Deep Fundamental Factor Models

Matthew F. Dixon*
Department of Applied Math
Illinois Institute of Technology

Nicholas G. Polson†
Booth School of Business
University of Chicago

March 2019

Abstract

Deep fundamental factor models are developed to capture non-linearity, interaction effects and non-parametric shocks in financial econometrics. Uncertainty quantification provides interpretability with interval estimation, ranking of factor importances and estimation of interaction effects. With no hidden layers we recover a linear factor model and for one or more hidden layers, uncertainty bands for the sensitivity to each input naturally arise from the network weights. Using 3290 assets in the Russell 3000 index over a period of December 1989 to January 2018, we assess a 49 factor model and generate information ratios that are approximately 2x greater than the OLS factor model. Furthermore, we compare our deep fundamental factor model with a quadratic LASSO model and demonstrate the superior performance and robustness to outliers.

Keywords: Deep learning, factor models, interpretability, financial econometrics

1 Introduction

In this paper, we present a framework for deep fundamental factor (DFF) models. The key aspect is a methodology for interpreting the deep learner which is applicable under mild restrictions on the network architecture. Our method explicitly identifies interaction effects and ranks the importance of the factors. The approach is applicable to many types of network architectures: fully-connected,

*Matthew Dixon is an Assistant Professor in the Department of Applied Math, Illinois Institute of Technology. E-mail: matthew.dixon@iit.edu.

†Nicholas Polson is Professor of Econometrics and Statistics at ChicagoBooth, University of Chicago. E-mail: ngp@chicagobooth.edu.

convolutional or recurrent layers, as well as for pooling operations. In the case when the network contains no hidden layers, we recover ordinary least squares estimators. For one or more hidden layers, we show how uncertainty bands for the sensitivity of the model to each input arise from the network weights. Moreover, for a certain choice of activation functions, we prove that such a distribution has bounded mean and variance, with variance bounds expressed as a function of the product of weights in the network.

Deep learning Deep learning applies hierarchical layers of hidden variables to construct nonlinear predictors which scale to high dimensional input space. The deep learning paradigm for data analysis is algorithmic rather than probabilistic (see Breiman (2001)). Deep learning has been shown to 'compress' the input space by projecting the input variables into a lower dimensional space using auto-encoders, as in deep portfolio theory (Heaton et al., 2017). A related approach introduced by Fan et al. (2017), referred to as *sufficient forecasting*, provides a set of sufficient predictive indices which are inferred from high-dimensional predictors. The approach uses projected principal component analysis under a semi-parametric factor model and has a direct correspondence with deep learning.

1.1 Why Deep Neural Networks?

Artificial neural networks have a long history in business and economic statistics. Building on the seminal work of Gallant and White (1988); Andrews (1989); Hornik et al. (1989), Swanson and White (1995); Kuan and White (1994); Lo (1994); Hutchinson et al. (1994); Baillie and Kapetanios (2007); Racine (2001) develop various studies in the finance, economics and business literature. Most recently, the literature has been extended to include deep neural networks (Sirignano et al., 2016; Dixon et al., 2016; Feng et al., 2018; Heaton et al., 2017; Chen et al., 2019).

It is well known that shallow neural networks are furnished with the universal representation theorem, which states that any shallow[1] feedforward neural network can represent all continuous functions (Hornik et al., 1989), *provided there are enough hidden units*. The extension to deep neural networks has only recently been motivated on theoretical grounds (Tishby and Zaslavsky, 2015; Poggio, 2016; Mhaskar et al., 2016; Martin and Mahoney, 2018; Bartlett et al., 2017). Poggio (2016) show that deep networks can achieve superior performance versus linear additive models, such as linear regression, while avoiding the curse of dimensionality. Martin and Mahoney (2018) show that deep networks are implicitly self-regularizing and Tishby and Zaslavsky (2015) characterizes the layers as 'statistically decoupling' the input variables.

There are additionally many recent theoretical developments which characterize the approximation behavior as a function of network depth, width and

[1] While there is no consensus on the definition of a shallow network, we shall refer to a network with one hidden layer as shallow.

sparsity level (Polson and Rockova, 2018). Recently Bartlett et al. (2017) prove upper and lower bounds on the expressability of deep feedforward neural network classifiers with the piecewise linear activation function, such as ReLU activation functions. They show that the relationship between expressability and depth is determined by the degree of the activation function.

There is further ample theoretical evidence to suggest that shallow networks can not approximate the class of non-linear functions represented by deep ReLU networks without blow-up. Telgarsky (2016) shows that there is a ReLU network with L layers and U units such that any network approximating it with only $\mathcal{O}(L^{1/3})$ layers must have $\Omega(2^{L^{1/3}})$ units. Mhaskar et al. (2016) discuss the differences between composition versus additive models and show that it is possible to approximate higher polynomials much more efficiently with several hidden layers than a single hidden layer.

Feng et al. (2018) show that deep neural networks provide powerful expressability[2] when combined with regularization, however their use in factor modeling presents some fundamental obstacles, one of which we shall address in this paper. Neural networks have been presented to the investment management industry as 'black-boxes'. As such they are not viewed as interpretable and their internal behavior can not be reasoned on statistical grounds. We add to the literature by introducing a method of interpretability which holds for almost any feedforward network, shallow or deep. One important caveat is that we do not attempt to solve the causation problem in economic modeling.

Non-linearity While high dimensional data representation is one distinguishing aspect of machine learning over linear regression, it is not alone. Deep learning resolves predictor non-linearities and interaction effects without over-fitting through a bias-variance tradeoff. As such, it provides a highly expressive regression model for complex data which relies on compositions of simple non-linear functions rather than being additive.

On the theoretical side, we develop a method of interpretability which holds for almost any neural network, shallow or deep. On the empirical side, we study the interpretability confidence intervals tighten with increasing hidden units in a feedforward network. We also show that the interpretability method compares favorably with other classical methods on standard benchmark problems. Finally, we develop a deep fundamental factor model with 50 factors for 3290 Russell 3000 indexed stocks over an approximate ten year period and compare factor interpretability and performance with the linear fundamental factor model.

The rest of the paper is outlined as follows. Section 1.2 provides the connection with deep learning and factor models in finance. Section 2 introduces the terminology and notation for defining neural network based fundamental factor models. Section 3 introduces our neural network parameter estimation approach which, in principle, is sufficiently general for fundamental factor mod-

[2]Expressability is a measure of the generality of the class of functions representated by the network. In the context of classification, expressibility is measure by VC dimension.

els such as the BARRA model. Section 3.1 describes our method for interpreting neural network based factor models and compares the approach with other interpretability techniques. Section 5.2 demonstrates the application of our framework to neural network based factor models. Finally, Section 6 concludes with directions for future research.

1.2 Connection with Finance Factor Models

Linear cross-sectional factor models, such as (Fama and MacBeth, 1973), FF(3) and FF(5) (Fama and French, 1993, 1992, 2015) and BARRA factor models (see Rosenberg and Marathe (1976); Carvalho et al. (2012)) are appealing because of their simplicity and their economic interpretability, generating tradable portfolios. Under the assumption of homoschedasticity, factor realizations can be estimated in the BARRA model by ordinary least squares regression[3]. OLS linear regression exhibits poor expressability and relies on the Gaussian errors being orthogonal to the regressors. Generalizing to non-linearities and incorporating interaction effects is a harder task.

Asset managers seek novel predictive firm characteristics to explain anomalies which are not captured by classical capital asset pricing and factor models. Recently a number of independent empirical studies, rooted in a data science approach, have shown the importance of using a higher number of economically interpretable predictors related to firm characteristics and other common factors Moritz and Zimmermann (2016); Harvey et al. (2015); Gu et al. (2018); Feng et al. (2018). Gu et al. (2018) analyze a dataset of more than 30,000 individual stocks over a 60 year period from 1957 to 2016, and determine over 900 baseline signals. Both Moritz and Zimmermann (2016) and Gu et al. (2018) highlight the inadequacies of OLS regression in variable selection over high dimensional datasets.

Our works build on recent literature that finds evidence of superior performance of non-linear regression techniques such as regression trees and neural networks (Gu et al., 2018). Feng et al. (2018) demonstrate the ability of a three-layer deep neural network to effectively predict asset returns from fundamental factors.

2 Deep Fundamental Factor Models

Rosenberg and Marathe (1976) introduced a cross-sectional fundamental factor model to capture the effects of macroeconomic events on individual securities. The choice of factors are microeconomic characteristics – essentially common factors, such as industry membership, financial structure, or growth orientation Nielsen and Bender (2010).

[3]The BARRA factor model is often presented in the more general form with heteroschedastic error but is not considered here.

The BARRA fundamental factor model expresses the linear relationship between K fundamental factors and N asset returns:

$$\mathbf{r_t} = B_t \mathbf{f}_t + \epsilon_t,\ t = 1, \ldots, T, \tag{1}$$

where $B_t = [\mathbf{1} \mid \boldsymbol{\beta}_1(t) \mid \cdots \mid \boldsymbol{\beta}_K(t)]$ is the $N \times K + 1$ matrix of known factor loadings (betas): $\beta_{i,k}(t) := (\boldsymbol{\beta}_k)_i(t)$ is the exposure of asset i to factor k at time t.

The factors are asset specific attributes such as market capitalization, industry classification, style classification. $\mathbf{f}_t = [\alpha_t, f_{1,t}, \ldots, f_{K,t}]$ is the $K + 1$ vector of unobserved factor realizations at time t, including α_t.

