows us to regain consistency. Two scenarios are shown. Consistency is apparent for all IV estimators in the linear case since MSE converges to zero, whereas dashed lines show that the naive estimators converge to a biased level, as expected. The lower set of figures show the bias of estimates, where we see that consistency is obtained because the bias goes to zero. The case is less clear cut for more non-linear wiggly function, where the bias does not come close to zero. However, note that for samples above about 50.000, the error starts to converge towards zero. The lower figures show that this occurs because the bias starts to go down. This may explain why the basic neural net was preferred for some of the more complex functions above: The IV estimator had not converged to any better level yet. However, given that the bias goes to zero, one should expect that the estimators perform better for large samples. Thus, if neural networks in general relies on medium to big data, IV approaches rely solely on big data.

In summary, neural IV networks provide a consistent estimator for true effects under typical assumptions, but this estimate is highly volatile since IV estimators discard variation presumed endogenous. Correspondingly, the results validate IV approaches, but I would caution against using them except with good data. This means both a large sample and relevant instruments. Even with a deceptively large number of observations, weak instruments tend to remove much variation, leading to a small effective sample. This is a well-known issue, but it is likely to affect neural methods even more due to their inherent volatility.

## 6.4. Case: Replication of Angrist & Krueger (1991)

The basic theoretical argument covered so far is that neural networks provide a viable econometric estimator for either direct or instrumental variable estimation of marginal effects. I will illustrate this by replicating a few findings from a classical study in applied econometrics: 'Does compulsory school attendance affect schooling and earnings?' by Angrist & Krueger (1991). The study is well-respected (and cited), the used data is freely available online, and their methodology is classic two-stage least squares. Hence it provides an open-and-shut case for applying neural networks on the real world. I find the same average effects of schooling, but argue that the returns seem heterogeneous, something the original study misses. Particularly, the returns seems to increase for the first extra years of schooling, and then decline.

### 6.4.1. Identification strategy: Compulsory schooling provides natural experiment

I will briefly review the theoretical argument of the paper, but I will not spend much effort discussing their identification strategy since the key focus is on comparing statistical properties.[57] The study looks to estimate the *returns to schooling*, the increase in expected wages caused by more education.[58] This is made difficult by the *ability bias* discussed previously: Ability is believed to be positively correlated with both years of schooling and expected wages. Therefore our estimates of the return to schooling are confounded if we do not appropriately control for ability, leading to (positive) omitted variable bias, where we overestimate the increase in wages. Angrist & Krueger (1991) suggests an identification strategy based on a natural experiment caused by compulsory schooling requirements:

> Children born in different months of the year start school at different ages, while compulsory schooling laws generally require students to remain in school until their 16th or 17th birthday. In effect, the interaction of school-entry requirements and compulsory schooling laws compel students born in certain months to attend school longer than students born in other months. (Angrist & Krueger (1991), p. 979)

This natural experiment can be leveraged by IV, with quarter of birth used as an instrument to isolate part of the variation in schooling. As always for IV, this relies on two arguments. First, the instrument must be *valid*. This requires that it is not itself confounded, so quarter of birth should be independently assigned from inherent ability. This is not a given, since season of birth is generally associated with later outcomes (such as health and earnings), possibly due to families with different socioeconomic backgrounds choosing to have children at different times of the year (Buckles & Hungerman, 2013). Validity also requires that the instrument only affects the outcome through the endogenous variable. However, since quarter of birth also assigns age at the beginning of school (which may affect educational attainment), this is not a given either. Despite these concerns, Angrist & Pischke (2014) argue that the instrument is valid, particularly since controlling for family background does not alter estimates much, and because younger children tend to be disfavoured, which runs counter to compulsory schooling, where younger children tend to be forced to more education (see also Angrist & Krueger (1992)).

[57] Note however that the study is cited by numerous textbooks in econometrics (only some of which Mr. Angrist wrote himself), usually with good remarks, so hopefully it is not entirely misguided.

[58] For an introduction to estimation strategies for returns to schooling, see Angrist & Pischke (2014).

Finally, instruments must also be *relevant* for the endogenous variable. The study presents evidence that quarter of birth is associated with educational achievement, and other studies have argued for the use of similar instruments (see e.g. Acemoglu & Angrist (2000)). However, even if quarter of birth can clearly account for some of schooling, it is not clear that it provides a strong instrument.[59] I am particularly worried that although the argument is clear why it should affect the first few years of schooling (where one cannot drop out), it is less clear that it can accurately predict higher levels of education. Hence estimates should probably mainly be interpreted as local effects on the first few years of education. These concerns may seem grave. However, note that it is always easier to criticize research designs than to construct them, and that instruments in particular can usually be challenged. The relative success of studies using quarter of birth instruments may be interpreted as evidence they get something right. Even if you remain unconvinced, their key use here is to illustrate and discuss statistical properties.

#### 6.4.2. Replication: Averages are similar, although volatile

The authors draw their main conclusions based on samples of men aged 40-49 from the United States, drawn from random sub-samples of the decennial censuses of 1970 and 1980. They present their main results in three tables for cohorts of men born in 1920-1929, 1930-1939 and 1940-1949 respectively. Replicating just one should be sufficient to illustrate neural networks. The authors themselves express greatest confidence in the instrument for the 1930-1939 cohort, so I will take a closer look at that table (Angrist & Krueger, 1991, table V, p. 1000). To keep results directly comparable, I analyse the full sample.[60]

I re-estimated their models, and obtained exactly the same results as those of their Stata log files. This indicates that I've obtained the right data, and am not a complete statistical klutz. These models use the logarithm of weekly wages as a dependent variable, and estimate the dependency on a main variable denoting years of schooling, as well as a number of covariates. The *basic* set-up only includes year of birth as control dummies. The second set-up adds age (and age squared) as control variables, and the third adds a number of control dummies (particularly race, region of residence and whether one is married). The *full* set-up include all mentioned covariates.

I also estimated a number of neural estimators. These correspond to the estimators used previously, except that I *standardized* all features before training the models, since neural networks tend to converge poorly if features are very small or large. It was unnecessary in my simulations since I drew features with unit variance and small means, but should not be forgotten in practice.[61] This requires a slight adjustment of the obtained marginal effects:

$$\begin{aligned} y_i = g(\tilde{\boldsymbol{x}}_i, \boldsymbol{\beta}) + u_i, \qquad \tilde{\boldsymbol{x}}_i = \frac{\boldsymbol{x}_i - \bar{\boldsymbol{x}}_i}{\boldsymbol{\sigma}_{\boldsymbol{x}}}, \\ \frac{\partial \operatorname{E}[y_i | \boldsymbol{x}_i]}{\partial x_{i,p}} = \frac{\partial g(\tilde{\boldsymbol{x}}_i, \boldsymbol{\beta})}{\partial x_{i,p}} = \frac{\partial g(\tilde{\boldsymbol{x}}_i, \boldsymbol{\beta})}{\partial \tilde{x}_{i,p}} \cdot \frac{\partial \tilde{x}_{i,p}}{\partial x_{i,p}} = \frac{1}{\sigma_{x_p}} \tilde{\gamma}_p(\boldsymbol{x}_i). \end{aligned} \tag{6.7}$$

[59] Bound *et al.* (1995) strongly critice weak instruments, using Angrist & Krueger (1991) specifically as an example, arguing that they imply that even minor violations of validity yields large inconsistencies.

[60] As mentioned, one might compare overall fit to a test set to evaluate whether a stable underlying process is identified. Either way though, one cannot directly evaluate marginal effects, since no true values are known.

[61] Without it, 2SNN estimates a 40 pct. wage increase from extra schooling. Which would certainly be nice.

**Table 6.4:** *Estimates of the return to schooling based on table V in Angrist & Krueger (1991)*

| | **OLS (I)** | **NN (I)** | **NN (II)** | **2SLS** | **2SNN** | **NN-OLS** | **OLS-NN** |
|---|---|---|---|---|---|---|---|
| **Basic** | 0.071*** | 0.073*** | 0.069*** | 0.089*** | 0.034 | 0.087*** | 0.044 |
| | (0.0004) | (0.0034) | (0.0047) | (0.0160) | (0.0625) | (0.0169) | (0.0416) |
| | {±0.001} | {±0.007} | {±0.011} | {±0.038} | {±0.133} | {±0.033} | {±0.087} |
| **w/age** | 0.071*** | 0.070*** | 0.072*** | 0.075** | -0.062 | 0.079*** | 0.043 |
| | (0.0004) | (0.0035) | (0.0039) | (0.0250) | (0.0692) | (0.0251) | (0.0464) |
| | {±0.001} | {±0.008} | {±0.008} | {±0.042} | {±0.234} | {±0.060} | {±0.090} |
| **w/dummies** | 0.063*** | 0.063*** | 0.067*** | 0.081*** | 0.023*** | 0.093*** | 0.077*** |
| | (0.0004) | (0.0030) | (0.0032) | (0.0167) | (0.0253) | (0.0057) | (0.0249) |
| | {±0.001} | {±0.006} | {±0.011} | {±0.035} | {±0.095} | {±0.017} | {±0.049} |
| **Full** | 0.063*** | 0.064*** | 0.067*** | 0.060*** | 0.069*** | 0.093*** | 0.053* |
| | (0.0004) | (0.0024) | (0.0026) | (0.0270) | (0.0224) | (0.0056) | (0.0353) |
| | {±0.001} | {±0.006} | {±0.010} | {±0.050} | {±0.047} | {±0.017} | {±0.079} |

*Notes:* The table shows the average marginal effect of an extra year of education. 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 measure. 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.

The validity of this approach can be seen by the fact that I also put the OLS estimates through it, and I still obtain the correct coefficients in the end. The resulting estimates are provided in table 6.4. I show the average marginal effect of an extra year of schooling on the log of wages. Notice that OLS and the neural networks agree quite closely in the basic set-ups, as I would expect based on my results above. Depending on the model specification, they estimate a 6-7 pct. increase in wages from an extra year of schooling. There is less agreement among the IV methods, where estimates differ somewhat based on the technique chosen for the second stage.[62] 2SLS tends to agree with the basic methods, but 2SNN predicts only a 3.4 pct. wage increase, and adding dummies actually predicts that wages decrease by 6 pct. However, in the full specification, 2SNN and 2SLS predicts a 6 and 7 pct. increase respectively. I would expect these differences to occur because 2SNN provides a volatile estimator, which has trouble finding a stable solution due to the small effect sample (due to weak instruments). Recall from simulation results that 2SNN requires much relevant variation to find solutions.

I accounted for this volatility by *bootstrapping* the estimates as described in section 5.3.[63] I use the resulting bootstrap distribution to estimate the distribution of possible estimates in a simple manner, which does not impose any assumptions of normality. Significance tests (at level $\alpha$) are then performed using the *percentile method*, which rejects the null if it falls outside the the lower and upper $\alpha/2$ quantiles of the bootstrap estimates (Cameron & Trivedi, 2005, p.364). The neural IV estimates are not significant in the models without the larger set of covariates.

[62] There is little disagreement in the first stage here. This is because, with the exception of age, all included variables are dummies, leading to *saturated* models which are linear by default.

[63] I used 99 bootstrapping replications which is on the low side, so estimates are somewhat uncertain. For discussions, see Cameron & Trivedi (2005), Davidson & MacKinnon (2004) and Andrews & Buchinsky (2000). 99 was chosen so $\alpha(B+1)$ is an integer for $\alpha \in \{0.01, 0.05, 0.1\}$, so the critical value is included (MacKinnon, 2002).

However, in the full model, they find a significant effect close to that of the non-IV estimates, indicating that these do not appear much biased (as the authors concluded in the original study). This suggests both that education has a positive effect on wages, and that this effect is not confounded much by ability bias. However, recall that we may criticise whether the effect is truly identified in the study. The key point here is that neural methods are capable of reproducing the average estimates of neural methods, as expected.

It is unwise to focus too heavily on significance tests, as reviewed in section 5.3. One may instead consider the estimated standard errors, which directly show that 2SNN provides a more volatile estimator. However, these are hard to judge with the naked eye. A popular choice is to instead look at confidence bounds on the estimates (Cumming, 2014). This allows us to directly evaluate a range of effect sizes, leading to a greater focus on practical significance rather than merely statistical. It can be easily estimated based on a bootstrap distribution using again a percentile method, where a confidence interval of level $\alpha$ is simply the distance between the upper and lower $\alpha/2$ quantiles (Cameron & Trivedi, 2005, p.365). This measure is not symmetric, since the actual estimate need not lie in middle. This is unfortunate, since experience tells us it is difficult to get economists to look at more than one summary number. I implemented a conservative measure where I calculated a 95 pct. confidence interval of estimates, and then reported the larger of the distances between the estimate and the end points of this interval.[64] This implies that the estimate lies within the bounds indicated with *at least* probability 95 pct. However, note that it cannot be used directly for significance testing, since even if the measure seems to overlap with zero, the larger difference might be on the opposite side. Still, it provides an easy way to summarize the range effects the estimate suggests. For the basic estimates, it suggests quite low volatility for both OLS and neural networks (although the two-layer network is slightly more volatile, as one would expect). The two-stage procedures are less certain: 2SLS estimates may be off by about 4 percentage points, and neural networks in the basic model specifications differ 13-23 percentage points as the sample is altered. No wonder it is not significant. However, the models with added covariates are far less volatile, with the fully specified neural network only allowing for a difference of 4.7 pct, less than that of 2SLS.

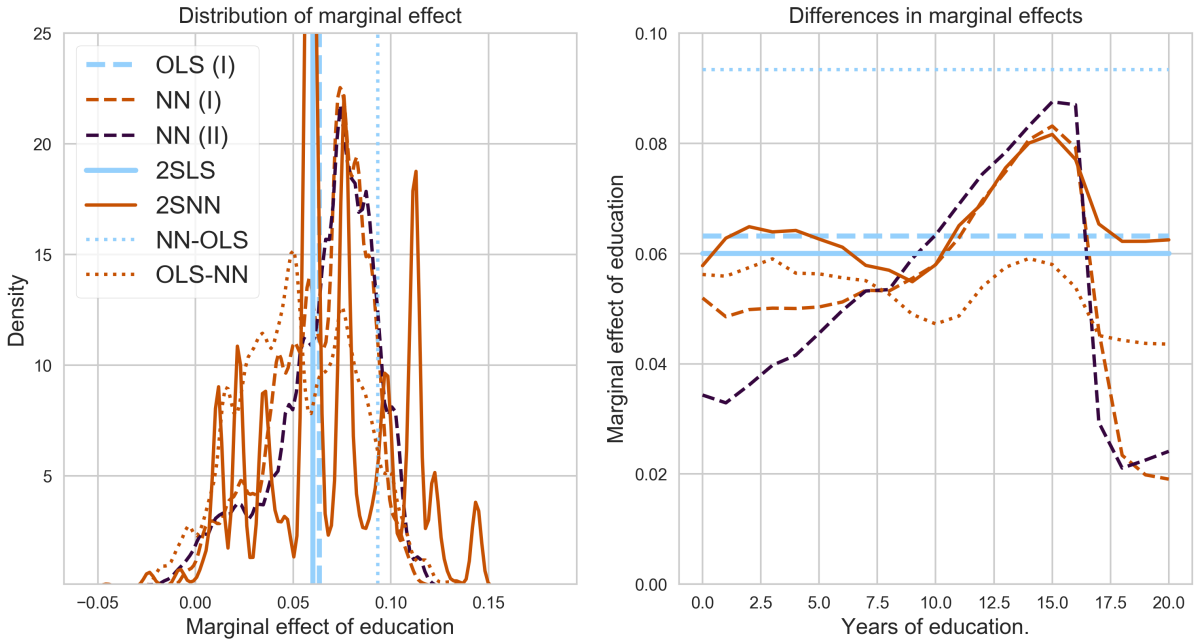
#### 6.4.3. Decreasing/increasing returns to education – Why not both?