$\mathbf{r_t}$ is the $N-$vector of asset returns at time t. The errors are assumed independent of the factor realizations $\rho(f_{i,t}, \epsilon_{j,t}) = 0, \forall i, j, t$ with homoschedastic Gaussian error, $\mathbb{E}[\epsilon_{j,t}^2] = \sigma^2$.

2.1 Deep Factor Models

Consider a non-linear cross-sectional fundamental factor model of the form

$$\mathbf{r}_t = F_t(B_t) + \epsilon_t, \tag{2}$$

where $\mathbf{r}_t$ are asset returns, $F_t : \mathbf{R}^K \to \mathbf{R}$ is a differentiable non-linear function that maps the i^{th} row of B to the i^{th} asset return at time t. The map is assumed to incorporate a bias term so that $F_t(\mathbf{0}) = \alpha_t$. In the special case when $F_t(B_t)$ is linear, the map is $F_t(B_t) = B_t \mathbf{f}_t$.

A key feature is that we do not assume that ϵ_t is described by a parametric distribution, such as a Gaussian distribution. In our setup, the model shall just be used to predict the next period returns only and stationarity of the factor realizations is not required.

We approximate a non-linear map, $F_t(B_t)$, with a feedforward neural network cross-sectional factor model:

$$\mathbf{r}_t = F_{W_t, b_t}(B_t) + \epsilon_t, \tag{3}$$

where F_{W_t, b_t} is a deep neural network with L layers, that is, a super-position of univariate *semi-affine* functions, $\sigma^{(\ell)}_{W_t^{(\ell)}, b_t^{(\ell)}}$, to give

$$\hat{\mathbf{r}}_t := F_{W_t, b_t}(B_t) = (\sigma^{(L)}_{W_t^{(L)}, b_t^{(L)}} \circ \cdots \circ \sigma^{(1)}_{W_t^{(1)}, b_t^{(1)}})(B_t), \tag{4}$$

and the unknown model parameters are a set of L weight matrices $W_t = (W_t^{(1)}, \ldots, W_t^{(L)})$ and a set of bias vectors $b_t = (b_t^{(1)}, \ldots, b_t^{(L)})$. Any weight matrix $W^{(\ell)} \in \mathbf{R}^{m \times n}$, can be expressed as n column m-vectors $W^{(\ell)} = [\mathbf{w}_{,1}^{(\ell)}, \ldots, \mathbf{w}_{,n}^{(\ell)}]$. We denote each weight as $w_{ij}^{(\ell)} := \left[W^{(\ell)}\right]_{ij}$.

The ℓ^{th} semi-affine function is itself defined as the composition of the activation function, $\sigma^{(\ell)}(\cdot)$, and an affine map:

$$\sigma^{(\ell)}_{W_t^{(\ell)}, b_t^{(\ell)}}(Z^{(\ell-1)}) := \sigma^{(\ell)}\left(W_t^{(\ell)} Z^{(\ell-1)} + b_t^{(\ell)}\right), \tag{5}$$

where $Z^{(\ell-1)}$ is the output from the previous layer, $\ell - 1$.

The activation functions, $\sigma^{(\ell)}(\cdot)$, e.g. $\sigma^{(\ell)}(\cdot) = \max(\cdot, 0)$, are critical to non-linear behavior of the model. Without them, F_{W_t,b_t} would be a linear map and, as such, would be incapable of capturing interaction effects between the inputs. This is true even if the network has many layers.

2.2 Architecture Design

We begin by considering a simple feedforward binary classifier with only two continuous features, as illustrated in Figure 1. The simplest configuration we shall consider has just two inputs and one output — this network is functionally equivalent to a multivariate regression model.

The next configuration we shall consider has one hidden layer — the number of hidden units shall be equal to the number of input neurons. This choice serves as a useful reference point as many hidden units are often needed for sufficient expressability. The final configuration has substantially more hidden units. Note that the second layer has been introduced purely to visualize the output from the hidden layer. These set of simple configurations (a.k.a. architectures) is ample to illustrate how a feedforward neural network method works.

There are many types of architectures for statistical and econometric modeling. Recurrent neural networks, Gated Recurrent Units and LSTMs can be used for dynamic factor modeling (Gu et al., 2018) and modeling of the limit order book (Dixon, 2018b,a). Dixon et al. (2018) introduce architectures for spatio-temporal modeling. Chen et al. (2019) use a combination of LSTMs and feedforward architectures in a Generative Adversarial Network (GAN) to enforce a no-arbitrage constraint.

Determining the weight and bias matrices, together with how many hidden units are needed for generalizable performance is the goal of training. However, we emphasize that some conceptual understanding of neural networks is needed to derive interpretability results and the utility of our framework rests on serving as either a linear factor model or a non-linear factor model, depending on whether the activation functions are disabled or enabled respectively.

3 Training

OLS estimators are consistent and asymptotically normal under heteroskedasticity and autocorrelation of the errors as long as the regressors are orthogonal to the errors. However, under heteroskedasticity and autocorrelation of the error, Generalized Least Squares (GLS) estimation provides a more efficient estimator. Under GLS estimation, we modify our loss function to the squared Mahalanobis length of the residual vector.

To construct and evaluate a deep learner for fundamental factor modeling, we start with training data $\mathcal{D} := \{\mathcal{D}_t\}_{t=1}^T$, of input-output pairs over all times $t = 1, \ldots, T$ where at each time $\mathcal{D}_t = \{y_t^{(i)}, \mathbf{x}_t^{(i)}\}_{i=1}^N$. $y_t^{(i)}$ denotes the i^{th} asset return and the $\mathbf{x}_t^{(i)}$ are the factor loadings of the i^{th} asset at time t. The goal is

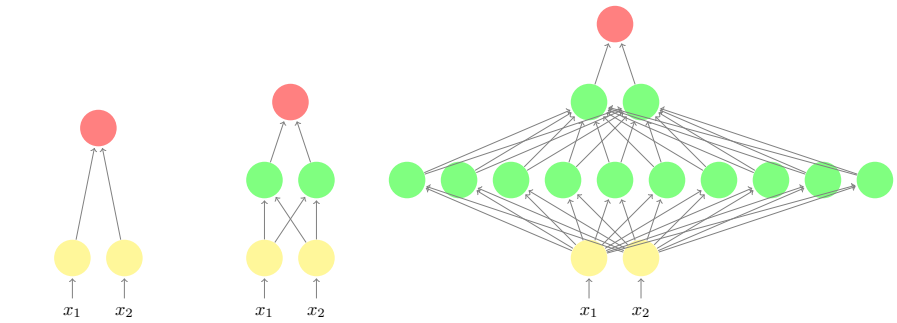


Figure 1: *Simple two variable feedforward networks with and without hidden layers. The yellow nodes denote input variables, the green nodes denote hidden units and the red nodes are outputs. A feedforward network without hidden layers is a linear regressor. A feedforward network with one hidden layer is a* shallow learner *and a feedforward network with two or more hidden layers is* deep learner.

to approximate the map $Y_t = F_t(X_t)$, with the deep learning model $F_{W_t,b_t}(X_t)$. Where there is no ambiguity, we drop t subscripts for ease of notation.

For each time period, we then solve a least squares minimization problem using data, $\mathcal{D}_t$:

$$\underset{W,b}{\text{minimize}} \quad f(W,b) + \lambda\phi(W,b), \tag{6}$$

$$\mathcal{L}(Y,\hat{Y}) := (Y-\hat{Y})^T(Y-\hat{Y}), \tag{7}$$

$$f(W,b) = \frac{1}{N}\sum_{i=1}^{N}\mathcal{L}(Y^{(i)},\hat{Y}(X^{(i)})), \tag{8}$$

where $\phi(W,b)$ is a regularization penalty term. The loss function is non-convex, possessing many local minima and is generally difficult to find a global minimum. The out-of-sample performance of the model is assessed over the next monthly period $\mathcal{D}_{t+1}$ and process is repeated over all times period as illustrated in Figure 2.

Regularization λ is a global regularization parameter which we tune using the out-of-sample predictive weighted mean-squared error (MSE) of the model. The regularization penalty, $\phi(W,b)$, introduces a bias-variance tradeoff. $\nabla\mathcal{L}$ is given in closed form by a chain rule and, through back-propagation, each layer's weights $\hat{W}^{(\ell)}$ and biases $b^{(\ell)}$ are fitted with stochastic gradient descent.

3.1 Factor Interpretability

Once the neural network has been trained, a number of important issues surface around how to interpret the model parameters. This aspect is by far the

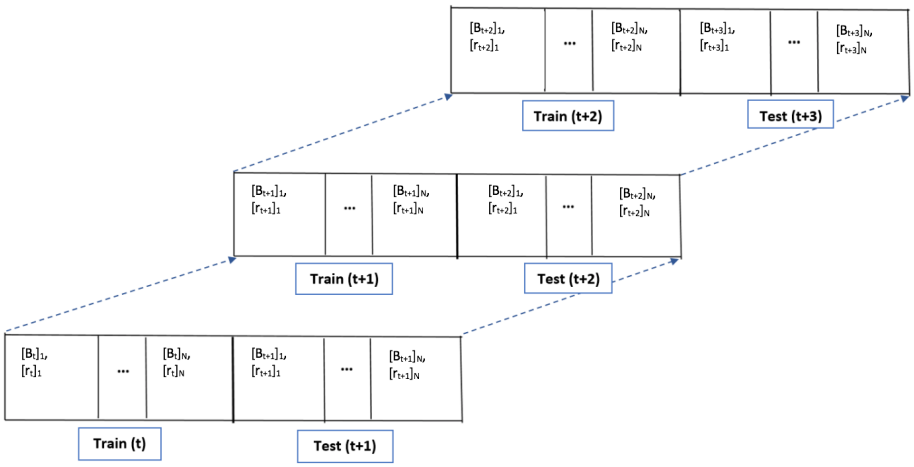


Figure 2: *The experiment is designed so that the deep network is trained over period t and then tested over period $t+1$. Each labeled training set is an factor loading matrix B_t and return vector r_t over N assets. Note that the assets are not fixed over the periods as the composition of the Russell 3000 index changes over time.*

most prominent issue in deciding whether to use neural networks in favor of other machine learning and statistical methods for estimating factor realizations, sometimes even if the latter's predictive accuracy is inferior.

In this section, we shall introduce and analyse a method for interpreting multi-layer perceptrons which imposes minimal restrictions on the neural network design.

There are numerous techniques for interpreting machine learning methods which treat the model as a black-box. The most common approach are perturbation, gradient-based, attribution methods. These include methods such as "occlusion" (Ancona et al., 2017), Partial Dependence Plots (PDPs) (Greenwell et al., 2018) and "saliency maps".

Occlusion removes, masks or alters the inputs to assess the sensitivity of the model output. PDPs which show the dependence between the target response and a set of 'target' features, marginalizing over the values of all other feature. Saliency maps constructs attributions by taking the absolute value of the partial derivative of the target output with respect to the inputs. PDPs, in particular do not scale well with high dimensional data and both methods can not easily resolve interaction effects. Saliency maps can not capture directional relationships between inputs and outputs.

Back-propagation approaches constitute another class of attribution methods and include techniques such as "Integrated Gradients" and closely related techniques such as "DeepLIFT" (Ancona et al., 2017). These techniques use back-propagation to evaluate the gradient and then multiply the partial gradi-

ent of the model by the input variable. As with perturbation methods, they compute the sample mean of the sensitivity and discard the distribution.

Various tests have been developed to identify the statistical significance of inputs to network outputs (Lee et al., 1993; Mohammadi, 2018). In particular Mohammadi (2018) introduces a formal test of the significance of neural network inputs. It is limited, however, to a linear relationship between the output of a neural network when all of the input variables are fixed at their mean values.

Other approaches also exist in the literature. Garson (1991) partitions hidden-output connection weights into components associated with each input neuron using absolute values of connection weights. Olden and Jackson (2002) determines the relative importance, R_{ij}, of the i^{th} output to the j^{th} predictor variable of the model as a function of the weights, according to a simple linear expression.

To summarize, there are four shortcomings of these approaches for financial, regression based, modeling which we address in this paper:

1. Techniques which scale the partial derivative by the input variable are not consistent with how OLS provides interpretability.

2. Interaction effects are important in factor modeling, yet are not addressed by the literature.

3. In taking the sample mean of the partial gradients, the literature on gradient-based interpretability of deep learning does not consider the distribution of the sensitivities; and

4. These studies do not offer insight into the effect of network architecture on the sharpness of the sensitivity.

To illustrate Item (1) above, consider the following data generation process $Y = X_1 + 2X_2 + \epsilon, X_1, X_2 \sim \mathcal{N}(0,1)$ and $\epsilon \sim \mathcal{N}(0, \sigma_n^2), \sigma_n << 1$. The fitted coefficients should be $\beta_1 = 1, \beta_2 = 2$ and hence provide a ranking of importance of variables. One important caveat is that we assume that the variables have been standardized. The sensitivities of the fitted network are consistent with the fitted coefficients when the data generation process is linear.

Of course the gradient attribution method generalizes to non-linear DGPs. However, we emphasize that it is through empirically establishing the consistency of the sensitivities with OLS coefficients that the practitioner can gain comfort with using a ANN in place of OLS regression. We will show that even when the network is non-linear, we may still establish consistent sensitivities with OLS on linear DGPs.

The empirical distribution provides information about the uncertainty of the sensitivity estimate and in particular, confidence intervals. This information should be factored into the assessment of the relative feature importance. For example, in the following Figure, the distributions of the sensitivities with respect to two input variables, J_1 and J_2, are shown. On average, J_2 is larger than J_1, however, J_1 is more concentrated than J_2 and consequently the relative importance is more difficult to assess.

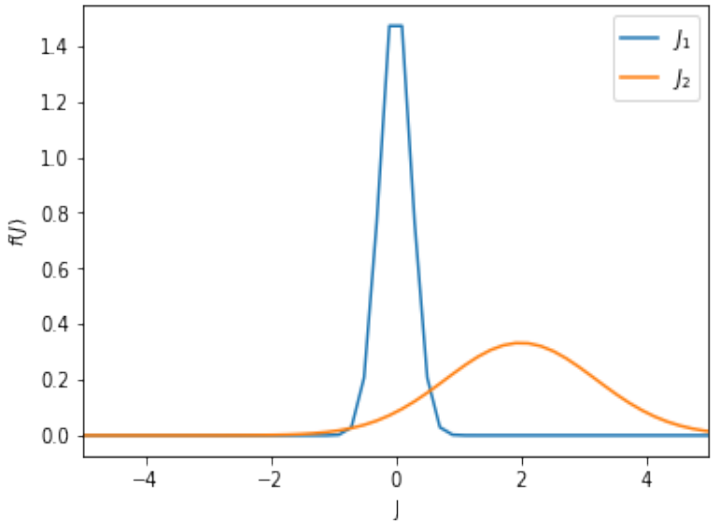


Figure 3: *The distributions of the sensitivities with respect to two input variables, J_1 and J_2, are shown. On average, J_2 is larger than J_1, however, J_1 is more concentrated than J_2 and consequently the relative importance is more difficult to assess.*

We use the variance of the sensitivity distribution to guide network architecture design. In particular, we derive bounds on the variance of the Jacobian which are independent of the distribution of the inputs. In certain limiting cases, we can show how these bounds depend on the number of hidden units in a layer and hence study the effect of the number of parameters on the sharpness of the Jacobian.

We seek to understand the limitations on the choice of activation functions and understand the effect of increasing layers and numbers of neurons on probabilistic interpretability. For example, under standard Gaussian i.i.d. data, how robust are the model's estimate of the importance of each input variable with variable number of neurons?

3.2 Sensitivity Approach

We turn to a 'white-box' technique for determining the importance of the input variables. This approach generalizes Dimopoulos et al. (1995) to a deep neural network with interaction terms. Moreover, the method is directly consistent with how coefficients are interpreted in linear regression - they are *model* sensitivities. Model sensitivities are the change of the fitted model output w.r.t. input.

As a control, we shall use this property to empirically evaluate how reliably neural networks, even deep networks, learn data from a linear model.

Such an approach is appealing to practitioners who evaluate the comparative performance of linear regression with neural networks and need the assurance that a neural network model is at least able to reproduce and match the coefficients on a linear dataset.

We also offset the common misconception that the activation functions must be deactivated for a neural network model to produce a linear output. Under linear data, any non-linear statistical model should be able to reproduce a statistical linear model under some choice of parameter values. Irrespective of whether data is linear or non-linear in practice - the best control experiment for comparing a neural network estimator with an OLS estimator is to simulate data under a linear regression model. In this scenario, the correct model coefficients are known and the error in the coefficient estimator can be studied.

To evaluate fitted model sensitivities analytically, we require that the function $\hat{Y} = F(X)$ is continuous and differentiable everywhere. Furthermore, for stability of the interpretation, we shall require that $F(x)$ is a Lipschitz continuous[4]. That is, there is a positive real constant K s.t. $\forall x_1, x_2 \in \mathbb{R}^p$, $|F(x_1) - F(x_2)| \leq K|x_1 - x_2|$. Such a constraint is necessary for the first derivative to be bounded and hence amenable to the derivatives, w.r.t. to the inputs, providing interpretability.

Fortunately, provided that the weights and biases are finite, each semi-affine function is Lipschitz continuous everywhere. For example, the function $\tanh(x)$ is continuously differentiable with derivative, $1 - \tanh^2(x)$, is globally bounded. With finite weights, the composition of $\tanh(x)$ with an affine function is also Lipschitz. Clearly $\text{ReLU}(x) := \max(\cdot, 0)$ is not continuously differentiable and one can not use the approach described here. Note that for the following examples, we are indifferent to the choice of homoscedastic or heteroscedastic error, since the model sensitivities are independent of the error.