So, why should we use neural networks if they just reproduce the same results? The reason, as argued in the simulation analysis, is that linear methods may miss heterogeneity in marginal effects by supplying averages for everyone. This provides not just poor individual estimates, but may cause researchers to misunderstand the basic forces at play, especially if marginal effects shift sign for some people. The study at hand provides a good example.

The left panel in figure 6.7 provides an initial indication of heterogeneity by looking at the distribution of estimates. The linear methods (blue) provide a single line since they offer the same marginal effect for everyone, while neural networks provide estimates ranging from close to zero to 0.15. The key challenge in this figure is to evaluate whether this reflects actual differences in marginal effects, or just the volatility of networks. Recall that even in the linear

[64] Formally, let $q_L/q_U$ be the lower/upper $\alpha/2$ quantile of the bootstrap estimates. Then i report $\epsilon = \max\{|\hat{\theta} - q_L|, |q_U - \hat{\theta}|\}$. Since $\theta \in \{q_L, q_U\}$ with probability 95 pct., we have $\theta = \hat{\theta} \pm \epsilon$ with at least 95 pct.

**Figure 6.7:** *But look! Heterogeneity!*



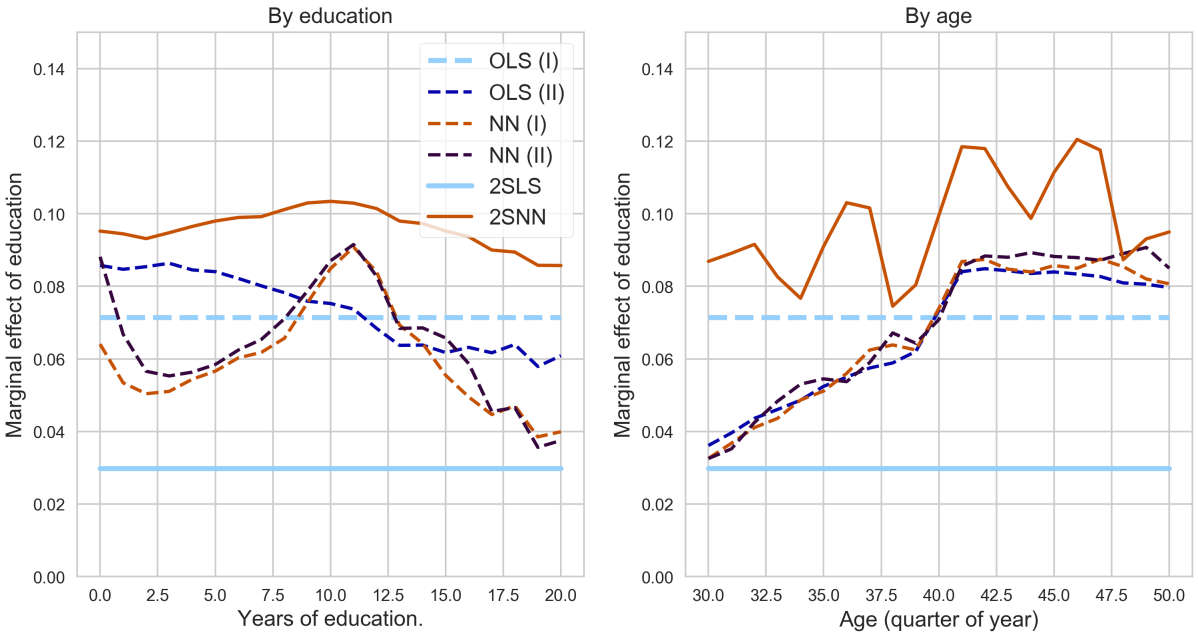

*Notes*: The left figure shows the distribution of estimates of the marginal effect of education. The right figure shows the average marginal effect of education for various levels of education. For every year of education, observations were grouped and the average marginal effect calculated. The estimated models correspond to the 'Full' model specification in table 6.4, which provide further notes.

simulations above, neural networks provided different individual marginal effects. In this case however, the effects looked much like a normal distribution around the linear estimates. The observed estimates here do not much resemble this picture, which indicates heterogeneity. It may be that wages respond differently to an extra year of education.

The figure does not tell us the actual shape of heterogeneity. Generally, we are interested in figuring out how the marginal effects depend on covariates (recall from section 2.3.1 that we can only uncover heterogeneity explained by included variables). I will look at one specific example: Are there increasing or decreasing returns to an extra year of education? This looks at how marginal effects depend on the current level of the regressor itself. I looked at this by computing the average marginal effect for every observation with a given level of education (in years). The right panel in figure 6.7 presents the results for the estimators. Again, the linear methods provide a straight line, but the neural networks present a more nuanced picture. Indeed, it appears that the question asked is too simple: There might be both increasing and decreasing returns. It seems the extra income from more schooling increases until about 16 years (at least from 10 years and onwards), but that it then starts to decrease, until an extra year only increases income by about 2 pct.

The results remain volatile, due to the low effective sample. In an effort to remedy this, I pooled the sample with the other two cohorts analysed in different tables in the study (males born in 1920-1929 measured in 1970, and 1940-1949 in 1980, respectively). This provides a full sample of about a million Americans, so it moves us securely into the realm of big data, and should reduce the variance of estimates. Despite this, estimates remain somewhat volatile to choice of starting

**Figure 6.8:** *Extra education yields greatest returns for mid-length schooling and greater age.*




*Notes*: The figures show the average marginal effect of education for various levels of the regressor on the first axis. For every year, observations were grouped and the average marginal effect was calculated. Data consists of 1,063,633 males born in the United States born between 1920-1949 and measured in 1970 or 1980. The model specification corresponds to the estimated 'Full' model in table 6.4, which provide further notes.

values (i.e. seed), and further attempts could be made to stabilize them. Particularly, more advanced forms of regularization (e.g. dropout) could be implemented, and stability of overall fit measured on a test set. Despite such concerns, the overall image seems clear, and follows the results shown in figure 6.8: The returns to educations increase at first, and start decreasing for longer education. This is slightly depressing news, as I finish 8 years at university.

The figure also shows an OLS estimator augmented with a polynomial expansion, the classical way to allow for decreasing marginal effects. This directly shows the drawback of the method, as estimates imposes that returns must be globally decreasing. While this seems to provide a better estimate of true effects than the constants of linear methods, it cannot provide as nuanced a picture as neural networks, because it ignores that returns are stable or increasing at first.

Neural networks also allow us to investigate how effects depend on other covariates. The right panel in figure 6.7 show that education pays off more for older men.[65] This indicates that extra schooling may be an investment which pays off more and more as one gets older, even though the schooling occurs early on. This is a plausible dependency, since it may be that people with longer education end up at different career tracks, with greater final payoffs.

In summary, neural networks are quite capable of recreating the main results of Angrist & Krueger (1991). Furthermore, they allow a more nuanced analysis of how the returns to schooling may depend on both current schooling and covariates.

[65]Note that the line shift around 40 years may reflect a difference in sample. All men below 40 were measured in 1970, while men above 40 may be from either 1980 or 1970. Therefore, they may not be directly comparable.

# 7. Conclusion: Neural networks provide an exciting econometric estimator

'Sometimes the questions are complicated and the answers are simple'.

– *Dr. Seuss (n.d.)*

My simulation study shows that neural networks provide a well-behaved estimator for marginal effects, without imposing assumptions on the functional form of the underlying relationship. This provides a useful extension of the econometric toolbox, because it allows assumption-free analysis of the heterogeneity of marginal effects. The estimator is mainly relevant with larger data sets, but bootstrapping may allow researchers to judge the robustness of their results no matter the sample size. The estimator is directly applicable to modern research since most identification strategies can likely be extended to allow for it, as I show for instrumental variables estimation. These claims are bold, and further research should investigate them as well. Particularly, it would be relevant to extend the results beyond my simulation study. If the results hold, it opens up a whole new venue of research, where econometricians can explore how to optimize neural networks for estimation of marginal effects.

## 7.1. Limitations and venues for future research

Before I summarize the results which underlie the claims above, it seems wise to consider what the paper does not show. The main purpose of the paper is to introduce and motivative the use of neural networks in econometrics by showing that they perform well in a relatively simple setting. However, the simplicity of the setting may cast doubt on the generality of the results. Similarly, the simplicity of the networks estimated may also cast doubt on whether performance is representative of actual research. Further research investigating either would be very interesting.

### 7.1.1. Do the results hold generally?

My results rely solely on a Monte Carlo simulation study which casts a natural shadow over generalizability since results are only valid for the simulated DGPs. One concern is whether claims of 'arbitrary non-linearity' can be justified. This concern should not be overstated, since my simulation set-up is quite general and investigated nine broad classes of underlying statistical processes, ranging from linear to highly complex. It is plausible to believe that if there was some general issue with the estimator, it would have been reflected in some of the scenarios.

A more pressing concern is that I only investigated well-behaved regressors. Less clean input could make the estimator even more volatile, further highlighting the reliance on big data. It could also become more prone to overfitting. All in all, more research investigating the behaviour of neural networks as an estimator of marginal effects would be welcome. Ideally, analytical derivations of properties under general conditions would solve the issues. Further simulation studies and analysis of real data could also do much to increase the reliability of the results.

Another concern could be theoretical. As reviewed in section 2.3.1, neural networks rely on assumptions of common functional form and independence across observations, which is slightly stricter than those offered in the literature (e.g. for local estimators). However, this basically means that heterogeneity must be visible through the covariates, which I argue most approaches assume (implicitly or explicitly), since one cannot reflect it otherwise.

#### 7.1.2. Is the estimator attractive compared to alternatives?

The paper focused on illustrating the properties of neural networks, not on comparing them to realistic alternatives. Such could include both typical econometric estimators (other than the baseline estimators analysed), and recent developments based on machine learning such as causal forests (as reviewed in section 2.1).[66] Given the success of networks in the literature on machine learning I consider it likely that they will be competitive, but research directly investigating the relative performance would be interesting.

One particular venue where I expect networks to perform well is with high-dimensional data (i.e. many regressors). Classical non-parametric estimators (such as kernel regression and local methods) typically run into trouble when dimensions become high, because data points become scarce (see e.g. Cameron & Trivedi (2005)). Deep learning has generally been quite succesful in high dimensions, and I expect the results to carry over for marginal effects. Unfortunately, I was unable to analyse this due to the limited memory budget available.

#### 7.1.3. Can we make the networks even better?

If convinced neural networks may work, one could look to optimize them. My study generally used default settings for basic networks, and no work was done individually optimizing them within simulations (since automatic approaches are computationally intensive, see e.g. Goodfellow *et al.* (2016)). There is a large literature on how to optimize neural networks. Since the goal is typically to reflect the underlying CEF more closely, it is likely that approaches would also improve estimates of its derivatives. There are many possible research venues in this direction.

It is not a given that all approaches in data science improve estimates. Although improving prediction often requires modelling the CEF more closely, not all their endeavours necessarily do so. One interesting venue is further research of the impact of regularization. Data scientists use it to prevent overfitting, which should improve estimation of the underlying CEF. However, theoretically it shifts the optimal solution to the objective function away from the CEF, which could affect marginal effects. Research on which effect is dominant would be quite interesting.

### 7.2. Summary: Consistent, fully heterogeneous marginal effects

I showed that neural networks provide a well-behaved estimator in two broad cases. First, I evaluated them in a classically 'clean' case where regressors are unconfounded (see section 2.2.1). Second, I evaluated their performance in an instrumental variables framework in a case where regressors of interest are confounded by unobserved variables.

#### 7.2.1. Neural networks do well if regressors are unconfounded

I found that neural networks provide a consistent estimator for true marginal effects in all simulated scenarios, where the data-generating processes investigated ranged from linear to highly complex. I also found that the approximation offered by neural networks was easily superior to linear approximations in an MSE-based framework (see section 2.3.2). However, the estimator was less efficient than maximum likelihood (which is reasonable with fewer assumptions), indicating that relatively large samples are needed to ensure decent estimates.

[66] However, forests are not continuous, so they may not be fitting for marginal effects of continuous regressors.

Neural networks allow analysis of heterogeneous marginal effects, without specifying the form. My simulation results showed how neural networks were capable of reflecting complicated underlying structures, enriching analysis beyond average effects. I also showed how this could allow for a more sophisticated analysis in a practical case by replicating a study by Angrist & Krueger (1991), where returns to education appear to be heterogeneous. Particularly, they appear to increase for the first extra years of schooling, and then decrease for higher education.

No technique will catch on in econometrics if it does not allow for hypothesis testing. Fortunately, bootstrapping techniques makes this easy even if limit distributions are unavailable. I define a bootstrapping approach for individual estimates, and validate it within my simulation set-up. I also validate it on the practical case, where it provides a quite reasonable robustness check.

#### 7.2.2. Neural networks can be extended to instrumental variables

Modern research often finds itself in situations where unconfoundedness does not hold, leading to biased estimates. Much of the research in econometrics has been focused on dealing with this, particularly the nefarious issue of confounders and omitted variable bias. Fortunately, both the issues and the solutions are largely dependent on properties of conditional expectations, rather than the estimator we use to obtain it. Therefore, they can be extended to neural networks.

I showed this by adapting an instrumental variables approach. I implemented a two-stage procedure for neural networks, and showed how it eliminated asymptotic bias from confounders in my simulation set-up. However, the estimator is volatile, and requires a large data sample to provide decent estimates (since IV procedures discard variation presumed endogenous). Given enough data, it allows valid, modern causal research using instrumental variables. This is seen both in my simulation results, and in the replicated study. In the latter, neural average estimates replicate linear effects. Two-stage estimates were generally volatile, but in the most well-defined model specification they produce reasonable, stable average results.

#### 7.2.3. Econometricians should really consider learning neural networks

These results underlie the basic claim made at the beginning of the paper: Neural networks can do anything one would expect of a regression, without imposing assumptions on the DGP. One easy way to see this is that throughout this entire paper, I have been able to show results from maximum likelihood, OLS and neural networks in the same tables and figures. The underlying theoretical goal is the same: Reflect the marginal effects of causal conditional expectations. Many of the pitfalls and biases are also the same, as I showed for irrelevant variables (noise), measurement error (attenuation bias) and confounders (omitted variable bias). Therefore, neural networks can be directly plugged into studies which rely on existing econometric estimators.

Many econometric studies could be enriched using neural networks. Fortunately, most econometricians should be able to learn how to use them fairly easily. Evaluating the output relies largely on tools provided in undergraduate courses, and the critical issues in training networks (which uses numerical optimization) are presented in graduate courses on microeconometrics. A few new concepts (such as overfitting and test sets) may actually enrich econometric analysis, while technicalities (such as efficient backpropagation) can be safely left to data scientists. 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'.

Acemoglu, Daron, & Angrist, Joshua. 2000. How large are human-capital externalities? Evidence from compulsory schooling laws. *NBER macroeconomics annual*, **15**, 9–59.

Amemiya, Takeshi. 1985. *Advanced econometrics*. Harvard university press.

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

Angrist, J, & Krueger, A. 1992. The Effect of Age at School Entry on Educational Attainment: An Application of Instrumental Variables With Moments on Two Samples. *Journal of the American Statistical Association*, **87**(418), 328–336.

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*.

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*.

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.

Buckles, Kasey S, & Hungerman, Daniel M. 2013. Season of birth and later outcomes: Old questions, new answers. *Review of Economics and Statistics*, **95**(3), 711–724.

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

Castro, Juan Luis, Mantas, Carlos Javier, & Benıtez, JM. 2000. Neural networks with a continuous squashing function in the output are universal approximators. *Neural Networks*, **13**(6), 561–563.

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.

Clarke, Arthur C. 1973. *Profiles of the Future*. Popular Library. Chap. Hazards of Prophecy: The Failure of Imagination.

Cumming, Geoff. 2014. The new statistics: Why and how. *Psychological science*, **25**(1), 7–29.

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**.

Foster, Ian, Ghani, Rayid, Jarmin, Ron S, Kreuter, Frauke, & Lane, Julia. 2017. *Big Data and Social Science–A Practical Guide to Methods and Tools*.

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

Fuller, Wayne A. 1987. *Measurement error models*. Wiley series in probability and mathematical statistics Probability and mathematical statistics. New York: Wiley.

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.

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

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

Guardian, The. 2014. Susan Greenfield: 'I've always marched to the beat of my own drum'. *Andrew Anthony*.

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

Heckman, James J. 2010. Building bridges between structural and program evaluation approaches to evaluating policy. *Journal of Economic literature*, **48**(2), 356–98.

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.

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.

Mitchell, Tom M. 1997. *Machine Learning*. McGraw-Hill, New York, NY, USA:.

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.

Pearl, Judea. 2000. *Causality, models, reasoning, and inference*. Cambridge University Press.

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.

Pischke, Steve. 2007. *Lecture notes on measurement error*. `http://econ.lse.ac.uk/staff/spischke/ec524/Merr_new.pdf`.

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

Robinson, Peter M. 1988. Root-N-consistent semiparametric regression. *Econometrica: Journal of the Econometric Society*, 931–954.

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.discrete.discrete_model.Logit*. `http://www.statsmodels.org/dev/generated/statsmodels.discrete.discrete_model.Logit.html`.

Statsmodels. 2018c. *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*.

Wansbeek, Tom, Tom J. 2000. *Measurement error and latent variables in econometrics*. Advanced textbooks in economics 37. Amsterdam: Elsevier.

Wasserman, Larry. 2016. 15: Prediction. *Lecture notes in Intermediate Statistics*, Carnegie Mellon University.

Watterson, Bill. 2005. *The Complete Calvin and Hobbes*. Andrews McMeel Publishing.

White, Halbert. 1980. A heteroskedasticity-consistent covariance matrix estimator and a direct test for heteroskedasticity. *Econometrica: Journal of the Econometric Society*, 817–838.

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.

Wikiquote. 2010. *Dr. Seuss*. `https://simple.wikiquote.org/wiki/Dr._Seuss`.

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

The comic on the front page is by Bill Watterson (2005).

All the initial quotes in sections are sourced above except the last one by Dr. Seuss. I could find no direct source, but it is generally attributed to him, see e.g. Wikiquote (2010).

# 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 $i = 1, ..., n$. |
| $k$ | Independent variables/features 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 $\ell$. Can be written $J$ if only one. Typically indexed $j = 0, ..., J^{(\ell)}$. |
| $B$ | Bootstrap replications. |
| $V$ | Irrelevant variables/features for each observation. |
| $C$ | Confounding variables for each observation. |
| $Z$ | Instruments for each observation. |

## 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), so e.g. $\boldsymbol{x}_{-i}$ are all features except those of observation $i$. Dimensions are noted below.

| | |
|---|---|
| $\boldsymbol{y}$ | $n \times 1$ column vector of dependent variables/labels for every observation. |
| $\boldsymbol{y}_{-i}$ | $(n-1) \times 1$ column vector of labels for every other observation $j \neq i$. |
| $y_i$ | Dependent variable/label for a single observation $i$ ($y_{\in} \boldsymbol{y}$). |
| $\mathcal{Y}$ | Regressand/label 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 variable, 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/features 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 $i$ ($x_{i,p} \in \boldsymbol{x}_i$). |
| $\boldsymbol{x}_{i,-p}$ | $1 \times (k-1)$ (or $1 \times k$) row vector of all features $j \neq p$ for observation $i$. |
| $\mathcal{X}$ | Regressor/feature space (allowed values of $\boldsymbol{x}_i$). |
| $\boldsymbol{u}$, | $n \times 1$ row vector of I.I.D. error terms. Substructure follows $\boldsymbol{y}$. |
| $\boldsymbol{v}$ | $n \times V$ matrix of irrelevant regressors/features. Substructure follows $\boldsymbol{x}$. |
| $\boldsymbol{c}$ | $n \times C$ matrix of confounders. Substructure follows $\boldsymbol{x}$. |
| $\boldsymbol{z}$ | $n \times Z$ matrix of instruments. Substructure follows $\boldsymbol{x}$. |
| $\boldsymbol{e}$ | $n \times k$ matrix of measurement errors. Substructure follows $\boldsymbol{x}$. |

$\boldsymbol{\Omega}$ 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.

$\boldsymbol{\theta}(\boldsymbol{x}_i)$ General expression for any estimator of a parameter or effect (e.g. $\boldsymbol{\beta}$, $f(\boldsymbol{x}_i)$ or $\gamma_p$).

$\epsilon$ 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 $\ell$.
$\boldsymbol{\beta}_j^{(\ell)}$ $(J^{(\ell-1)}+1) \times 1$ row vector of coefficients for node $j$ in layer $\ell$ (e.g. $J^{0)} = (k+1) \times 1$).
$\boldsymbol{\beta}_p^{(\ell)}$ $1 \times J^{\ell}$ column vector of coefficients for regressor $p$ across all nodes.
$\boldsymbol{\beta}_{j,p}^{(\ell)}$ Coefficient for feature $p$ in node $j$ in layer $\ell$.

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

Typically, to denote a slightly different version of any of these I use e.g. $\tilde{x_i}$. Its definition changes often in the paper, and is always clear from context (hopefully).

### I.3. Functions, derivatives & models

Functions with scalar output are denoted in normal font (e.g. $f(\boldsymbol{x}_i)$), and vector-valued functions in bold (e.g. $\boldsymbol{f}(\boldsymbol{x})$). 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. $\hat{f}(bx_i)$). They may be either common (e.g. $f(\boldsymbol{x}_i)$) or individual-specific (e.g. $f_i(\boldsymbol{x}_i)$). I do not repeat this for each.

$\boldsymbol{f}(\boldsymbol{x})$ 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 $\eta(s) = \max\{0, s\}$.
$F(\cdot)$ Cumulative distribution function (CDF) for a distribution.

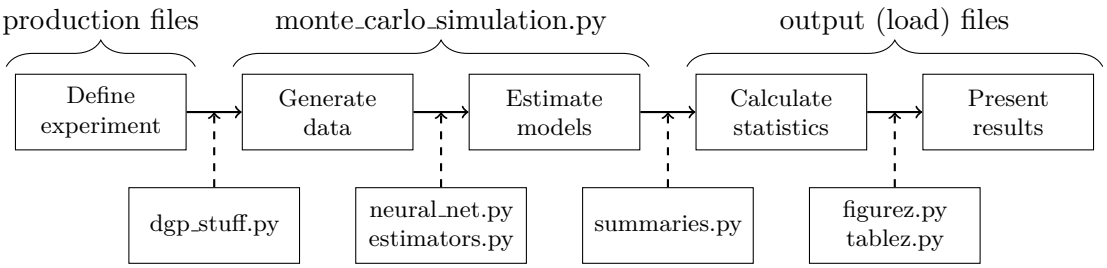
## II. Code (available at GitHub)

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. Therefore, I spend considerable effort here describing it. First, I provide a general overview of the files provided. Then I describe the simulations implemented. Then I review the main philosophies of the DGP, estimation and summary modules provided. Finally, I provide a summary of the output files available (which provide the figures actually used in the paper).

### II.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. The process is visualized in figure II.1.

**Figure II.1:** *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 may either be saved directly in memory, or saved to files on the PC. Results are then presented in an output file, which may load the results from files on the PC. This 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 described below.

**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 described in the paper.

**neural_net** implements the estimators and functions related to neural nets.

**estimators** implements estimator and functions related to the econometric estimators used.

**summaries** performs computations on the simulation results, as described below.

**figurez/tablez** provides tables and figures for the summaries.

The next two sections describe these modules in detail. The final section describes the output files included on GitHub, which provide all the figures and tables used in the paper.

## II.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, I also implemented general functions which repeated this algorithm in relevant ways, described next.

### II.2.1. Basic simulation algorithm

The basic algorithms performs M simulations for a given DGP and a given set of estimators. General parameters for the simulation, as well as specifics of the chosen DGP and estimators, are provided in three Python dictionaries to the function. All parameters are described in the beginning of the code for the algorithm, where defaults are also specified if they are not provided. A basic version of the algorithm would look like this:

**Algorithm II.1.** Monte Carlo simulation (simple version).
**Input**: Parameters, DGP, estimators
**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 $m = 1, ..., M$
   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 and estimators.

This basic algorithm is largely sufficient to understand the implementation. However, in practice I implemented a somewhat more complex algorithm (provided below). This was partly to allow for more complex scenarios, but it was also motivated by a number of technical challenges in implementing a simulation study of this size.

**Train and test data**. One could easily fear that overfitting interferes with the results here since I compare DGP properties calculated on the data rather than coefficients from the DGP. This implies that overfitting will provide an expectation close to the true one calculated, even if the estimator replicates noise rather than the true process. To account for this, I drew separate train and test data sets, and results provided in the paper are from test sets (unless otherwise stated). In the algorithm, all steps which draw data were repeated for two sets. Estimators were then only trained on the first sets, but results were calculated for both.

**Embarrassingly parallel**. Computation times were significant for the algorithm, particularly since training at least one neural network (and often more) in each iteration takes time. Fortunately, each iteration in the for loop over simulations is completely independent, a situation often called embarrassingly parallel because it is quite easy to parallelize the code (rather than run it serially). My algorithm uses a parallel version of the loop, where the simulations are split over the available processors on the PC, which typically yields a 3-4 increase in efficiency.

**Save files**. The simulations often took a long time. I therefore let the algorithm save the results, so one did not have to repeat the simulation every time one would like to look at the results.

**Actual/observed models**. The paper implements a number of alterations, where the data assumed to be observable is different than the actual data used by the DGP. My algorithm could either estimate a model for both of these cases, or just the observed case. The alterations include allowing for confounders $\boldsymbol{c}_i$, irrelevant variables $\boldsymbol{v}_i$, instruments $\boldsymbol{z}_i$ and measurement error $\boldsymbol{e}$. These are described in more detail in the paper. The formal implications are inserted in the algorithm. Note that in all formulas, if they are not provided simply insert the empty set $\emptyset$ instead.

**Bootstrapping**, as described in the paper, was also implemented in the algorithm. Note that bootstrapping greatly increases the computation time (since e.g. for 100 simulations, a boostrap estimator based on 100 simulations need to train $10^4$ neural networks instead of 100.

**Algorithm II.2.** Monte Carlo simulation (implemented version).
**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 $g\left(\begin{pmatrix}\boldsymbol{x}_i & \boldsymbol{c}_i\end{pmatrix}, \boldsymbol{\beta}\right)$ as specified (where $\boldsymbol{c}_i$ may be $\emptyset$).
   1.2. Draw $\boldsymbol{\mu}$, $\boldsymbol{\Omega}$ for $\boldsymbol{x}_i$ (and optionally $\boldsymbol{v}_i, \boldsymbol{c}_i$ and $\boldsymbol{z}_i$) as specified.
   1.3. Optionally draw $\Omega_{\boldsymbol{e}}$ for measurement error $\boldsymbol{e}$.
2. For simulation $m = 1, ..., M$ (either serially or in parallel):
   2.1. Generate data:
      2.1.1. Draw $\boldsymbol{x}$ (and optionally $\boldsymbol{v}_i, \boldsymbol{c}_i$ and $\boldsymbol{z}_i$) and $\boldsymbol{u}$ as specified.*
      2.1.2. Compute $\boldsymbol{y}$ as specified based on $g\left(\begin{pmatrix}\boldsymbol{x}_i & \boldsymbol{c}_i\end{pmatrix}, \boldsymbol{\beta}\right)$ 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 $\begin{pmatrix}\boldsymbol{x}_i & \boldsymbol{c}_i\end{pmatrix}$ (actual model, optional).
         2.2.2.2. Train estimator on $\begin{pmatrix}\boldsymbol{x}_i & \boldsymbol{v}_i\end{pmatrix} + \boldsymbol{e}$ with $\boldsymbol{z}_i$ as instruments (observable).
         2.2.2.3. Compute expectation and marginal effects.*
         2.2.2.4. Optionally bootstrap estimates.
3. Optionally save results to files on the PC.

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

### II.2.2. Extension to multiple DGPs

The paper compares various DGP functions $g(\boldsymbol{x}_i, \boldsymbol{\beta})$. I implemented an algorithm for easy computation of this. Each simulation starts from from the same pseudo-random seed for somewhat comparable responds (while each iteration of course starts from a different one).

**Algorithm II.3.** Simulation for different DGPs.
**Input**: Parameters, set of DGPs, estimators
**Procedure**:
1. For $g(\cdot)$ in the set of DGPS:
   1.1. Update parameters based on current DGP (e.g. network size).
   1.2. Perform simulation study following algorithm II.2.
   1.3. Optionally save results to single files for each simulation.
2. Optionally save files to aggregate files.

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

**Memory budget**. The option for saving individual files turned out to be necessary in practice, as simulation studies for multiple DGP's easily exceeded the meagre 8GB of RAM available on my PC. The current set-up only relies on memory sufficient for holding a single simulation study at a time. This also proved slightly restricting in practice, as many observations or explanatory variables could violate it, but I deemed it sufficient in practice (as the paper hopefully shows).

### II.2.3. Extension to series of parameters

The paper shows consistency by repeating the same simulation studies with various sample sizes $n$. I implemented an algorithm for easy computation of this. This relied on specifying a parameter space, such as $n \in \{250, 500, 750, 1.000, 2.500, ..., 75.000, 100.000\}$.

**Algorithm II.4.** Simulation for parameter series
**Input**: Parameters, set of DGPs, estimators, parameter space
**Procedure**:
1. For parameter (e.g. $n$) in specified parameter space:
   1.1. Perform simulation studies for each DGP following algorithm **??**.
   1.2. Optionally save results to individual files if not done in sub algorithm.
2. Optionally save results in aggregate files.

**Output**: Results for parameters, DGPs and estimators.

Note that the algorithm easily hits memory problems, so the latter option for aggregate saving is rarely useful in practice. Mostly, I simpled used the individual saving in the subalgorithm, again only restricting myselves to a single simulation study in memory at a time.