In a linear regression model

$$\hat{Y} = F_\beta(X) := \beta_0 + \beta_1 X_1 + \cdots + \beta_K X_K, \tag{9}$$

the model sensitivities are

$$\partial_{X_i} \hat{Y} = \beta_i. \tag{10}$$

In a FFWD neural network, we can use the chain rule to obtain the model sensitivities

$$\partial_{X_i} \hat{Y} = \partial_{X_i} F_{W,b}(X) = \partial_{X_i} \sigma^{(L)}_{W^{(L)},b^{(L)}} \circ \cdots \circ \sigma^{(1)}_{W^{(1)},b^{(1)}}(X). \tag{11}$$

For example, with one hidden layer, $\sigma(x) := \tanh(x)$ and $\sigma^{(1)}_{W^{(1)},b^{(1)}}(X) := \sigma(I^{(1)}) := \sigma(W^{(1)}X + b^{(1)})$:

$$\partial_{X_j} \hat{Y} = \sum_i \mathbf{w}^{(2)}_{,i}(1 - \sigma^2(I^{(1)}_i))w^{(1)}_{ij} \quad \text{where } \partial_x \sigma(x) = (1 - \sigma^2(x)). \tag{12}$$

[4]If Lipschitz continuity is not imposed, then a small change in one of the input values could result in an undesirable large variation in the derivative

In matrix form, with general σ, the Jacobian [5] of σ w.r.t X is $J = D(I^{(1)})W^{(1)}$ of σ,

$$\partial_X \hat{Y} = W^{(2)} J(I^{(1)}) = W^{(2)} D(J^{(1)}) W^{(1)}, \tag{13}$$

where $D_{ii}(I) = \sigma'(I_i),\ D_{ij} = 0,\ i \neq j$ is a diagonal matrix. Bounds on the sensitivities are given by the product of the weight matrices

$$\min(W^{(2)}W^{(1)}, 0) \leq \partial_X \hat{Y} \leq \max(W^{(2)}W^{(1)}, 0). \tag{14}$$

Multiple hidden layers The model sensitivities can be readily generalized to an L layer deep network by evaluating the Jacobian matrix:

$$\partial_X \hat{Y} = W^{(L)} J(I^{(L-1)}) = W^{(L)} D(I^{(L-1)}) W^{(L-1)} \dots D(I^{(1)}) W^{(1)}. \tag{15}$$

Example: Step test To illustrate our interpretability approach, we shall consider a simple example. The model is trained to the following data generation process where the coefficients of the features are stepped and the error, here, is i.i.d. uniform:

$$\hat{Y} = \sum_{i=1}^{10} i X_i,\ X_i \sim \mathcal{U}(0, 1). \tag{16}$$

Figure 4 shows the ranked importance of the input variables in a neural network with one hidden layer. Our interpretability method is compared with well known black-box interpretability methods such as Garson's algorithm (Garson, 1991) and Olden's algorithm (Olden and Jackson, 2002). Our approach is the only technique to interpret the fitted neural network which is consistent with how a linear regression model would interpret the input variables.

3.3 Interaction Effects

The previous example is too simplistic to illustrate another important property of our interpretability method, namely the ability to capture pairwise interaction terms. The pairwise interaction effects are readily available by evaluating the elements of the Hessian matrix. For example, with one hidden layer, the Hessian takes the form:

$$\partial^2_{X_i X_j} \hat{Y} = W^{(2)} diag(W_i^{(1)}) D'(I^{(1)}) W_j^{(1)}, \tag{17}$$

where it is assumed that the activation function is at least twice differentiable everywhere, e.g. $tanh(x)$.

More generally, for a L layer network, we define the $(i, j)^{(th)}$ element of the Hessian as

$$\partial^2_{X_i X_j} \hat{Y} = \sum_{\ell=1}^{L-1} H_{i,j,\ell}, \tag{18}$$

[5] When σ is an identity function, the Jacobian $J(I^{(1)}) = W^{(1)}$.

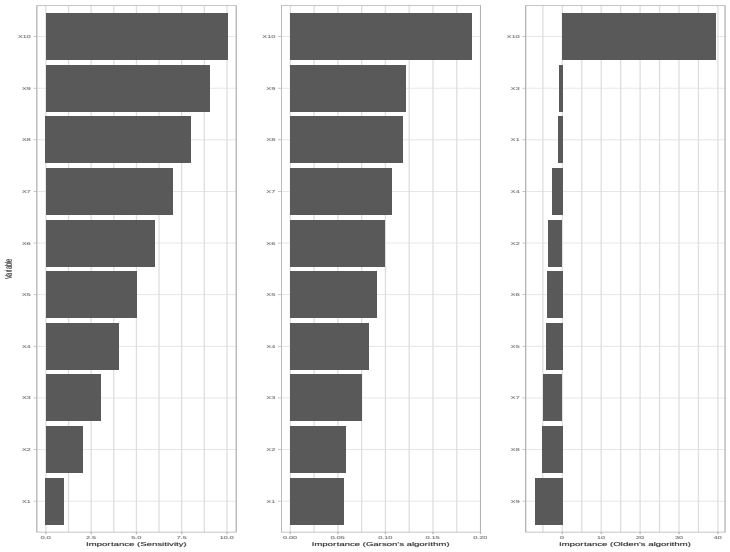


Figure 4: *Step test: This figure shows the ranked importance of the input variables in a neural network with one hidden layer. (left) Our sensitivity based approach for input interpretability. (Center) Garson's algorithm and (Right) Olden's algorithm. Our approach is the only technique to interpret the fitted neural network which is consistent with how a linear regression model would interpret the input variables.*

where the components of the sum are given by

$$H_{i,j,\ell} := W^{(L)} D^{(L-1)} W^{(L-1)} \dots \partial_{X_j} D^{(\ell)} W^{(\ell)} \dots w_i^{(1)}, \tag{19}$$

where the derivative is taken over $D^{(\ell)}$ only and is given by the chain rule.

For example, with two hidden layers, each with n activated hidden units:

$$\frac{\partial \hat{Y}}{\partial X_i X_j} = W^{(3)} W^{(2)} \left(\frac{\partial D^{(2)}}{\partial X_j} D^{(1)} + \frac{\partial D^{(1)}}{\partial X_j} D^{(2)} \right) w_i^{(1)}$$

where applying the chain rule we have

$$\begin{aligned} \frac{\partial D^{(2)}}{\partial X_j} &= D'(I^{(2)}) W^{(2)} D^{(1)} diag(w_j^{(1)}), \\ \frac{\partial D^{(1)}}{\partial X_j} &= D'(I^{(1)}) diag(w_j^{(1)}). \end{aligned}$$

Under a smooth approximation to ReLU

$$\sigma(x;t) := \frac{1}{t} ln(1 + e^{tx}),$$

which is ReLU in the asymptotic limit of large t

$$\lim_{t\to\infty} \sigma(x,t) = \max(x,0) \tag{20}$$

with first derivative $\partial_x\sigma(x;t) = \frac{1}{1+e^{-tx}}$ and second derivative $\partial_x^2\sigma(x;t) = te^{-tx}\left(\partial_x\sigma(x;t)\right)^2$ gives

$$D_{i,i}(I_i^{(\ell)}) = \frac{1}{1+e^{-tI_i^{(\ell)}}}, \tag{21}$$

and

$$D_{i,i}^{'}(I_i^{(\ell)}) = te^{-tI_i^{(\ell)}}\left(\partial_x\sigma(I_i^{(\ell)};t)\right)^2, D_{i,j\neq i}^{'}(I_i^{(2)}) = 0. \tag{22}$$

Example: Friedman data To illustrate our input variable and interaction effect ranking approach, we will use one of the classical benchmark regression problems described in (Friedman, 1991) and (Breiman, 1996). The input space consists of ten i.i.d. uniform $\mathcal{U}(0,1)$ random variables; however, only five out of these ten actually appear in the true model. The response is related to the inputs according to the formula

$$Y = 10\sin(\pi x_1 x_2) + 20\left(x_3 - 0.5\right)^2 + 10x_4 + 5x_5 + \epsilon,$$

using white noise error, $\epsilon \sim \mathcal{N}\left(0,\sigma^2\right)$. We fit a NN with one hidden layer containing eight units and a weight decay of 0.01 (these parameters were chosen using 5-fold cross-validation) to 500 observations simulated from the above model with $\sigma = 1$. The cross-validated R^2 value was 0.94.

Figures 5 and 12 respectively compare the ranked model sensitivities and ranked interaction terms of the fitted neural network with Garson's and Olden's algorithm.

4 Bounds on the Variance of the Jacobian

Consider a *ReLU* activated single layer feed-forward network. In matrix form, with $h(x) = \max(x,0)$, the Jacobian, J, can be written as a linear combination of Heaviside functions:

$$J := J(X) = \partial_X\hat{Y}(X) = W^{(2)}J(Z^{(1)}) = W^{(2)}H(W^{(1)}X + b^{(1)})W^{(1)}, \tag{23}$$

where $H_{ii}(Z) = H(Z_i^{(1)}) = \mathbb{1}_{Z_i^{(1)}>0}$, $H_{ij} = 0,\ j \geq i$.

We recall a well known result, namely the set of n hyperplanes defines a hyperplane arrangement. Such an arrangement of $n \geq p$ hyperplanes in $\mathbb{R}^p$ has $n_c \leq \sum_{j=0}^{p}\binom{n}{j}$ convex regions over a bounded domain (Montúfar et al., 2014).

For example in a two dimensional input space, three neurons with ReLU activation functions will divide the space into $n_c \leq \sum_{j=0}^{2}\binom{3}{j} = 7$ regions, as shown in Figure 7.

We label each of the convex regions as $\{A_1,\ldots,A_{n_c}\}$. We can now state the following bound on the Jacobian of the network.