## II.3. Description of the supporting modules

Six supporting modules are used. The first three are used directly in the simulation module, and provides the DGP functions, data distributions and estimators used in the algorithms above. The next three provides tools for summarizing and presenting the results of these algorithms.

### II.3.1. DGP module

The module *dgp_stuff.py* provides two main features. First, it provides functions for drawing the various stochastic variables used from distributions. This concerns drawing population parameters such as $\boldsymbol{\beta}$, $\boldsymbol{\mu}$ and $\boldsymbol{\Omega}$ under the various constraints mentioned in the paper, and drawing observations such as $\boldsymbol{x}_i, \boldsymbol{u}_i, \boldsymbol{c}_i$ etc. The distributions used are described in the paper.

Secondly, it provides all the DGPs used in the paper. For every scenario considered (and a number of scenarios discarded), it provides a $g(\boldsymbol{x}_i, \boldsymbol{\beta})$ function, its marginal effects $\partial g(\boldsymbol{x}_i, \boldsymbol{\beta})/\partial \boldsymbol{x}_i$ and its gradient $\partial g(\boldsymbol{x}_i, \boldsymbol{\beta})/\partial \boldsymbol{\beta}$ for use in maximum likelihood estimation. Functions for both of these features are passed as inputs to the simulation functions above (as part of the parameters). All the functions used in the paper are intuitively described in the main paper, and formally described in appendix IV.

### II.3.2. Neural network & estimator modules

The module *neural_net.py* provides all my implementations for neural networks, and the module *estimators.py* provides similar features for the econometric estimators used for comparison in the the paper. Estimators are passed to the simulation function as an input, and must therefore follow a fixed format. Generally, this follows the following algorithm:

**Algorithm II.5.** General estimation of statistical model.
**Input**: Data, estimation parameters
**Note:** Steps marked with * are repeated for a train and test set.
**Procedure**:

1. Obtain estimated coefficients $\hat{\boldsymbol{\beta}}$ by fitting the model on training data.
2. Calculate expectation $\hat{\boldsymbol{f}}(\boldsymbol{x})$ based on $\hat{\boldsymbol{\beta}}$.*
3. Calculate marginal effects $\hat{\boldsymbol{\gamma}}(\boldsymbol{x}_i)$ based on $\hat{\boldsymbol{\beta}}$.*.

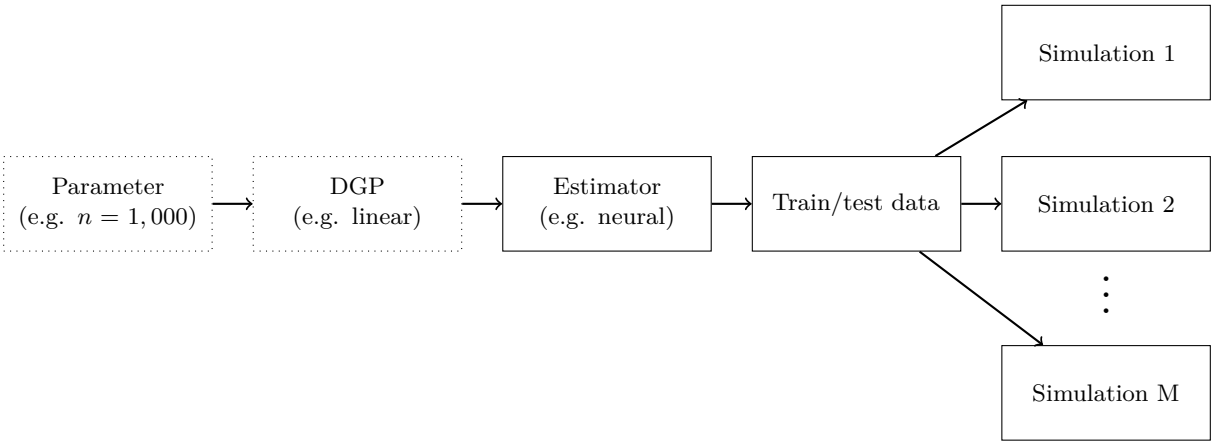
**Output**: Estimated coefficients $\hat{\boldsymbol{\beta}}$, expectations $\hat{\boldsymbol{f}}(\boldsymbol{x})$, marginal effects $\hat{\boldsymbol{\gamma}}(\boldsymbol{x}_i)$.

The steps for fitting the model and extracting results relies on procedures specific to estimation techniques at hand, which are provided in the modules for all estimators used. They are described formally in appendix III. Particularly, algorithms for extracting results from neural networks are provided. The neural network code also provides a bootstrapping algorithm corresponding to definition 5.2, which can be used for any estimator following the algorithm above.

### II.3.3. Summary module

The module *summaries.py* provides algorithms which can calculate summaries of the simulated data within the data structure used. The algorithms produce a plethora of data, with different results both within a single simulation study for every used estimator, and with different results across studies for various DGPs and parameters. To keep myself sane, I used a disciplined data structure based on a nested logic (implemented using Python's dictionaries). The logic is visualized in figure II.2. Generally, everything to the right of a given level is available for all current levels (e.g. simulation results for train and test data are available for every estimator).

**Figure II.2:** *Data structure*




Results for every simulation performed are at the bottom. This could either coefficients for the estimated models, a vector of expectations for every observation, or a matrix of marginal effects for every observation and variable. Such results are available for both test and train data. Results for these are in turn available for every estimator. If simulations were run for multiple DGPs, the full set of results are avaiable for each, and similarly for a simulationsa cross a parameter space. This quickly produces a large number of objects.

This yielded a minor coding hurdle, since the objects of main interest (simulation results) are at the bottom of the objects provided. Writing custom functions for every goal would be needlessly tiresome. Instead, I designed a few I wrote a few general algorithms for summarizing these . They all relied on defininining *bottom-level* functions which apply to a set of simulations, and e.g. calculated average marginal effects or MSE. In general, they provide a single summary number for every simulation. They are provided in the summary module.

I then wrote a number of general wrappers for passing such functions through the code structure. The most basic of these was for a single simulation:

**Algorithm II.6.** Summarize results for single simulation study:
**Input**: Simulation results, bottom-level function
**Procedure**:
1. For estimator in estimators:
    1.1. For train and test data:
        1.1.1. Apply bottom-level function to set of simulation results.

**Output**: Summary following figure II.2, but with a single value for each simulation.

This was not directly used in the paper. Instead, simulations were often run for a set of DGP's. Summary here was slightly complicated by the fact that results for each simulation study were often saved to individual files for memory reasons (see above). These were typically summarized in tables since they are naturally 2D (estimators and DGPs). Following the table functions described below, this could easily be summarized using the same data structure as above. Therefore, I designed a similar wrapper:

**Algorithm II.7.** Summarize results for simulation studies of vairous DGPs.
**Input**: Simulation results, bottom-level function, DGPs
**Procedure**:
1. For DGP in the set of DGPs:
   1.1. If result saved individually, load current file.
   1.2. Apply algorithm II.6.

**Output**: Summary following figure II.2, but with a single value for each simulation.

The second approach often used in the paper is to repeat simulation studies for a set of parameters. This leads to a similar wrapper. However, the end goal now is to summarize the results in a figure for each DGP, since data is now naturally 3D (DGP, estimator and parameter), which is why I split the DGP part out. The summarizer therefore has a slightly different output, where a dataframe is provided for each DGP with the parameter space along the rows, and estimators as columns.

**Algorithm II.8.** Summarize results for simulation studies of parameter space.
**Input**: Simulation results, bottom-level function, DGPs, parameter space.
**Procedure**:
1. Prepare output dataframes.
2. For parameter in parameter space:
   2.1. If result saved individually, load current file.
   2.2. Apply algorithm II.7 (may load files).
   2.3. Save result for for parameter as row in each DGP dataframe.

**Output**: Summary dataframe for each DGP with parameter rows and estimator columns.

The general idea is to allow for an quite general set-up. Similar to how the simulation algorithms can easily be changed to allow for different data, DGPs or estimators, new summaries can easily be coded up. One only has to define a new bottom-level function, and then run it through the wrappers. The results can then easily be summarized in tables or figures corresponding to the ones in the paper, as i review next. However, one can also easily define new tables or figures, since these are implemented using the same bottom-level principle.

#### II.3.4. Table & figure modules

The modules *tablez.py* and *figurez.py* provides the necessary tools for building tables and figures like those in the paper. They provided two main functionalities. First, they provide general wrappers which outputs from the summary algorithms above and move through the data structure to create general tables and sets of figures by once again applying bottom-level functions to the simulation results. Second, they provide such bottom-level functions for the wrappers, which creates different cells and figures. New presentations can be constructed by solely supplying new bottom-level functions.

## II.4. 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', and named after the section they feature in (e.g. files starting with '5_1' are presented in section 5.1.). Note that you may need to change the paths supplied to import the code modules.

### II.4.1. Production files

The simulation experiments are carried out using production files (marked by 'prod' in their titles). These call the simulation functions, and save the resulting estimates to disk. There may be several production files for a given section, corresponding to different simulation functions. These are distinguished by 'versions' (denoted by a 'v' in the name), where 'v2' refers to simulations for one sample across DGPs (tables and figures), and 'v3' refers to simulation over a parameter series (figures). Titles also denote whether output is binary or continuous.

One will need to run the production files before the loading files below can be used, since no simulation files are provided since they take up alot of memory. Note that simulations generally take a while to run (10-12 hours for basic results, 20-24 for more advanced).

### II.4.2. Load files (notebooks)

Actual figures and tables are then presented in Jupyter notebooks (marked by 'load' in their titles). These load simulation results saved by the corresponding production file, and presents a number of tables and figures, some of which are featured in the paper. Note that one may need to update the titles of the simulation results with the current date.

Load files cannot be updated without running the production files above. However, in most notebooks the last run results should be available, so one can inspect the range of figures. However, these results need not coincide with the exact ones used in the paper, since I often reran simulations with different seeds and specifications to check robustness.

### II.4.3. Replication files

The code files for the replication of Angrist & Krueger (1991) have a unique structure. They perform an entire data pipeline, loading and processing data, estimating models and presenting results. However, the estimators still rely on the general code module.

They rely on a dataset which is available from `https://economics.mit.edu/faculty/angrist/data1/data/angkru1991`. Data processing largely follows the Stata files also provided there.

### II.4.4. Auxilliary files

A few further files are provided. One file generates the examples for neural networks in section 3, using the basic neural estimator on a range of regressor values. Another file generally produces the visualizations of functions used in section 4 and appendix IV. It can generally visualize any function provided in the DGP module. Only one file is provided, so one would need to customize it to the function at hand to replicate figures.

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

# III. 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.

## III.1. Neural networks

Estimated neural networks are based on the implementation provided by *Scikit-learn* (Pedregosa *et al.*, 2011). The implementation is 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 use their implementation mainly to train the networks, and have implemented my own algorithms capable of finding predictions and marginal effects based on the resulting coefficients. The implementations are found in the code file *neural_net.py*.

### III.1.1. Estimation of optimal parameters

The optimal parameters are estimated by minimizing the following loss functions for regression and classification, respectively. They correspond to section 3.2.2 with added L2-regularization. $f(\boldsymbol{x}_i \,|\, \boldsymbol{x}_i)$ follows the definition of a neural network provided in section 3.3.

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

**Optimization: Adam**. Numerical optimization of equation (III.1) is typically done using some version of stochastic gradient descent. There is a large literature on how to do so efficiently, with various competing algorithms. I employed the *Adam* algorithm, which which provides a good default algorithm for relatively large datasets (+1000 obs). The name derives from 'adaptive moment estimation', reflecting that the stochastic optimizers automatically decides how much to update the gradient based on estimation of the lower moments. The algorithm is provided and described in Kingma & Ba (2014) I used default parameters from *Scikit-learn* (particularly a training rate of $10^{-4}$), except that I increased the allowed number of iterations to 500 epochs to increase chances of convergence, since I could not monitor each simulation closely.

**Regularization: L2**. The objective above adds a *weight decay* term to the basic likelihood function, which discourages large coefficients in $\boldsymbol{\beta}$. Particularly, it adds $L2$-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 monitors a validation set and stops the optimization when the validation score stops improving rather than the training score (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). A brief argument for

why is provided in section 3.2.1. 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 purpose of the output function is to ensure predictions belong to the correct space. The implementation uses the linear output $\sigma(s) = s$ for regression (which is correct by definition) and sigmoid units in the binary (which is correct with-in my simulation set-up), corresponding to the approach for squashing linear signals in logistic regression. 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 4.1 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.

Generally, I used quite rigid hyperparameters. It is likely that better estimates could have been obtained by tailoring the networks more closely to the functions. Particularly, it is likely a custom architecture for each function would do better. This would not necessarily have relied on knowledge of the DGP - performance could probably have been approved by implementing grid search or similar features. Doing so repeatedly would be computationally infeasible, but it could have been done once for each function and repeated. However, I wanted to show that neural networks do well even without excessive customization. In this sense, my results show how a naive approach can still obtain decent, consistent estimates of the objects of interest.

#### III.1.2. Estimation of expectations and marginal effects

The expectation $f(\boldsymbol{x}_i \,|\, \boldsymbol{\beta})$ for a deep neural network is provided in definition 3.2. The matrix notation is neat for a single $\boldsymbol{x}_i$, but not for the entire dataset $\boldsymbol{x}$, since there will be a dimensional mismatch. Instead, it serves to employ a simple algorithm for forwards propagation[67]:

**Algorithm III.1.** Forward pass through neural network as in definition 3.2.
**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**: $\boldsymbol{f}(\boldsymbol{x} \,|\, \boldsymbol{\beta}) = \boldsymbol{h}^{(L)}$.

[67] The algorithm here largely follows Abu-Mostafa *et al.* (2012b).

Since expectations are provided by *Scikit-learn*, I did not need this algorithm directly. However, since marginal effects are not provided by their module, I had to implement an algorithm for calculating these. The computation relies relies on elements from the forward pass, so I had to recreate it. The algorithm recreates the results of applying proposition 3.2, but sidesteps the dimensional mismatch of using the entire $\boldsymbol{x}$, allowing for relatively efficient computation.

**Algorithm III.2.** Computation of marginal effects for a neural network as in def. 3.2.
**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 III.3.
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 provides the exact same result as going through the proposition for each observation. The algorithm does not depend particularly on the Scikit-learn implementation, but just on adjusting the coefficient matrices to fit the dimensions used here.

### III.2. Maximum likelihood estimation

Maximum likelihood was implemented using the Python module *statsmodels* (Seabold & Perktold, 2010), which provides a generic likelihood framework allowing custom likelihoods (Statsmodels, 2018a). It implements traditional maximum likelihood estimator, which is the set of parameters which maximizes the likelihood of data (or, equivalently, maximizes the log-likelihood)

$$\mathcal{L}_N(\boldsymbol{\beta}) = \ln L_N(\boldsymbol{\beta}) = \ln f(\boldsymbol{y}, \boldsymbol{x} \,|\, \boldsymbol{\beta}). \tag{III.2}$$

$f(\boldsymbol{y}, \boldsymbol{x} \,|\, \boldsymbol{\beta})$ here denotes the joint probability mass or density function. These are defined for the cases at hand below. Typically, there is no analytical solution, leading to numerical optimization. Statsmodels allows for a number of gradient and non-gradient based optimizers. I've opted for gradient-based ones for faster convergence, which leads to the first order conditions

$$\frac{\partial \mathcal{L}_N(\boldsymbol{\beta})}{\partial \boldsymbol{\beta}} = \sum_{i=1}^{n} \frac{\partial \ln f(y_i | \, \boldsymbol{x}_i, \boldsymbol{\beta})}{\partial \boldsymbol{\beta}} = \mathbf{0}. \tag{III.3}$$

This requires specification of the gradient and possibly the Hessian. I used the BFGS algorithm, which provides a quasi-Newton optimizer which relies on the gradient, but does not evaluate the Hessian directly. General expressions for the gradient are therefore derived below. They, and the log-likelihood, depend on the function and gradient, which are derived for all scenarios in appendix IV. The implementation is found in the code file *estimators.py*.

### III.2.1. Log-likelihood & gradient for regression with Gaussian noise

The regression case I've studied (see definition (4.2) is the quite standard case of providing a mean $g(\boldsymbol{x}_i, \boldsymbol{\beta})$ and a Gaussian error term. This leads to normally distributed data around said mean $\mathcal{N}(g(\boldsymbol{x}_i, \boldsymbol{\beta}), \sigma_u^2)$, leading to the following (log-)likelihood:

$$\begin{aligned} f(y_i|\, \boldsymbol{x}_i, \boldsymbol{\beta}) &= \frac{1}{\sqrt{2\pi\sigma_u^2}} \exp\left\{ -\frac{(y_i - g(\boldsymbol{x}_i, \boldsymbol{\beta}))^2}{2\sigma_u^2} \right\}, \\ \mathcal{L}(\beta) &= -\frac{1}{2\sigma_u^2} \sum_{i=1}^{n} (y_i - g(\boldsymbol{x}_i, \boldsymbol{\beta}))^2 \underbrace{-\frac{n}{2} \ln 2\pi - n \ln \sigma_u}_{\text{No } \boldsymbol{\beta}}. \end{aligned} \tag{III.4}$$

My implementation does not seek to identify the variance $\sigma_u^2$. Hence, my formulas set $\sigma_u^2 = 1$ and drop the last part (which does not affect optimization, since it does not depend on parameters). Derivation of the gradient is straightforward:

$$\frac{\partial \mathcal{L}(\beta)}{\partial\, \boldsymbol{\beta}} = \frac{1}{\sigma_u^2} \sum_{i=1}^{n} (y_i - g(\boldsymbol{x}_i, \boldsymbol{\beta})) \frac{\partial g\left(\boldsymbol{x}_i, \boldsymbol{\beta}\right)}{\partial\, \boldsymbol{\beta}}. \tag{III.5}$$

I again set $\sigma_u^2 = 1$ in my implementation. Expectations and marginal effects are computed just as they are for the DGP (see section 4.2):

$$\mathrm{E}[y_i|\, \boldsymbol{x}_i] = g(\boldsymbol{x}_i, \boldsymbol{\beta}), \qquad \frac{\partial\, \mathrm{E}[y_i|\, \boldsymbol{x}_i]}{\partial x_{i,p}} = \frac{\partial g(\boldsymbol{x}_i, \boldsymbol{\beta})}{\partial \beta_p}. \tag{III.6}$$

### III.2.2. Log-likelihood & gradient for binary choice

The derivation follows ch. 14. in Cameron & Trivedi (2005), and is quite standard. For a binary $y_i \in \{0, 1\}$, the likelihood function is Bernoulli-distributed by definition:

$$\begin{aligned} f(y_i|\, \boldsymbol{x}_i, \boldsymbol{\beta}) &= p_i^{y_i}(1 - p_i)^{1-y_i}, \qquad p_i = F\left(g\left(\boldsymbol{x}_i, \boldsymbol{\beta}\right)\right), \\ \mathcal{L}(\beta) &= \sum_{i=1}^{n} \left[ y_i \ln F\left(g\left(\boldsymbol{x}_i, \boldsymbol{\beta}\right)\right) + (1 - y_i) \ln\left(1 - F\left(g\left(\boldsymbol{x}_i, \boldsymbol{\beta}\right)\right)\right) \right]. \end{aligned} \tag{III.7}$$

The gradient for general binary processes is not so pretty though:

$$\begin{aligned} \frac{\partial \mathcal{L}(\beta)}{\partial\, \boldsymbol{\beta}} &= \sum_{i=1}^{n} \left[ \frac{y_i}{F\left(g\left(\boldsymbol{x}_i, \boldsymbol{\beta}\right)\right)} - \frac{1 - y_i}{1 - F\left(g\left(\boldsymbol{x}_i, \boldsymbol{\beta}\right)\right)} \right] F'\left(g\left(\boldsymbol{x}_i, \boldsymbol{\beta}\right)\right) \frac{\partial g\left(\boldsymbol{x}_i, \boldsymbol{\beta}\right)}{\partial\, \boldsymbol{\beta}} \\ &= \sum_{i=1}^{n} \left[ \frac{y_i - F\left(g\left(\boldsymbol{x}_i, \boldsymbol{\beta}\right)\right)}{F\left(g\left(\boldsymbol{x}_i, \boldsymbol{\beta}\right)\right)\left(1 - F\left(g\left(\boldsymbol{x}_i, \boldsymbol{\beta}\right)\right)\right)} \right] F'\left(g\left(\boldsymbol{x}_i, \boldsymbol{\beta}\right)\right) \frac{\partial g\left(\boldsymbol{x}_i, \boldsymbol{\beta}\right)}{\partial\, \boldsymbol{\beta}}. \end{aligned} \tag{III.8}$$

I can simplify slightly when I assume $F(\cdot)$ is CDF for the logistic distributions $\Lambda(z) = 1/(1+e^{-z})$, since $\Lambda'(z) = \Lambda(z)[1 - \Lambda(z)]$ and $\Lambda(z) = 1 - \lambda(-z)$.

$$\begin{aligned} \mathcal{L}(\beta) &= \sum_{i=1}^{n} \left[ y_i \ln \Lambda\left(g\left(\boldsymbol{x}_i, \boldsymbol{\beta}\right)\right) + (1 - y_i) \ln \Lambda\left(-g\left(\boldsymbol{x}_i, \boldsymbol{\beta}\right)\right) \right], \\ \frac{\partial \mathcal{L}(\beta)}{\partial\, \boldsymbol{\beta}} &= \sum_{i=1}^{n} \left[ y_i - \Lambda\left(g\left(\boldsymbol{x}_i, \boldsymbol{\beta}\right)\right) \right] \frac{\partial g\left(\boldsymbol{x}_i, \boldsymbol{\beta}\right)}{\partial\, \boldsymbol{\beta}}. \end{aligned} \tag{III.9}$$

Expectations and marginal effects may again be computed just as for the DGP:

$$\mathrm{E}[y_i|\,\boldsymbol{x}_i] = F_u(g(\boldsymbol{x}_i,\boldsymbol{\beta})), \qquad \frac{\partial\,\mathrm{E}[y_i|\,\boldsymbol{x}_i]}{\partial\beta_p} = F_u'(g(\boldsymbol{x}_i,\boldsymbol{\beta}))\frac{\partial g\,(\boldsymbol{x}_i,\boldsymbol{\beta})}{\partial.\,\boldsymbol{\beta}} \tag{III.10}$$

## III.3. Linear methods

The linear estimation techniques were also implemented using the Python module *statsmodels* (Seabold & Perktold, 2010). They are standard implementations, so there should be few surprises. All implementations are included in the code file *estimators.py*.

### III.3.1. Ordinary least squares

Statsmodels (2018c) provides a completely standard implementation of OLS, with assumed linearity leading to easy expressions and an analytical solution:

$$\boldsymbol{\beta} = \left(\boldsymbol{x}^T\,\boldsymbol{x}\right)^{-1}\boldsymbol{x}^T\,\boldsymbol{y}, \qquad \mathrm{E}[y_i|\,\boldsymbol{x}_i] = \boldsymbol{x}_i\,\boldsymbol{\beta}, \qquad \frac{\partial\,\mathrm{E}[y_i|\,\boldsymbol{x}_i]}{\partial x_{i,p}} = \beta_p. \tag{III.11}$$

### III.3.2. Logistic regression

The logistic regression provided by Statsmodels (2018b) is a standard implementation which follows the maximum likelihood approach for binary choice above, simply assuming a single-index model $g(\boldsymbol{x}_i,\boldsymbol{\beta}) = \boldsymbol{x}_i\,\boldsymbol{\beta}$ with logistic error:

$$\begin{aligned}\mathcal{L}(\beta) &= \sum_{i=1}^{n}\left[y_i \ln\Lambda\left(\boldsymbol{x}_i\,\boldsymbol{\beta}\right) + (1-y_i)\ln\Lambda\left(-\,\boldsymbol{x}_i\,\boldsymbol{\beta}\right)\right],\\ \frac{\partial\mathcal{L}(\beta)}{\partial\,\boldsymbol{\beta}} &= \sum_{i=1}^{n}\left[y_i - \Lambda\left(\boldsymbol{x}_i\,\boldsymbol{\beta}\right)\right]\boldsymbol{x}_i\,.\end{aligned} \tag{III.12}$$

Expectations and marginal effects also follow those above:

$$\mathrm{E}[y_i|\,\boldsymbol{x}_i] = F_u(\boldsymbol{x}_i\,\boldsymbol{\beta}), \qquad \frac{\partial\,\mathrm{E}[y_i|\,\boldsymbol{x}_i]}{\partial x_{i,p}} = F_u'(\boldsymbol{x}_i\,\boldsymbol{\beta})\beta_p. \tag{III.13}$$

### III.3.3. Polynomial expansions

I implemented a polynomial expansion of the features for all linear methods. This replaces the original $\boldsymbol{x}_i$ with a second-degree polynomial expansion to form a new set of input features

$$\tilde{\boldsymbol{x}_i} = \begin{pmatrix} x_{i,1} & \ldots & x_{i,k} & x_{i,1}^2 & \ldots & x_{i,k}^2 & x_{i,1}x_{i,2} & \ldots x_{i,k-1}x_{i,k}\end{pmatrix}. \tag{III.14}$$

This add squared and interaction terms. However, all models remain linear in parameters, just with more features. Estimation hence follows the exact same procedure as above, simply using $\tilde{\boldsymbol{x}}$ instead of $\boldsymbol{x}$. Expectations are also calculated using the same formulas, which will then use the following linear signal instead

$$\tilde{x_1}\tilde{\boldsymbol{\beta}} = \beta_1^{(1)}x_{i,1} + \ldots + \beta_k^{(1)}x_{i,k} + \beta_1^{(2)}x_{i,1}^2 + \ldots + \beta_k^{(2)}x_{i,k}^2 + \beta_{1,2}^{(I)}x_{i,1}x_{i,2} + \ldots + \beta_{k-1,k}^{(I)}x_{i,k-1}x_{i,k}. \tag{III.15}$$

Marginal effects differ here, since a change in $x_{i,p}$ now also affect the squared term $x_{i,p}$, as well as all interaction terms featuring the variable. All in all, changing the parameter leads to

$$\frac{\partial \tilde{x}_i \tilde{\boldsymbol{\beta}}}{\partial x_{i,p}} = \beta_p^{(1)} + 2\beta_p^{(2)} x_{i,p} + \sum_{q \neq p} \beta_{p,q}^{(I)} x_{i,q}. \tag{III.16}$$

Estimation procedures for OLS and logistic regression which uses a polynomial expansion and properly adjusts the marginal effects are implemented in the code file *estimators.py*.

### III.4. IV estimators

All IV estimators are implemented using a simple two-stage procedure, as specified in section 6.3. 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 III.3.** Two-stage IV procedure as described in section 6.3.
**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.

Estimation and calculation of expectation and marginal effect in each stage depends on the chosen estimation procedure, but simply follows the procedures described in the sections above. In practice, the two procedures employed where OLS and neural networks.

The estimates from classical two-stage least squares (2SLS) are typically motivated by a similar procedure, except that one replaces $\boldsymbol{x}_i$ with $\hat{\boldsymbol{x}}_i$ in the second stage, rather than adding $\boldsymbol{u}_{x,i}$ (although in practice, the resulting exactly identified estimator can be obtained in a single step). Such an estimator can be seen in most undergraduate texts (see e.g. Wooldridge (2015)). Straightforward extension of this algorithm to non-linear methods leads to an inconsistent estimator (Cameron & Trivedi, 2005, p. 199), which was also reaffirmed in initial simulations, which lead to decent estimates of the average effect, but quite poor estimates for initial effects. Instead, the above algorithm was employed, following the approach to non-parametric IV in Newey *et al.* (1999) (although they use series approximations rather than neural networks). Due to a few basic mechanics of the OLS residuals, it yields basic 2SLS if we employ OLS in both steps. However, for other combinations, this approach yields a different solution.

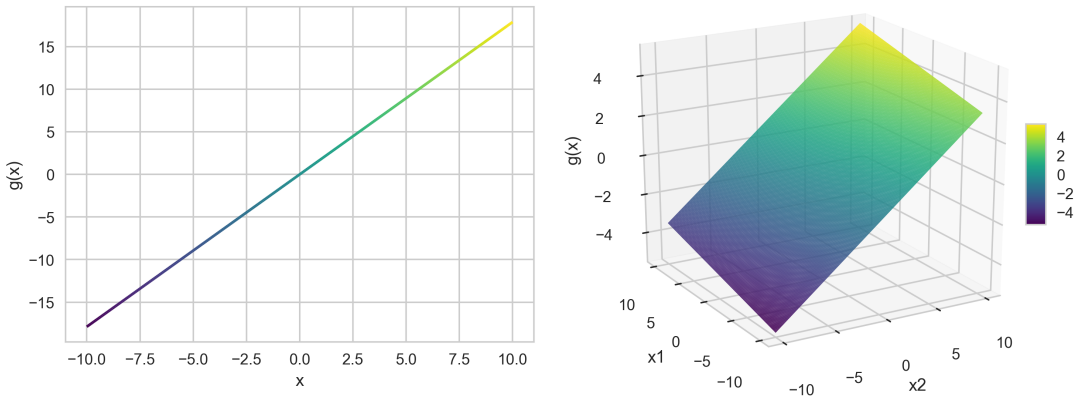
# IV. Data-generating processes

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.