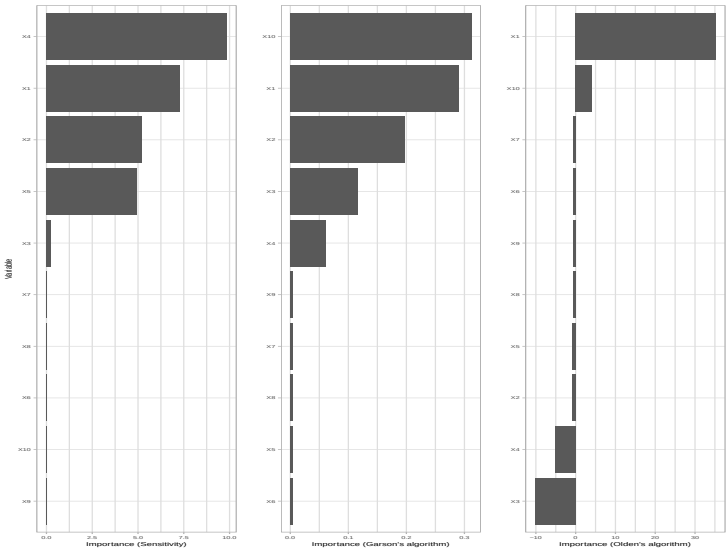


Figure 5: *Friedman test: Ranked model sensitivities of the fitted neural network to the input. (left) Our sensitivity based approach for input interpretability. (Center) Garson's algorithm and (Right) Olden's algorithm.*

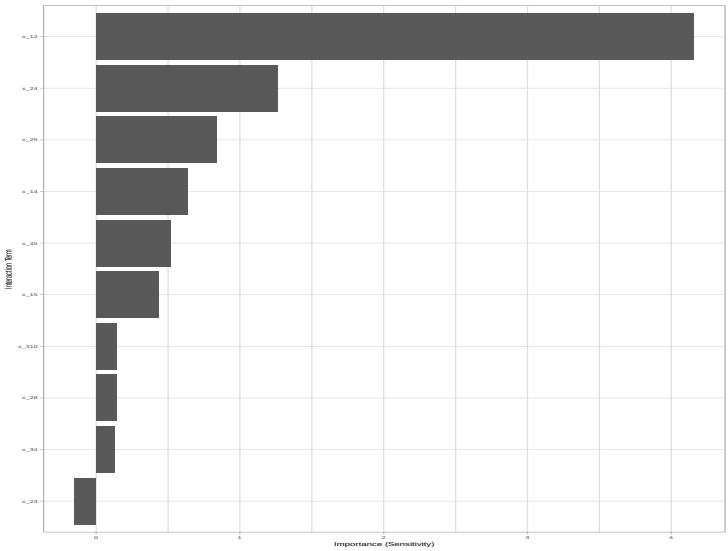


Figure 6: *Friedman test: Ranked pairwise interaction terms in the fitted neural network to the input. (left) Our sensitivity based approach for ranking interaction terms. (Center) Garson's algorithm and (Right) Olden's algorithm.*

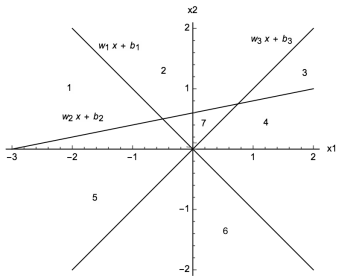


Figure 7: *Hyperplanes defined by three neurons in the hidden layer, each with ReLU activation functions, form a hyperplane arrangement. An arrangement of 3 hyperplanes in* $\mathbb{R}^2$ *has* $n_c \leq \sum_{j=0}^{2} \binom{3}{j} = 7$ *convex regions.*

Lemma (Dixon & Polson, 2019)**.** *If* $X \in \mathbb{R}^p$ *is i.i.d. and there are* n *hidden units, with at most* n_c *convex regions in the bounded hyperplane arrangement, then the variance of the Jacobian of a feedforward network with a single ReLU activated hidden layer has the following upper bound:*

$$\mathbb{V}[J_{ij}] \leq \frac{1}{2}||\mathbf{a}||_2^2. \tag{24}$$

where $\mathbf{a} := (a_1, \ldots, a_{n_c})$ *is a vector whose* m^{th} *component is the weighted sum of* n *Heaviside functions over convex region* A_m *and* $a_m \leq (\mathbf{w}_{i,}^{(2)})^T \mathbf{w}_{,j}^{(1)}$, $\forall m \in \{1, \ldots, n_c\}$.

Proof. The Jacobian can be written in matrix element form as

$$J_{ij} = [\partial_X \hat{Y}]_{ij} = \sum_{k=1}^{n} w_{ik}^{(2)} w_{kj}^{(1)} H(I_k^{(1)}) = \sum_{k=1}^{n} c_k H_k(I^{(1)}) \tag{25}$$

where $c_k := c_{ijk} := w_{ik}^{(2)} w_{kj}^{(1)}$ and $H_k(I^{(1)}) := H(I_k^{(1)})$ is the Heaviside function.

For each region, A_m, define the weighted sum of activated functions:

$$a_m := \sum_{k=1}^{n} c_k H_k(I^{(1)}(\mathbf{x})),\ \mathbf{x} \in A_m. \tag{26}$$

As a linear combination of indicator functions, we have

$$J_{ij} = \sum_{m=1}^{n_c} a_m \mathbb{1}_{A_m}. \tag{27}$$

The expectation of the Jacobian is given by

$$\mu_{ij} := \mathbb{E}[J_{ij}] = \sum_{m=1}^{n_c} a_m p_m, \tag{28}$$

where $p_m := \Pr(X \in A_m)$.

For finite weights, the expectation is bounded above by $\sum_{m=1}^{n_c} a_m$. We can write the variance of the Jacobian as:

$$\mathbb{V}[J_{ij}] = \sum_{m=1}^{n_c} a_m^2 \mathbb{V}[\mathbb{1}_{\{X \in A_m\}}] = \sum_{m=1}^{n_c} a_m^2 p_m (1 - p_m). \tag{29}$$

It follows that:

$$\begin{aligned} \mathbb{V}[J_{ij}] &= \sum_{m=1}^{n_c} a_m^2 p_m (1 - p_m) \\ &\leq \frac{1}{2} \sum_{m=1}^{n_c} a_m^2. \end{aligned}$$

□

Remark 4.1. *Note that the result holds without (i) distributional assumptions on X other than i.i.d.; and (ii) specifying the number of data points.*

,

Remark 4.2. *This result also suggests that the inputs should ideally be rescaled so that μ_{ij}, the expected value of each element of the Jacobian, is a small positive value.*

Remark 4.3. *Alternatively, we can derive probabilistic bounds on the variance of the Jacobians, for any choice of activation function, using Chernoff bounds. The bounds tend to be weak but also introduce a dependency on μ. See Section B for further details.*

We generalize this lemma to a ReLU activated neural network with L layers.

Theorem (Dixon & Polson, 2019)**.** *If $X \in \mathbb{R}^p$ is i.i.d. and there are $\{n^{(1)}, \ldots, n^{(L-1)}\}$ hidden units across $L-1$ hidden layers, with at most $\{n_c^{(1)}, \ldots, n_c^{(L-1)}\}$ convex regions in the bounded hyperplane arrangements of each layer, then the variance of the Jacobian of a feedforward network with a single ReLU activated hidden layer has the following upper bound:*

$$\mathbb{V}[J_{ij}] \leq \frac{1}{2} ||\mathbf{a}||_2^2. \tag{30}$$

Proof. Recall that a L layer network has the jacobian:

$$J_{ij} = \sum_{k^{(1)}=1,\ldots,k^{(L-1)}=1}^{n^{(1)},\ldots,n^{(L-1)}} w^{(L)}_{i,k^{(L-1)}} w^{(L-1)}_{k^{(L-1)},k^{(L-2)}} \ldots w^{(1)}_{k^{(1)},j} \prod_{\ell=1}^{L-1} D^{(\ell)}_{k^{(\ell)},k^{(\ell)}} \tag{31}$$

which we can write as

$$J_{ij} = \sum_{k^{(1)}=1,\ldots,k^{(L-1)}=1}^{n^{(1)},\ldots,n^{(L-1)}} c_{k^{(1)},\ldots,k^{(L-1)}} H^{(1),\ldots,(L-1)}_{k^{(1)},\ldots,k^{(L-1)}} \tag{32}$$

where for ease of notation $c_{k^{(1)},\ldots,k^{(L-1)}} := c_{i,j,k^{(1)},\ldots,k^{(L-1)}}$ and the product of Heaviside functions is $H^{(1),\ldots,(L-1)}_{k^{(1)},\ldots,k^{(L-1)}} := H_{k^{(L-1)}}(I^{(L-1)}) \cdot \cdots \cdot H_{k^{(1)}}(I^{(1)})$. As before we can rewrite the sum of Heavisides as the sum of indicator functions over convex regions

$$J_{ij} = \sum_{m^{(1)}=1,\ldots,m^{(L-1)}=1}^{n_c^{(1)},\ldots,n_c^{(L-1)}} a_{m^{(1)},\ldots,m^{(L-1)}} \prod_{\ell=1}^{L-1} \mathbb{1}_{Z^{(\ell-1)} \in A^{(\ell)}_{m^{(\ell)}}}, \tag{33}$$

where the weighted sum of Heavisides over the activated convex regions $\{A^{(1)}_{m^{(1)}}, \ldots, A^{(L-1)}_{m^{(L-1)}}\}$ is

$$a_{m^{(1)},\ldots,m^{(L-1)}} := \sum_{k^{(1)}=1,\ldots,k^{(L-1)}=1}^{n^{(1)},\ldots,n^{(L-1)}} c_{k^{(1)},\ldots,k^{(L-1)}} \prod_{\ell=1}^{L-1} H^{(\ell)}_{k^{(\ell)}}(Z^{(\ell-1)}), \;\; Z^{(\ell-1)} \in A^{(\ell)}_{m^{(\ell)}}. \tag{34}$$