## IV.1. Linear model (OLS/logistic regression)

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

**Figure IV.1:** *Linear function*



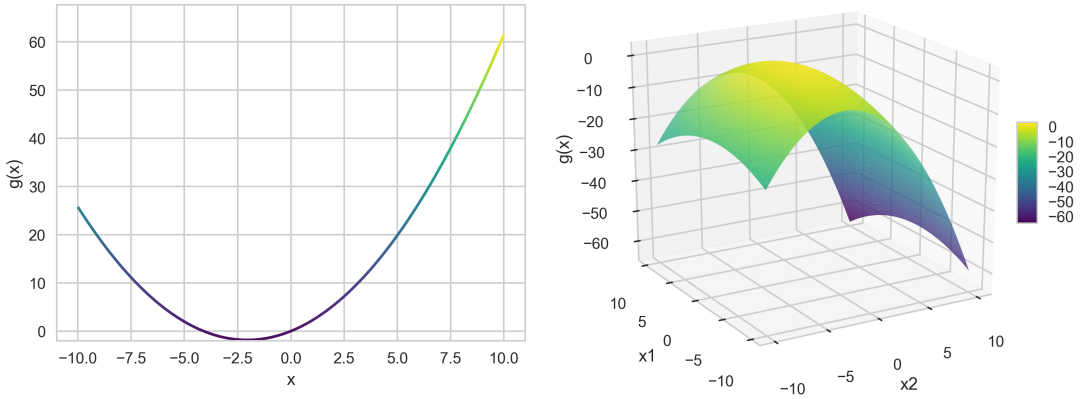

## IV.2. Polynomial function(s)

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

$$g(\boldsymbol{x}_i, \boldsymbol{\beta}) = \sum_{q=1}^{o} \sum_{p=1}^{k} \beta_{p,q} x_{i,p}^{q}, \qquad \frac{\partial g(\boldsymbol{x}_i, \boldsymbol{\beta})}{\partial x_{i,p}} = \sum_{q=1}^{o} q \beta_{p,q} x_{i,p}^{(q-1)}, \qquad \frac{\partial g(\boldsymbol{x}_i, \boldsymbol{\beta})}{\partial \beta_{p,q}} = x_{i,p}^{q}. \tag{IV.2}$$

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

**Figure IV.2:** *Adding quadratic terms*



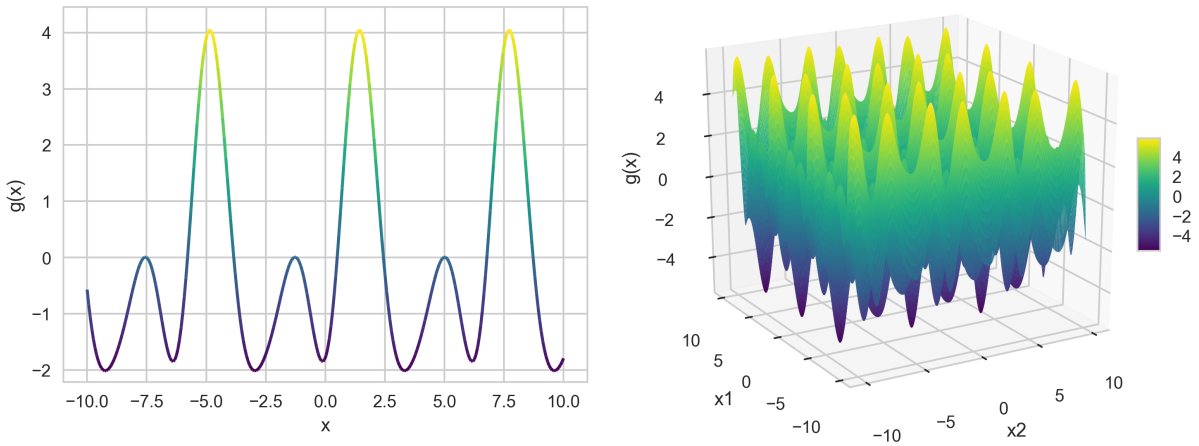
### IV.3. Trigonometric polynomials

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 that of *trigonometric polynomials* (which can be written as a polynomial of $\sin(\boldsymbol{x}_i)$ and $\cos(\boldsymbol{x}_i)$):

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

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 IV.3:** *Trigonometric polynomial of third order*



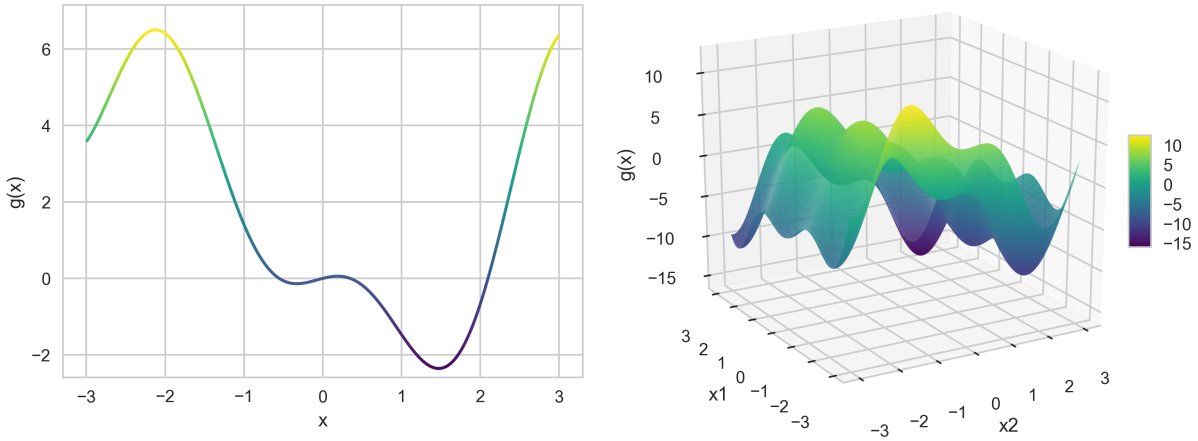

### IV.4. Wiggle, wiggle, wiggle

The following function aim to recreate a quintessentally 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).

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

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

**Figure IV.4:** *How do you fit all that, in them figures?*



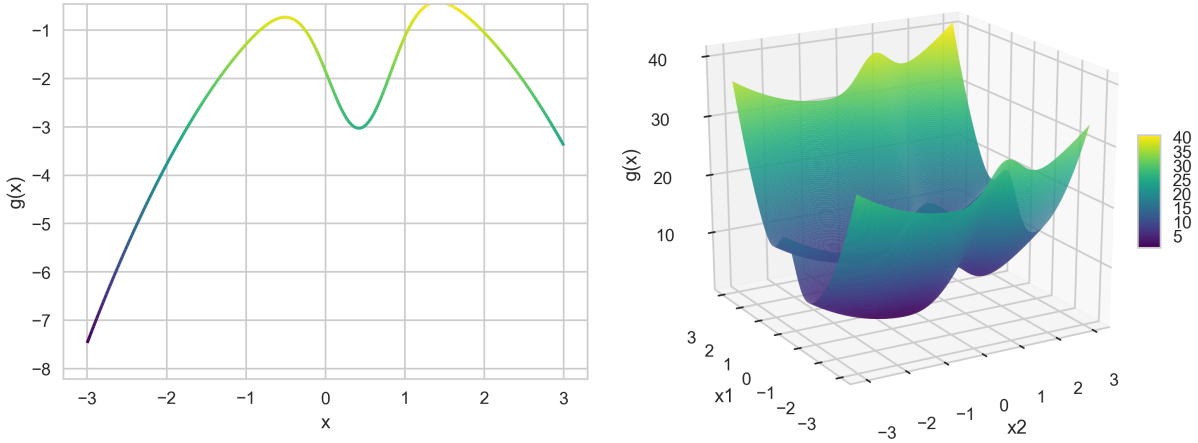

## IV.5. 'Pointy' function

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).

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

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

**Figure IV.5:** *Pointy function*



### IV.6. Ackley function

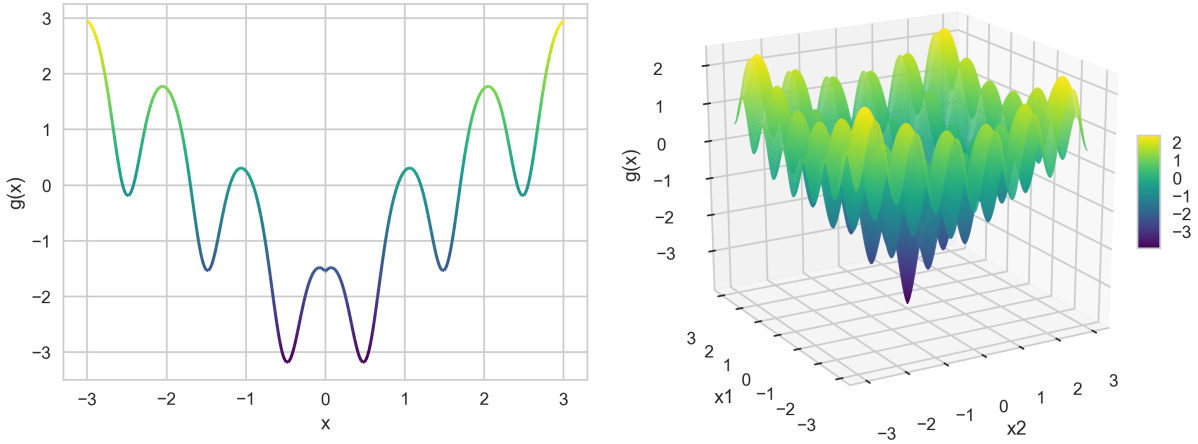
The Ackley function is characterized by a relatively flat outer region, and a sudden drop in the middle (Surjanovic & Bingham, 2017). In the traditional version (a=1, f=1), the minimum is at zero. Reducing $d$ allows us to obtain both positive and negative values.

$$
\begin{aligned}
g(\boldsymbol{x}_i, \boldsymbol{\beta}) &= a\left(-b\exp\left(-c\sqrt{\frac{1}{k}\sum_{p=1}^{k}|\beta_{p,1}|x_{i,p}^2}\right) - d\exp\left(\frac{1}{k}\sum_{p=1}^{k}\beta_{p,2}\cos(ex_{i,p})\right) + f\left(b+\exp(1)\right)\right), \\
\frac{\partial g(\boldsymbol{x}_i, \boldsymbol{\beta})}{\partial x_{i,p}} &= a\left(bc\frac{\exp\left(-c\sqrt{\frac{1}{k}\sum_{p=1}^{k}|\beta_{p,1}|x_{i,p}^2}\right)}{k\sqrt{\frac{1}{k}\sum_{p=1}^{k}|\beta_{p,1}|x_{i,p}^2}}|\beta_{p,1}|x_{i,p} + \frac{de}{k}\exp\left(\frac{1}{k}\sum_{p=1}^{k}\beta_{p,2}\cos(ex_{i,p})\right)\beta_{p,2}\sin(ex_{i,p})\right).
\end{aligned}
\tag{IV.6}
$$

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

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

**Figure IV.6:** *Ackley function*



## IV.7. Rastrigin function

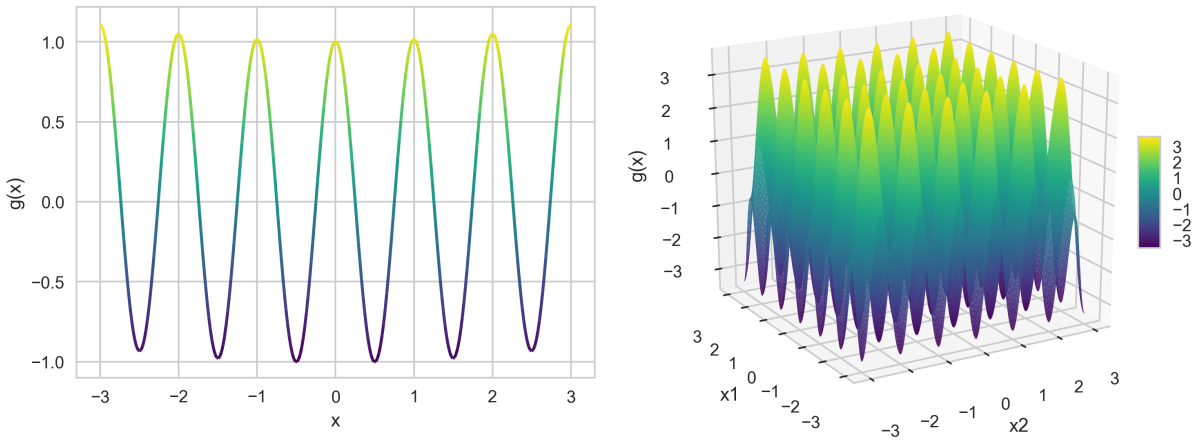
The Rastrigin function is highly multimodal, with many regularly distributed local minima, which in the binary cases yields a spotted pattern of successes and failures (Surjanovic & Bingham, 2017). In its regular form the minimum is at zero, but removing an added constant yields both positive and negative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

$$\begin{aligned} g(\boldsymbol{x}_i, \boldsymbol{\beta}) &= a\left(\sum_{p=1}^{k}\left(b\beta_{p,1}x_{i,p}^2 - c\beta_{p,2}\cos(2\pi x_{i,p})\right)\right), \\ \frac{\partial g(\boldsymbol{x}_i, \boldsymbol{\beta})}{\partial x_{i,p}} &= a\left(2b\beta_{p,1}x_{i,p} + 2\pi c\beta_{p,2}\sin(2\pi x_{i,p})\right). \end{aligned} \tag{IV.8}$$

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

$$\frac{\partial g(\boldsymbol{x}_i, \boldsymbol{\beta})}{\partial \beta_{p,1}} = abx_{i,p}^2, \quad \frac{\partial g(\boldsymbol{x}_i, \boldsymbol{\beta})}{\partial \beta_{p,2}} = -ac\cos(2\pi x_{i,p}). \tag{IV.9}$$

**Figure IV.7:** *Rastrigin function*




## IV.8. Drop-Wave function

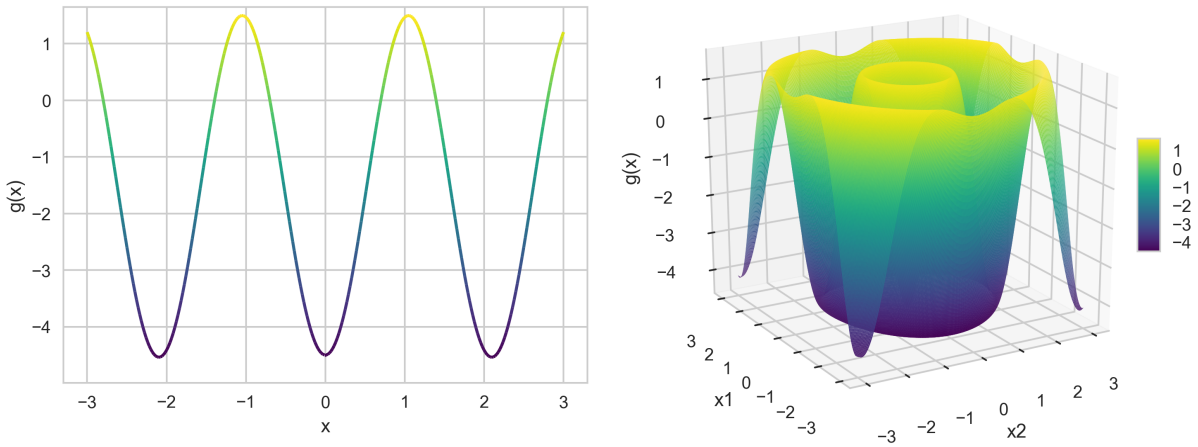
The Drop-wave function is once again complex and multimodal, and leads to shape of waves emerging from a center (Surjanovic & Bingham, 2017). Hence the name.

$$\begin{aligned} g(\boldsymbol{x}_i, \boldsymbol{\beta}) &= a\left(-\frac{b + \cos\left(c\sqrt{\sum_{p=1}^{k}|\beta_{p,1}|x_{i,p}^2}\right)}{d + e\sum_{p=1}^{k}\beta_{p,2}x_{i,p}^2} + f\right), \\ \frac{\partial g(\boldsymbol{x}_i, \boldsymbol{\beta})}{\partial x_{i,p}} &= \frac{ac\sin\left(c\sqrt{\sum_{p=1}^{k}|\beta_{p,1}|x_{i,p}^2}\right)}{\sqrt{\sum_{p=1}^{k}|\beta_{p,1}|x_{i,p}^2}\left(d + e\sum_{p=1}^{k}\beta_{p,2}x_{i,p}^2\right)}|\beta_{p,1}|x_{i,p} \\ &\quad + \frac{2ae\left(b + \cos\left(c\sqrt{\sum_{p=1}^{k}|\beta_{p,1}|x_{i,p}^2}\right)\right)}{\left(d + e\sum_{p=1}^{k}\beta_{p,2}x_{i,p}^2\right)^2}\beta_{p,2}x_{i,p}. \end{aligned} \tag{IV.10}$$

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.

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

**Figure IV.8:** *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.

# V. Derivations & proofs

## V.1. Proof of proposition 2.1

The overall logic relies on Imbens (2000). The proposition basically states that observed changes in the conditional expectation reflects what would occur if we induced an exogenous change in $x_{i,p}$. This is ensured by the assumption of weak unconfoundedness. Start from the observed conditional expectation. I may directly replace the conditioning on $x_{i,p}$ with the assignment variable $D(x) = 1\{x_{i,p} = x\}$, and replace the observed $y_i$ with the realized potential outcome. Since assignment is independent of potential outcomes, I may then drop $D(x_{i,p})$.

$$\begin{aligned}
\mathrm{E}[y_i|x_{i,p}, \boldsymbol{x}_{i,-p}] &= \mathrm{E}[y_i|D(x_{i,p}) = 1, \boldsymbol{x}_{i,-p}] && \text{(V.1)}\\
&= \mathrm{E}[f_i(x_i, p)|D(x_{i,p}) = 1, \boldsymbol{x}_{i,-p}] && \text{(V.2)}\\
&= \mathrm{E}[f_i(x_i, p)|\, \boldsymbol{x}_{i,-p}]. && \text{(V.3)}
\end{aligned}$$

This still depends on $\boldsymbol{x}_{i,-p}$. There are two possible cases. Either $\boldsymbol{x}_{i,-p}$ does not affect the effect of $x_{i,p}$. In this case, I could drop the conditioning. Alternatively, the effect does depend on $\boldsymbol{x}_{i,-p}$. In this case, this must be included in the potential outcomes $f_i(x_{i,p})$, so that I could write it $f_i(x_{i,p}, \boldsymbol{x}_{i,-p})$. In this case, conditioning allows us to correctly reflect $f_i(x_{i,p})$, because the observed change reflects an exogenous change only if I include the covariates:

$$\gamma_i(x_{i,p}) = \frac{\partial\, \mathrm{E}[f_i(x_{i,p})]}{\partial x_{i,p}} = \frac{\partial\, \mathrm{E}[y_i|x_{i,p}, \boldsymbol{x}_{i,-p}]}{\partial x_{i,p}}. \tag{V.4}$$

## V.2. Proof of proposition 2.2

The proof follows Wasserman (2016), but borrows slightly from an alternative proof provided in Angrist & Pischke (2008), to achieve overall elegance. Let $m(\boldsymbol{x}_i)$ be any function of $\boldsymbol{x}_i$.

$$\begin{aligned}
\mathrm{E}\left[(y_i - m(\boldsymbol{x}_i))^2\right] &= \mathrm{E}\left[(y_i - \mathrm{E}[y_i|\, \boldsymbol{x}_i] + \mathrm{E}[y_i|\, \boldsymbol{x}_i] - m(\boldsymbol{x}_i))^2\right] && \text{(V.5)}\\
&= \mathrm{E}\left[(y_i - \mathrm{E}[y_i|\, \boldsymbol{x}_i])^2\right] + \mathrm{E}\left[(\mathrm{E}[y_i|\, \boldsymbol{x}_i] - m(\boldsymbol{x}_i))^2\right] && \text{(V.6)}\\
&\quad + 2\,\mathrm{E}\left[(y_i - \mathrm{E}[y_i|\, \boldsymbol{x}_i])\,(\mathrm{E}[y_i|\, \boldsymbol{x}_i] - m(\boldsymbol{x}_i))\right]. && \text{(V.7)}
\end{aligned}$$

The last term yields zero by the law of iterated expectations. Note that only $y_i$ is affected by taking the expectation, since remaining terms are deterministic.

$$\begin{aligned}
\mathrm{E}\left[(y_i - m(\boldsymbol{x}_i))^2\right] &= \mathrm{E}\left[(y_i - \mathrm{E}[y_i|\, \boldsymbol{x}_i])^2\right] + \mathrm{E}\left[(\mathrm{E}[y_i|\, \boldsymbol{x}_i] - m(\boldsymbol{x}_i))^2\right] && \text{(V.8)}\\
&\quad + 2\,\mathrm{E}\left[\mathrm{E}\left[(y_i - \mathrm{E}[y_i|\, \boldsymbol{x}_i])\,(\mathrm{E}[y_i|\, \boldsymbol{x}_i] - m(\boldsymbol{x}_i))\right]|\, \boldsymbol{x}_i\right] && \text{(V.9)}\\
&= \mathrm{E}\left[(y_i - \mathrm{E}[y_i|\, \boldsymbol{x}_i])^2\right] + \mathrm{E}\left[(\mathrm{E}[y_i|\, \boldsymbol{x}_i] - m(\boldsymbol{x}_i))^2\right] && \text{(V.10)}\\
&\quad + 2\,\mathrm{E}\left[(\mathrm{E}[y_i|\, \boldsymbol{x}_i] - \mathrm{E}[y_i|\, \boldsymbol{x}_i])\,(\mathrm{E}[y_i|\, \boldsymbol{x}_i] - m(\boldsymbol{x}_i))\right] && \text{(V.11)}\\
&= \mathrm{E}\left[(y_i - \mathrm{E}[y_i|\, \boldsymbol{x}_i])^2\right] + \mathrm{E}\left[(\mathrm{E}[y_i|\, \boldsymbol{x}_i] - m(\boldsymbol{x}_i))^2\right]. && \text{(V.12)}
\end{aligned}$$

The first term does not depend on $m(\boldsymbol{x}_i)$. The second is minimized at zero when $m(\boldsymbol{x}_i) = \mathrm{E}[y_i|\, \boldsymbol{x}_i]$.

### V.3. Proof of proposition 2.3

The proof follows Wasserman (2016). Let $m(\boldsymbol{x}_i)$ be any function of $\boldsymbol{x}_i$ and let $h(\boldsymbol{x}_i) = 1\left(m(\boldsymbol{x}_i) \geq 0.5\right)$ be the accompanying classifier.

$$\mathrm{P}\left(y_i \neq h\left(\boldsymbol{x}_i\right)\right) - \mathrm{P}\left(y_i \neq h\left(\mathrm{E}[y_i|\, \boldsymbol{x}_i]\right)\right) = h\left(\boldsymbol{x}_i\right)\mathrm{P}\left(y_i \neq 1|\, \boldsymbol{x}_i\right) + \left(1 - h\left(\boldsymbol{x}_i\right)\right)\mathrm{P}\left(y_i \neq 0|\, \boldsymbol{x}_i\right) \tag{V.13}$$

$$- h\left(\mathrm{E}[y_i|\, \boldsymbol{x}_i]\right)\mathrm{P}\left(y_i \neq 1|\, \boldsymbol{x}_i\right) + \left(1 - h\left(\mathrm{E}[y_i|\, \boldsymbol{x}_i]\right)\right)\mathrm{P}\left(y_i \neq 0|\, \boldsymbol{x}_i\right) \tag{V.14}$$

$$= h\left(\boldsymbol{x}_i\right)\left(1 - \mathrm{E}[y_i|\, \boldsymbol{x}_i]\right) + \left(1 - h\left(\boldsymbol{x}_i\right)\right)\mathrm{E}[y_i|\, \boldsymbol{x}_i] \tag{V.15}$$

$$- h\left(\mathrm{E}[y_i|\, \boldsymbol{x}_i]\right)\left(1 - \mathrm{E}[y_i|\, \boldsymbol{x}_i]\right) + \left(1 - h\left(\mathrm{E}[y_i|\, \boldsymbol{x}_i]\right)\right)\mathrm{E}[y_i|\, \boldsymbol{x}_i] \tag{V.16}$$

$$= 2\left(\mathrm{E}[y_y|\, \boldsymbol{x}_i] - 0.5\right)\left(h\left(\mathrm{E}[y_i|\, \boldsymbol{x}_i]\right) - h\left(\boldsymbol{x}_i\right)\right) \tag{V.17}$$

$$\geq 0. \tag{V.18}$$

The inequality follows from that $h\left(\mathrm{E}[y_i|\, \boldsymbol{x}_i]\right) = 1$ if and only if $\mathrm{E}[y_i|\, \boldsymbol{x}_i] \geq 0.5$. Any classifier not based on the conditional expectation has greater classification error than one that is.

### V.4. Proof of proposition 2.5

The proof is inspired by Angrist & Pischke (2008). Start from the result in appendix V.2, but restrict to linear functions $m(\boldsymbol{x}_i) = \boldsymbol{x}_i\,\boldsymbol{b}$.

$$\mathrm{E}\left[\left(y_i - \boldsymbol{x}_i\,\boldsymbol{b}\right)^2\right] = \mathrm{E}\left[\left(y_i - \mathrm{E}[y_i|\, \boldsymbol{x}_i]\right)^2\right] + \mathrm{E}\left[\left(\mathrm{E}[y_i|\, \boldsymbol{x}_i] - \boldsymbol{x}_i\,\boldsymbol{b}\right)^2\right]. \tag{V.19}$$

The first term does not depend on $\boldsymbol{x}_i\,\boldsymbol{b}$, and may be ignored. To minimize the second term I solve a familiar first order condition, which provides the result.

$$\mathrm{E}\left[\boldsymbol{x}_i\left(\mathrm{E}[y_i|\, \boldsymbol{x}_i] - \boldsymbol{x}_i\,\boldsymbol{b}\right)\right] = 0 \quad \Leftrightarrow \quad \boldsymbol{b} = \mathrm{E}\left[\boldsymbol{x}_i\,\boldsymbol{x}_i'\right]^{-1}\mathrm{E}[\boldsymbol{x}_i\,y_i|\, \boldsymbol{x}_i] = \boldsymbol{\beta}\,. \tag{V.20}$$

### V.5. Derivation of proposition 3.1

Start from the expression for neural network output from definition 3.1. Condense the linear signals as $s_{i,j}^{(1)} = \beta_{0,j}^{(1)} + \sum_{p=1}^{k}\beta_{j,p}^{(1)}x_{i,k}$ and $s_i^{(2)} = \beta_0^{(2)} + \sum_{j=1}^{J}\beta_j^{(2)}\eta\left(s_{i,j}^{(1)}\right)$.

$$\tilde{f}(\boldsymbol{x}_i) = \sigma\left(\beta_0^{(2)} + \sum_{j=1}^{J}\beta_j^{(2)}\eta\left(\beta_{0,j}^{(1)} + \sum_{p=1}^{k}\beta_{j,p}^{(1)}x_{i,k}\right)\right) \tag{V.21}$$

$$= \sigma\left(\beta_0^{(2)} + \sum_{j=1}^{J}\beta_j^{(2)}\eta\left(s_{i,j}^{(1)}\right)\right) = \sigma\left(s_i^{(2)}\right). \tag{V.22}$$

The proposition follows from successive application of the chain rule:

$$\frac{\partial\tilde{f}(\boldsymbol{x}_i)}{\partial x_{i,p}} = \frac{\partial\sigma\left(s_i^{(2)}\right)}{\partial\boldsymbol{s}_i^{(2)}}\cdot\frac{\partial s_i^{(2)}\left(\boldsymbol{s}_i^{(1)}\right)}{\partial x_{i,p}} \tag{V.23}$$

$$= \sigma'\left(s_i^{(2)}\right)\left(\sum_{j=1}^{J}\beta_j^{(2)}\frac{\partial\eta\left(s_{i,j}^{(1)}\right)}{\partial x_{i,p}}\right) \tag{V.24}$$

$$= \sigma'\left(s_i^{(2)}\right)\left(\sum_{j=1}^{J}\beta_j^{(2)}\frac{\partial\eta\left(s_{i,j}^{(1)}\right)}{\partial s_{i,k}^{(1)}}\cdot\frac{\partial s_{i,j}^{(1)}}{\partial x_{i,p}}\right) \tag{V.25}$$

$$= \sigma'\left(s_i^{(2)}\right)\left(\sum_{j=1}^{J}\beta_j^{(2)}\eta'\left(s_{i,j}^{(1)}\right)\beta_{j,p}^{(1)}\right). \tag{V.26}$$

This uses the standard exposition that e.g. $\sigma'(s) = \partial\sigma(s)/\partial s$.

Consider the matrix notation instead. Let $\boldsymbol{s}_i^{(1)} = \boldsymbol{x}_i\,\boldsymbol{\beta}^{(1)}$ and $s_i^{(2)} = \eta(s_i^{(1)})\,\boldsymbol{\beta}^{(2)}$

$$\tilde{f}(\boldsymbol{x}_i) = \sigma\left(\eta\left(\boldsymbol{x}_i\,\boldsymbol{\beta}^{(1)}\right)\boldsymbol{\beta}^{(2)}\right) \tag{V.27}$$

$$= \sigma\left(\eta\left(\boldsymbol{s}_i^{(1)}\right)\boldsymbol{\beta}^{(2)}\right) = \sigma\left(s_i^{(2)}\right). \tag{V.28}$$

Once again, iteratively apply the chain rule:

$$\frac{\partial\tilde{f}(\boldsymbol{x}_i)}{\partial x_{i,p}} = \frac{\partial\sigma\left(s_i^{(2)}\right)}{\partial s_i^{(2)}}\cdot\frac{\partial s_i^{(2)}\left(\boldsymbol{s}_i^{(1)}\right)}{\partial x_{i,p}} \tag{V.29}$$

$$= \sigma'\left(s_i^{(2)}\right)\left(\frac{\partial\eta\left(\boldsymbol{s}_i^{(1)}\right)}{\partial x_{i,p}}\,\boldsymbol{\beta}_{1:}^{(2)}\right) \tag{V.30}$$

$$= \sigma'\left(s_i^{(2)}\right)\left(\frac{\partial\eta\left(\boldsymbol{s}_i^{(1)}\right)}{\partial\,\boldsymbol{s}_i^{(1)}}\mathrm{diag}\left(\frac{\partial\,\boldsymbol{s}_i^{(1)}}{\partial x_{i,p}}\right)\boldsymbol{\beta}_{1:}^{(2)}\right) \tag{V.31}$$

$$= \underbrace{\sigma'\left(s_i^{(2)}\right)}_{1\times1}\underbrace{\eta'\left(\boldsymbol{s}_i^{(1)}\right)}_{1\times J}\underbrace{\mathrm{diag}\left(\boldsymbol{\beta}_p^{(1)}\right)}_{J\times J}\underbrace{\boldsymbol{\beta}_{1:}^{(2)}}_{J\times1}. \tag{V.32}$$

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 $\eta'(s_{i,k}^{(1)})$ is multiplied by its own derivative, as we'd like. $\boldsymbol{\beta}_{1:}^{(2)}$ excludes the intercept $\beta_0^{(2)}$ of $\boldsymbol{\beta}^{(2)}$, because it is accompanied by a zero in the derivative.