The $n^{(\ell)}$ hyperplanes in layer ℓ partitions each bounded input space $\mathbb{R}^{n^{(\ell-1)}}$ into at most $n_c^{(\ell-1)}$ convex regions $\{A^{(\ell)}_1, \ldots, A^{(\ell)}_{n_c^{(\ell)}}\}$ where $n^{(0)} = p$ and $Z^{(0)} = X$. Under expectations we have

$$\mu_{ij} := \mathbb{E}[J_{ij}] = \sum_{m^{(1)}=1,\ldots,m^{(L-1)}=1}^{n_c^{(1)},\ldots,n_c^{(L-1)}} a_{m^{(1)},\ldots,m^{(L-1)}} p_{m^{(1)},\ldots,m^{(L-1)}}, \tag{35}$$

where the joint probabilities of the event sequence $\{Z^{(0)} \in A^{(1)}_{m^{(1)}}, \ldots, Z^{(L-2)} \in A^{(L-1)}_{m^{(L-1)}}\}$ implies that

$$p_{m^{(1)},\ldots,m^{(L-1)}} := \Pr(Z^{(0)} \in A^{(1)}_{m^{(1)}}, \ldots, Z^{(L-2)} \in A^{(L-1)}_{m^{(L-1)}}), \tag{36}$$

if sequence is assumed unique and an independent set. The variance is given by

$$\begin{aligned} \mathbb{V}[J_{ij}] &= \sum_{m^{(1)}=1,\ldots,m^{(L-1)}=1}^{n_c^{(1)},\ldots,n_c^{(L-1)}} a^2_{m^{(1)},\ldots,m^{(L-1)}} p_{m^{(1)},\ldots,m^{(L-1)}} (1 - p_{m^{(1)},\ldots,m^{(L-1)}}) \\ &\leq \frac{1}{2} \sum_{m^{(1)}=1,\ldots,m^{(L-1)}=1}^{n_c^{(1)},\ldots,n_c^{(L-1)}} a^2_{m^{(1)},\ldots,m^{(L-1)}}. \end{aligned}$$

□

In the next section, we provide empirical evidence that other activation functions, such as *tanh* do exhibit convergent Jacobian distributions as the number of units increases.

5 Applications

5.1 Simulated Example

In this section, we demonstrate the estimation properties of neural network sensitivities applied to data simulated from a linear model. We show that the sensitivities in a neural network are consistent with the linear model, even if the neural network model is non-linear. We also show that the confidence intervals, estimated by sampling, converge with increasing hidden units.

We generate 400 simulated training samples from the following linear model with i.i.d. Gaussian error:

$$Y = \beta_1 X_1 + \beta_2 X_2 + \epsilon, \quad X_1, X_2, \epsilon \sim N(0,1), \quad \beta_1 = 1, \beta_2 = 1. \tag{37}$$

Table 1 compares an OLS estimator with a zero hidden layer feedforward network (NN_0) and a one hidden layer feedforward network with 10 hidden neurons and tanh activation functions (NN_1). The functional form of the first two regression models are equivalent, although the OLS estimator has been computed using a matrix solver whereas the zero layer hidden network parameters have been fitted with stochastic gradient descent.

Model	Intercept		Sensitivity of X_1		Sensitivity of X_2	
OLS	$\hat{\beta}_0$	0.01	$\hat{\beta}_1$	1.0154	$\hat{\beta}_2$	1.018
NN_0	$\hat{b}^{(1)}$	0.02	$\hat{W}_1^{(1)}$	1.0184141	$\hat{W}_2^{(1)}$	1.02141815
NN_1	$\hat{W}^{(2)}\sigma(\hat{b}^{(1)}) + \hat{b}^{(2)}$	0.02	$\mathbb{E}[\hat{W}^{(2)}D^{(1)}(X)\hat{W}_1^{(1)}]$	1.013887	$\mathbb{E}[\hat{W}^{(2)}D^{(1)}(X)\hat{W}_2^{(1)}]$	1.02224

Table 1: *This table compares the functional form of the variable sensitivities and values with an OLS estimator. NN_0 is a zero hidden layer feedforward network and NN_1 is a one hidden layer feedforward network with 10 hidden neurons and tanh activation functions.*

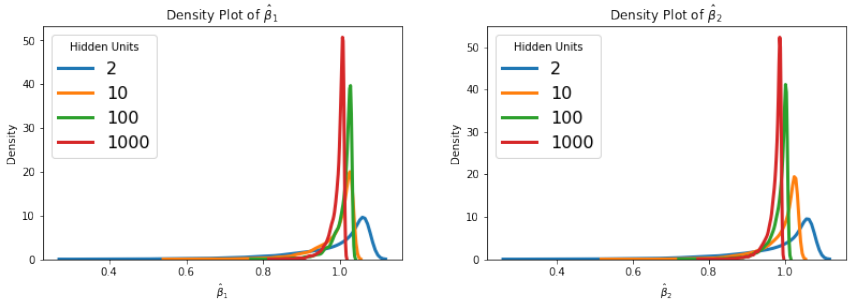
The fitted parameters values will vary slightly with each optimization as the stochastic gradient descent is randomized. However, the sensitivity terms are given in closed form and easily mapped to the linear model. In an industrial setting, such a one-to-mapping is useful for migrating to a deep factor model where, for model validation purposes, compatibility with linear models should be recovered in a limiting case. Clearly, if the data is not generated from a linear model, then the parameter values would vary across models.

Figure 8 and Tables 2 and 3 shows the empirical distribution of the fitted sensitivities using the single hidden layer model with increasing hidden units. The sharpness of the distributions is observed to converge monotonically with the number of hidden units. The confidence intervals are estimated under a non-parametric distribution.

In general, provided the weights and biases of the network are finite, the variances of the sensitivities are bounded for any input and choice of activation function.

Note that probabilistic bounds on the variance of the Jacobians can be derived for the case when the network is ReLU activated are given in Section B.

We do not recommend using ReLU activation because it does not permit identification of the interaction terms and has provably non-convergent sensitivity variances as a function of the number of hidden units (see Appendix B).



(a) density of $\hat{\beta}_1$ (b) density of $\hat{\beta}_2$

Figure 8: *This figure shows the empirical distribution of the sensitivities $\hat{\beta}_1$ and $\hat{\beta}_2$. The sharpness of the distribution is observed to converge monotonically with the number of hidden units.*

Hidden Units	Mean	Median	Std.dev	1% C.I.	99% C.I.
2	0.980875	1.0232913	0.10898393	0.58121675	1.0729908
10	0.9866159	1.0083131	0.056483902	0.76814914	1.0322522
50	0.99183553	1.0029879	0.03123002	0.8698967	1.0182846
100	1.0071343	1.0175397	0.028034585	0.89689034	1.0296803
200	1.0152218	1.0249312	0.026156902	0.9119074	1.0363332

Table 2: *This table shows the moments and and 99% confidence interval of the empirical distribution of the sensitivity $\hat{\beta}_1$. The sharpness of the distribution is observed to converge monotonically with the number of hidden units.*

Hidden Units	Mean	Median	Std.dev	1% C.I.	99% C.I.
2	0.98129386	1.0233982	0.10931312	0.5787732	1.073728
10	0.9876832	1.0091512	0.057096474	0.76264584	1.0339714
50	0.9903236	1.0020974	0.031827927	0.86471796	1.0152498
100	0.9842479	0.9946766	0.028286876	0.87199813	1.0065105
200	0.9976638	1.0074166	0.026751818	0.8920307	1.0189484

Table 3: *This table shows the moments and and 99% confidence interval of the empirical distribution of the sensitivity $\hat{\beta}_2$. The sharpness of the distribution is observed to converge monotonically with the number of hidden units.*

5.2 Russell 3000 Factor Modeling

This section presents the application of our framework to the fitting of a fundamental factor model. The BARRA factor model includes many more explanatory variables than used in our experiments below, but the purpose, here, is to illustrate the application of our framework to a larger dataset.

For completeness, we provide evidence that our deep neural network factor model generates positive and higher information ratios than OLS when used to sort portfolios from a larger universe, using 49 factors (see Table 4 for a description of the factors). The dataset is not provided due to data licensing restrictions.

We define the universe as 3290 stocks from the Russell 3000 index. Factors are given by Bloomberg and reported monthly over the period from November 2008 to November 2018. Five fold cross-validation over $\{50, 100, 200\}$ hidden units per layer, $\lambda_1 \in \{0, 10^{-3}, 10^{-2}, 10^{-1}\}$ and $\{1, 2, 3\}$ hidden layers is performed. We find that the optimal architecture has two hidden layers, 50 units per layer and no L_1 regularization. Tanh activation functions are used to provide differentiability.

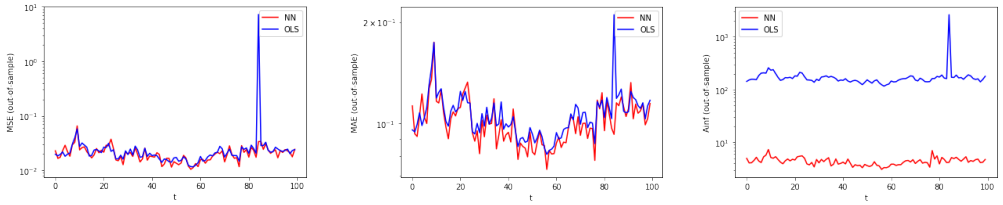
We use `Tensorflow` Abadi et al. (2016) to implement the deep feed-forward network and develop a custom implementation for factor sensitivites. The OLS model is implemented using Python `StatsModels` module.

Figure 10 compares the out-of-sample performance of neural networks and OLS regression under various norms. We observe the ability of the neural network to capture outliers, with the L_∞ norm of the error in the NN being an order of magnitude smaller than in the OLS model.