## V.6. Derivation of proposition 3.2

The derivation follows that of appendix V.5. The start and end steps are the same - there are simply a few more steps in between. Start from definition 3.2:

$$\tilde{f}(\boldsymbol{x}_i) = \sigma\left(\beta_0^{(L)} + \sum_{j_{L-1}=1}^{J^{(L-1)}}\beta_{j_{L-1}}^{(L)}\eta\left(\ldots\eta\left(\beta_{j_2,0}^{(2)} + \sum_{j_1=1}^{J^{(1)}}\beta_{j_2,j_1}^{(2)}\eta\left(\beta_{j_1,0}^{(1)} + \sum_{p=1}^{k}\beta_{j_1,p}^{(1)}x_{i,k}\right)\right)\right)\right). \tag{V.33}$$

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:

$$\frac{\partial \tilde{f}(\boldsymbol{x}_i)}{\partial x_{i,p}} = \frac{\partial \sigma\left(s_i^{(L)}\right)}{\partial s_i^{(L)}} \cdot \frac{\partial s_i^{(L)}\left(\boldsymbol{s}_i^{(L-1)}\right)}{\partial x_{i,p}} \tag{V.34}$$

$$= \sigma'\left(s_i^{(L)}\right) \left( \sum_{j_{L-1}=1}^{J^{(L-1)}} \beta_{j_{L-1}}^{(L)} \frac{\partial \eta(s_{i,j_{L-1}}^{(L-1)})}{\partial x_{i,p}} \right) \tag{V.35}$$

$$= \sigma'\left(s_i^{(L)}\right) \left( \sum_{j_{L-1}=1}^{J^{(L-1)}} \beta_{j_{L-1}}^{(L)} \frac{\partial \eta(s_{i,j_{L-1}}^{(L-1)})}{\partial s_{i,j_{L-1}}^{(L-1)}} \cdot \frac{\partial s_{i,j_{L-1}}^{(L-1)}}{\partial x_{i,p}} \right) \tag{V.36}$$

$$= \sigma'\left(s_i^{(L)}\right) \left( \sum_{j_{L-1}=1}^{J^{(L-1)}} \beta_{j_{L-1}}^{(L)} \eta'\left(\boldsymbol{s}_i^{(L-1)}\right) \cdot \ldots \cdot \frac{\partial \eta(s_{i,j_2}^{(2)})}{\partial x_{i,p}} \right) \tag{V.37}$$

$$= \sigma'\left(s_i^{(L)}\right) \left( \sum_{j_{L-1}=1}^{J^{(L-1)}} \beta_{j_{L-1}}^{(L)} \eta'\left(\boldsymbol{s}_i^{(L-1)}\right) \cdot \ldots \cdot \frac{\partial \eta(s_{i,j_2}^{(2)})}{\partial s_{i,j_2}^{(2)}} \cdot \frac{\partial s_{i,j_2}^{(2)}}{\partial x_{i,p}} \right) \tag{V.38}$$

$$= \sigma'\left(s_i^{(L)}\right) \left( \sum_{j_{L-1}=1}^{J^{(L-1)}} \beta_{j_{L-1}}^{(L)} \eta'\left(\boldsymbol{s}_i^{(L-1)}\right) \ldots \eta'\left(\boldsymbol{s}_i^{(2)}\right) \left( \sum_{j_1=1}^{J^{(1)}} \beta_{j_2,j_1}^{(2)} \frac{\partial \eta\left(s_{i,j}^{(1)}\right)}{\partial x_{i,p}} \right) \right) \tag{V.39}$$

$$= \sigma'\left(s_i^{(L)}\right) \left( \sum_{j_{L-1}=1}^{J^{(L-1)}} \beta_{j_{L-1}}^{(L)} \eta'\left(\boldsymbol{s}_i^{(L-1)}\right) \ldots \eta'\left(\boldsymbol{s}_i^{(2)}\right) \left( \sum_{j_1=1}^{J^{(1)}} \beta_{j_2,j_1}^{(2)} \frac{\partial \eta\left(s_{i,j}^{(1)}\right)}{\partial s_{i,j}^{(1)}} \frac{\partial s_{i,j}^{(1)}}{\partial x_{i,p}} \right) \right) \tag{V.40}$$

$$= \sigma'\left(s_i^{(L)}\right) \left( \sum_{j_{L-1}=1}^{J^{(L-1)}} \beta_{j_{L-1}}^{(L)} \eta'\left(\boldsymbol{s}_i^{(L-1)}\right) \ldots \eta'\left(\boldsymbol{s}_i^{(2)}\right) \left( \sum_{j_1=1}^{J^{(1)}} \beta_{j_2,j_1}^{(2)} \eta'\left(s_{i,j_1}^{(1)}\right) \beta_{j_1,p}^{(1)} \right) \right). \tag{V.41}$$

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.

Enter instead the matrix:

$$\tilde{f}(\boldsymbol{x}_i) = \sigma\left(\eta\left(\ldots \eta\left(\eta\left(\boldsymbol{x}_i \boldsymbol{\beta}^{(1)}\right) \boldsymbol{\beta}^{(2)}\right) \ldots\right) \boldsymbol{\beta}^{(L)}\right). \tag{V.42}$$

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

$$\frac{\partial \tilde{f}(\boldsymbol{x}_i)}{\partial x_{i,p}} = \frac{\partial \sigma\left(s_i^{(L)}\right)}{\partial s_i^{(L)}} \cdot \frac{\partial s_i^{(L)}\left(\boldsymbol{s}_i^{(L-1)}\right)}{\partial x_{i,p}} \tag{V.43}$$

$$= \sigma'\left(s_i^{(L)}\right) \left( \frac{\partial \eta\left(\boldsymbol{s}_i^{(L-1)}\right)}{\partial x_{i,p}} \boldsymbol{\beta}_{1:}^{(L)} \right) \tag{V.44}$$

$$= \sigma'\left(s_i^{(L)}\right) \left( \frac{\partial \eta\left(\boldsymbol{s}_i^{(L-1)}\right)}{\partial \boldsymbol{s}_i^{(L-1)}} \mathrm{diag}\left( \frac{\partial \boldsymbol{s}_i^{(L-1)}}{\partial x_{i,p}} \right) \boldsymbol{\beta}_{1:}^{(L)} \right) \tag{V.45}$$

$$= \sigma'\left(s_i^{(L)}\right) \left( \eta'\left(\boldsymbol{s}_i^{()}\right) \mathrm{diag}\left( \ldots \mathrm{diag}\left( \frac{\partial \eta\left(\boldsymbol{s}_i^{(1)}\right)}{\partial x_{i,p}} \right) \boldsymbol{\beta}_{1:}^{(2)} \ldots \right) \boldsymbol{\beta}_{1:}^{(L)} \right) \tag{V.46}$$

$$= \sigma'\left(s_i^{(L)}\right)\left(\eta'\left(\boldsymbol{s}_i^{(1)}\right)\operatorname{diag}\left(\ldots\operatorname{diag}\left(\frac{\partial\eta\left(\boldsymbol{s}_i^{(1)}\right)}{\partial\,\boldsymbol{s}_i^{(1)}}\operatorname{diag}\left(\frac{\partial\,\boldsymbol{s}_i^{(1)}}{\partial x_{i,p}}\right)\right)\boldsymbol{\beta}_{1:}^{(2)}\ldots\right)\boldsymbol{\beta}_{1:}^{(L)}\right) \tag{V.47}$$

$$= \underbrace{\sigma'\left(s_i^{(L)}\right)}_{1\times1}\underbrace{\eta'\left(\boldsymbol{s}_i^{(L-1)}\right)}_{1\times J^{(L-1)}}\operatorname{diag}\underbrace{\left(\ldots\operatorname{diag}\left(\underbrace{\underbrace{\eta'\left(\boldsymbol{s}_i^{(1)}\right)}_{1\times J^{(1)}}\underbrace{\operatorname{diag}\left(\boldsymbol{\beta}_p^{(1)}\right)}_{J^{(1)}\times J^{(1)}}\underbrace{\boldsymbol{\beta}_{1:}^{(2)}}_{J^{(1)}\times J^{(2)}}}_{J^{(2)}\times J^{(2)}}\right)\ldots\right)}_{J^{(L-1)}\times J^{(L-1)}}\underbrace{\boldsymbol{\beta}_{1:}^{(L)}}_{J^{(L-1)}\times1}. \tag{V.48}$$

The mechanics of adding the diagonal is explained in more detail in appendix V.5, but it is really just that multiplying with the diagonal corresponds to element-wise multiplication, here with the inner derivative as prescribed by the chain rule. For all weights, the subscript indicates that I drop the intercept node, because it is unaffected by changes in $\boldsymbol{x}_i$.

## V.7. Proof of proposition 5.1

The proof follows p. 110 in Rao (1973). The proposition follows directly from a version of Chebyshev's inequality (Rao, 1973, 95), which states that for any a random variable $\hat{\theta}$, constants $\theta$ and $\epsilon > 0$:

$$\mathrm{P}\left(\left|\hat{\theta}-\theta\right|>\epsilon\right)\leq\frac{1}{\epsilon^2}\,\mathrm{E}\left[\left(\hat{\theta}-\theta\right)^2\right]. \tag{V.49}$$

The right hand side goes to zero if MSE goes to zero. This provides a sufficient condition for convergence in probability (and hence consistency):

$$\lim_{n\to\infty}\mathrm{E}\left[\left(\hat{\theta}-\theta\right)^2\right]=0 \qquad \Rightarrow \qquad \lim_{n\to\infty}\mathrm{P}\left(\left|\hat{\theta}-\theta\right|<\epsilon\right)=0. \tag{V.50}$$

Note however that the condition is not necessary, since it is based only on an inequality.

## V.8. Derivation of proposition 5.2

Assume that a true value function $\theta(\boldsymbol{x}_i)$ exists, and that we are estimating an approximate function $\tilde{\theta}(\boldsymbol{x}_i)=\theta(\boldsymbol{x}_i)-\epsilon_A(\boldsymbol{x}_i)$ (which corresponds to the approximation error of the model as in definition 2.9), with an estimator $\hat{\theta}(\boldsymbol{x}_i)$. Suppress $(\boldsymbol{x}_i)$ for notational simplicity.

$$\mathrm{E}\left[\left(\hat{\theta}-\theta\right)^2\middle|\,\boldsymbol{x}_i\right]=\mathrm{E}\left[\hat{\theta}^2+\theta^2-2\hat{\theta}\theta\middle|\,\boldsymbol{x}_i\right] \tag{V.51}$$

$$=\mathrm{E}\left[\hat{\theta^2}\middle|\,\boldsymbol{x}_i\right]+\mathrm{E}\left[\theta^2\middle|\,\boldsymbol{x}_i\right]-2\,\mathrm{E}\left[\hat{\theta}\theta\middle|\,\boldsymbol{x}_i\right]. \tag{V.52}$$

$$\tag{V.53}$$

Use that for any stochastic variable $\xi$ we have $\mathrm{V}(\xi|\,\boldsymbol{x}_i) = \mathrm{E}[\xi^2|\,\boldsymbol{x}_i] - \mathrm{E}[\xi|\,\boldsymbol{x}_i]^2 \Leftrightarrow \mathrm{E}[\xi^2|\,\boldsymbol{x}_i] = \mathrm{V}(\xi|\,\boldsymbol{x}_i) + \mathrm{E}[\xi|\,\boldsymbol{x}_i]^2$. Use further that $\theta = \tilde{\theta} + \epsilon_A$ is deterministic so $\mathrm{E}[\theta|\,\boldsymbol{x}_i] = \theta$.

$$\mathrm{E}\left[\left(\hat{\theta}-\theta\right)^2 \middle|\, \boldsymbol{x}_i\right] = \mathrm{V}(\hat{\theta}|\,\boldsymbol{x}_i) + \mathrm{E}[\hat{\theta}|\,\boldsymbol{x}_i]^2 + \left(\tilde{\theta}+\epsilon_A\right)^2 - 2(\tilde{\theta}+\epsilon_A)\,\mathrm{E}[\hat{\theta}|\,\boldsymbol{x}_i] \tag{V.54}$$

$$= \mathrm{V}(\hat{\theta}|\,\boldsymbol{x}_i) + \left(\epsilon_A^2 + 2\tilde{\theta}\epsilon_A - 2\epsilon_A\,\mathrm{E}[\hat{\theta}|\,\boldsymbol{x}_i]\right) + \left(\mathrm{E}[\hat{\theta}|\,\boldsymbol{x}_i]^2 + \tilde{\theta}^2 - 2\tilde{\theta}\,\mathrm{E}[\hat{\theta}|\,\boldsymbol{x}_i]\right) \tag{V.55}$$

$$= \mathrm{V}(\hat{\theta}|\,\boldsymbol{x}_i) + \left(\mathrm{E}[\hat{\theta}|\,\boldsymbol{x}_i] - \tilde{\theta}\right)^2 + \left(\epsilon_A^2 + 2\tilde{\theta}\epsilon_A - 2\epsilon_A\,\mathrm{E}[\hat{\theta}|\,\boldsymbol{x}_i]\right) \tag{V.56}$$

$$= \mathrm{V}(\hat{\theta}|\,\boldsymbol{x}_i) + \mathrm{Bias}(\hat{\theta},\tilde{\theta}|\,\boldsymbol{x}_i)^2 + \tilde{\epsilon}_A(\boldsymbol{x}_i). \tag{V.57}$$

I may obtain the overall expectation by the law of iterated expectations:

$$\mathrm{E}\left[\left(\hat{\theta}-\theta\right)^2\right] = \mathrm{E}\left[\mathrm{E}\left[\left(\hat{\theta}-\theta\right)^2 \middle|\, \boldsymbol{x}_i\right]\right] \tag{V.58}$$

$$= \mathrm{E}\left[\mathrm{V}(\hat{\theta}|\,\boldsymbol{x}_i)\right] + \mathrm{E}\left[\mathrm{Bias}(\hat{\theta},\tilde{\theta}|\,\boldsymbol{x}_i)^2\right] + \mathrm{E}\left[\tilde{\epsilon}_A(\boldsymbol{x}_i)\right] \tag{V.59}$$

$$= \text{Variance} + \text{Bias}^2 + \text{Approximation error}. \tag{V.60}$$

The final result follows the traditional notation of e.g. Friedman *et al.* (2009) or Abu-Mostafa *et al.* (2012a), but adds the term formally expressing approximation error. If there is no approximation error, I obtain the typical results. Notice that if $\epsilon_A \to 0$ then $\tilde{\epsilon_A} \to 0$. The theorems for neural nets argued that we could achieve arbitrarily low $\epsilon_A$ for all $\boldsymbol{x}_i$. Notice further that if the estimator is unbiased $\mathrm{E}[\hat{\theta}|\,\boldsymbol{x}_i] = \tilde{\theta}$ then $\tilde{\epsilon}_A = \epsilon_A^2$, so the MSE is variance plus squared approximation error.