Figure 9 compares the information ratios of a portfolio selection strategy which selects the n stocks with the highest predicted monthly returns. The information ratios are evaluated for various size portfolios, using the Russell 3000 index as the benchmark. Also shown, for control, are randomly selected portfolios, without the use of a predictive signal. We observe that the information ratio of the portfolio returns, using the deep learning model, is approximately 2x larger than the OLS model. We also observe that the baseline random portfolio has an information ratio which is negligible.

Figure 11 ranks the factors which are most likely to fall under the top 15 most important factors. Some probabilities are obtained by repeatedly ranking the factors by their sensitivities over each time period and counting the number of occurrences of a factor in the top fifteen factors. Note that the choice of 15 is arbitrary and chosen for illustrative purposes. The factor codes (see Table 4) are shown on the y axis and the probabilities are shown on the x-axis.

For the DNN, EBIDTA to EV (5), Forecasted Earning/Price (6) and Earnings Volatility to Total Assets (12) are the three most likely factors. For the OLS model, Earnings/Price (3), Forecasted Earning/Price (6) and EBIDTA to EV (5) are the top three most likely factors. Hence both models agree on many of the factor sensitivities.



(a) L_2 error (b) L_1 error (c) L_∞ error

Figure 9: *The out-of-sample error, under various norms, is compared between OLS and a two-hidden layer deep network applied to a universe of 3290 stocks from the Russell 3000 index over the period from November 2008 to November 2018.*

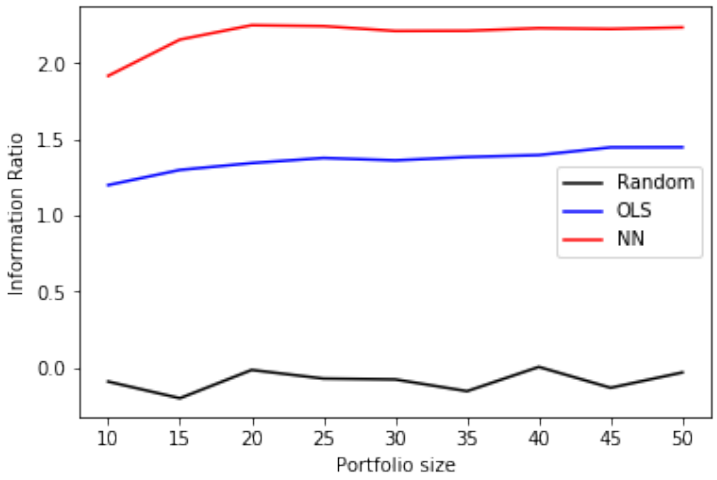
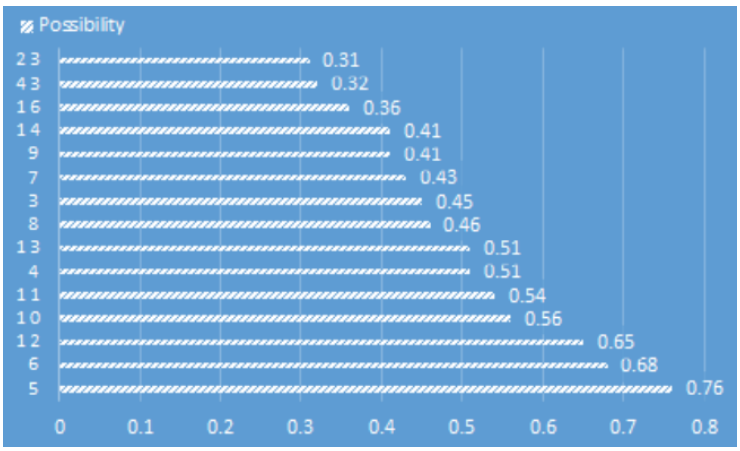
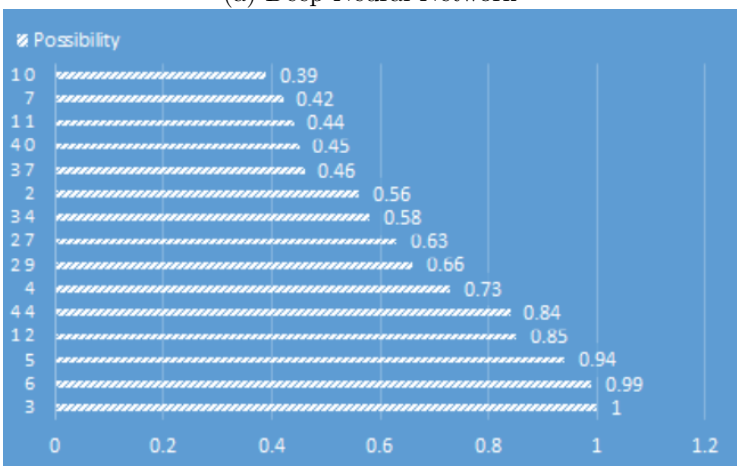


Figure 10: *The information ratios of a portfolio selection strategy which selects the n stocks from the universe with the highest predicted monthly returns. The information ratios are evaluated for various size portfolios. The information ratios are based on out-of-sample predicted asset returns using OLS regression, neural networks and randomized selection with no predictive model.*



(a) Deep Neural Network



(b) OLS

Figure 11: *Both the OLS and DNN are fitted to cross-sectional data for every time period. The overall probability that the following factors are in the top fifteen, as ranked by sensitivities, is compared between the DNN (top) and OLS (bottom). For the DNN, EBIDTA to EV (5), Forecasted Earning/Price (6) and Earnings Volatility to Total Assets (12) are the three most likely factors. For the OLS model, Earnings/Price (3), Forecasted Earning/Price (6) and EBIDTA to EV (5) are the top three most likely factors. Hence both models agree on many of the factor importances. See Table 4 for a short description of the fundamental factors.*

5.3 Interaction terms

To assess the importance of the interaction terms and compare the effects between the DNN and the linear model, we replace the OLS model with a 2nd order polynomial LASSO model. This LASSO model captures quadratic terms in the factors and pairwise interactions. The amount of regularization needed in the LASSO model is found by cross-validation using the `statsmodels`Python package.

Figure 12 compares the distribution of factor model sensitivities over the entire ten year period using the (top) neural network and (bottom) LASSO model. The sensitivities are sorted in descending order from left to right by their absolute median values and the top 20 are shown in the figure. Terms with XX_YY denote pairwise interaction effects between factors XX and factors YY.

We note that interaction terms feature prominently in the LASSO model but not in the DNN. Trading Activity (TrA), Log Market Cap (MC) and Log (Total Assets) (TA) are observed to be the most important factors which are common between the models. We also observe considerably more variability in the sensitivity of the DNN model to the most sensitive factors than in the LASSO model which may explain why the DNN is able to capture outliers more effectively than the LASSO model (see Figure 13). For example, while the interquartile range is moderate, we note that the DNN has the capacity to capture outlier sensitivities which are ignored in the LASSO model.

Figure 14 compares the medians of factor model sensitivities and interaction terms over the entire ten year period using the deep neural network and the LASSO model. The sensitivities are sorted in descending order from left to right by their absolute median values and only the top 20 terms are shown. See Table 4 for a brief description of the fundamental factors. We note that interaction terms feature prominently in the LASSO model but not in the DNN. Also, the sensitivities to Trading Activity (TrA) and Log Market Cap (MC) are both positive and Log Total Assets (TA) is negative in LASSO and the DNN.

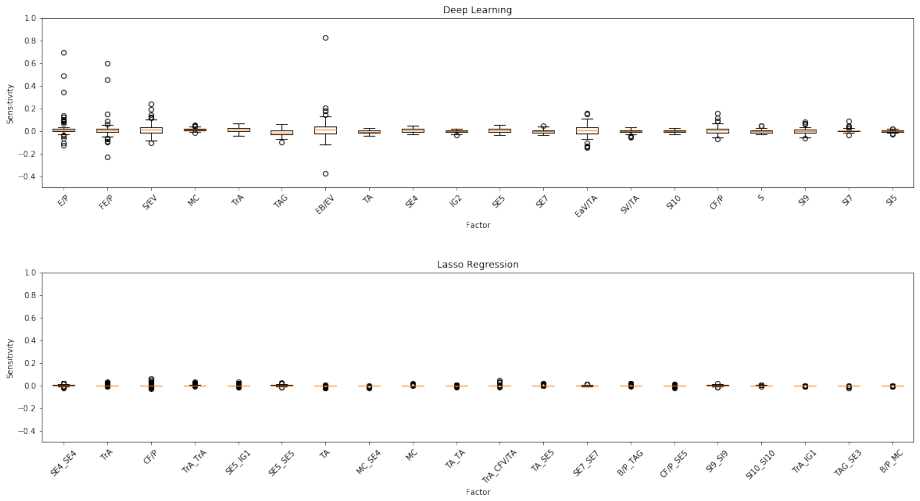


Figure 12: *The distribution of factor model sensitivities and interaction terms over the entire ten year period using the deep neural network applied to the Russell 3000 asset factor loadings (top). The sensitivities are sorted in descending order from left to right by their absolute median values and the top 20 terms have been shown. The same sensitivities using LASSO regression are shown (bottom). See Table 4 for a brief description of the fundamental factors. Terms with XX_YY denote pairwise interaction effects between factors XX and factors YY. We note that interaction terms feature prominently in the LASSO model but not in the DNN. Trading Activity (TrA), Log Market Cap (MC) and Log (Total Assets) (TA) are observed to be the most important factors which are common between the models.*

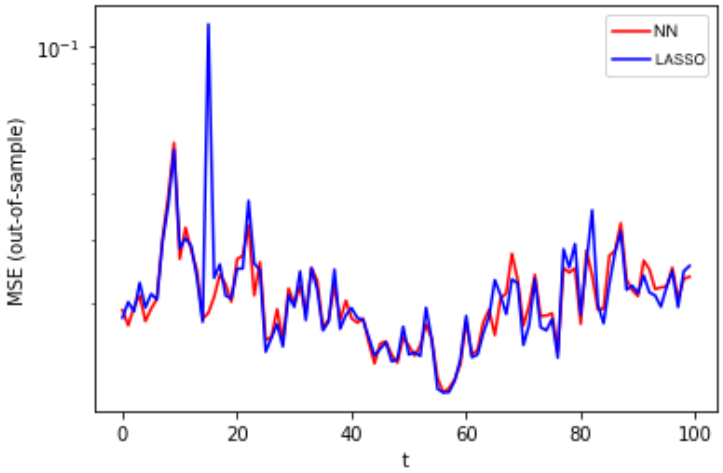


Figure 13: *The out-of-sample MSE for the DNN and the LASSO model.*

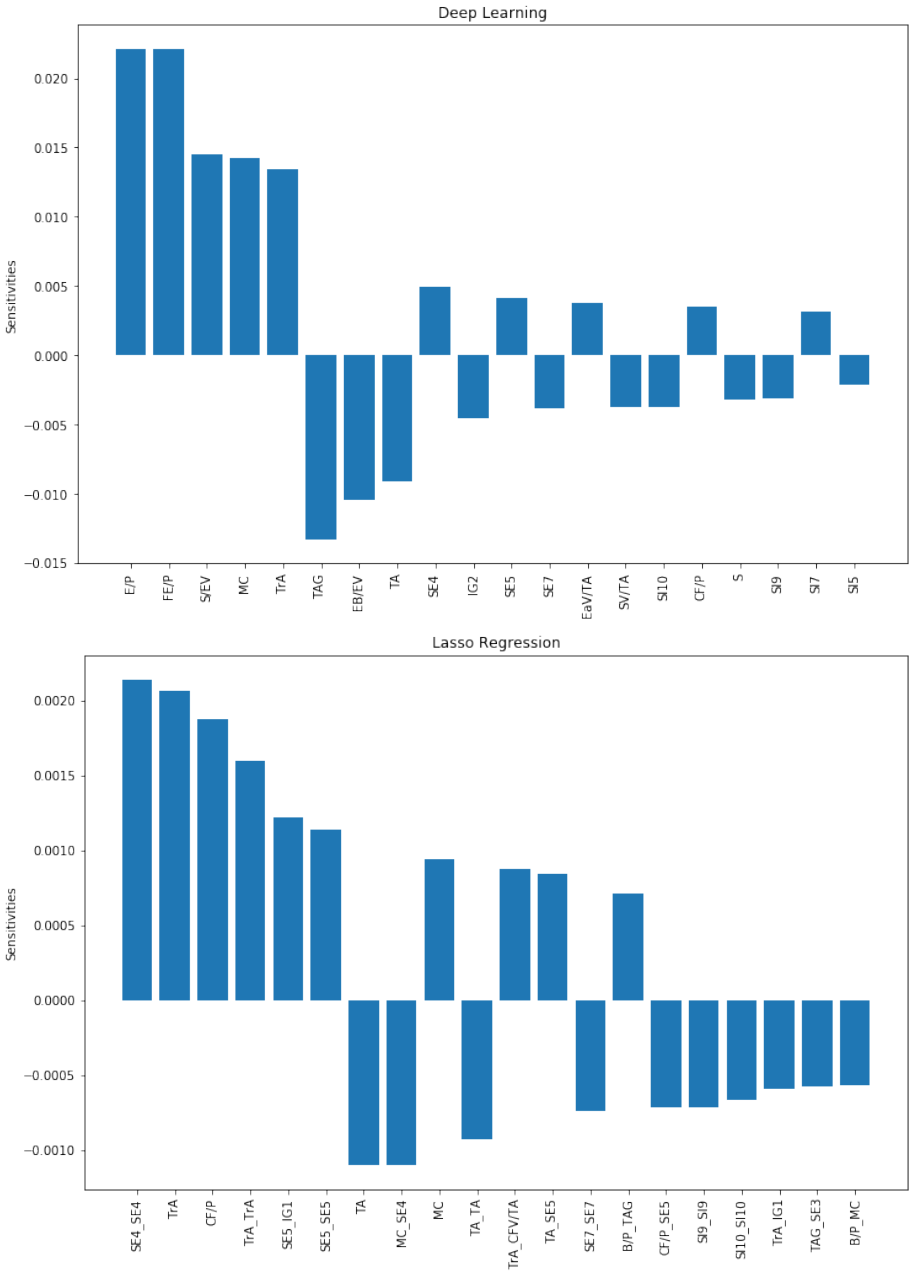


Figure 14: *The medians of factor model sensitivities and interaction terms over the entire ten year period using the deep neural network applied to the Russell 3000 asset factor loadings (top). The sensitivities are sorted in descending order from left to right by their absolute median values and the top 20 terms have been shown. The same sensitivities using LASSO regression are shown (bottom). See Table 4 for a brief description of the fundamental factors. We note that interaction terms feature prominently in the LASSO model but not in the DNN. Also, the sensitivities to Trading Activity (TrA) and Log Market Cap (MC) are both positive and Log Total Assets (TA) is negative in LASSO and the DNN.*

Table 4: *A short description of the factors used in the Russell* 3000 *deep learning factor model.*

ID	Symbol	**Value Factors**
1	B/P	Book to Price
2	CF/P	Cash Flow to Price
3	E/P	Earning to Price
4	S/EV	Sales to Enterprise Value (EV). EV is given by EV=Market Cap + LT Debt + max(ST Debt-Cash,0), where LT (ST) stands for long (short) term
5	EB/EV	EBIDTA to EV
6	FE/P	Forecasted E/P. Forecast Earnings are calculated from Bloomberg earnings consensus estimates data. For coverage reasons, Bloomberg uses the 1-year and 2-year forward earnings.
17	DIV	Dividend yield. The exposure to this factor is just the most recently announced annual net dividends divided by the market price. Stocks with high dividend yields have high exposures to this factor.
		Size Factors
8	MC	Log (Market Capitalization)
9	S	Log (Sales)
10	TA	Log (Total Assets)
		Trading Activity Factors
11	TrA	Trading Activity is a turnover based measure. Bloomberg focuses on turnover which is trading volume normalized by shares outstanding. This indirectly controls for the Size effect. The exponential weighted average (EWMA) of the ratio of shares traded to shares outstanding: In addition, to mitigate the impacts of those sharp shortlived spikes in trading volume, Bloomberg winsorizes the data: first daily trading volume data is compared to the long-term EWMA volume(180 day half-life), then the data is capped at 3 standard deviations away from the EWMA average.
		Earnings Variability Factors
12	EaV/TA	Earnings Volatility to Total Assets. Earnings Volatility is measured over the last 5 years/Median Total Assets over the last 5 years
13	CFV/TA	Cash Flow Volatility to Total Assets. Cash Flow Volatility is measured over the last 5 years/Median Total Assets over the last 5 years
14	SV/TA	Sales Volatility to Total Assets. Sales Volatility over the last 5 years/Median Total Assets over the last 5 year
		Volatility Factors
15	RV	Rolling Volatility which is the return volatility over the latest 252 trading days
16	CB	Rolling CAPM Beta which is the regression coefficient from the rolling window regression of stock returns on local index returns
		Growth Factors
7	TAG	Total Asset Growth is the 5-year average growth in Total Assets divided by the Average Total Assets over the last 5 years
18	EG	Earnings Growth is the 5-year average growth in Earnings divided by the Average Total Assets over the last 5 years
		GSIC sectorial codes
19-24	(I)ndustry	$\{10, 20, 30, 40, 50, 60, 70\}$
25-35	(S)ub-(I)ndustry	$\{10, 15, 20, 25, 30, 35, 40, 45, 50, 60, 70, 80\}$
36-45	(SE)ctor	$\{10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60\}$
46-49	(I)ndustry (G)roup	$\{10, 20, 30, 40, 50\}$

6 Summary

An important aspect in adoption of neural networks in factor modeling is the existence of a statistical framework which provides the transparency and statistical interpretability of linear least squares estimation. Moreover, one should expect to use such a framework applied to linear data and obtain similar results to linear regression, thus isolating the effects of non-linearity versus the effect of using different optimization algorithms and model implementation environments.

In this paper, we introduce a deep learning framework for fundamental factor modeling which generalizes the linear fundamental factor models by capturing non-linearity, interaction effects and non-parametric shocks in financial econometrics. Our framework provides interpretability, with confidence intervals, and ranking of the factor importances and interaction effects. In the case when the network contains no hidden layers, our approach recovers a linear fundamental factor model. The framework allows the impact of non-linearity and non-parametric treatment of the error on the factors over time and forms the basis for generalized intepretability of fundamental factors. Our neural network model is observed to generate information ratios which are a factor of two higher than OLS. Furthermore, if interaction effects are included in a LASSO model, the DNN is still observed to better capture outliers on account of its universal representation properties.

References

Abadi, M., P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, et al. (2016). TensorFlow: A System for Large-scale Machine Learning. In *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation*, OSDI'16, pp. 265–283.

Ancona, M., E. Ceolini, A. C. Öztireli, and M. H. Gross (2017). A unified view of gradient-based attribution methods for deep neural networks. *CoRR abs/1711.06104*.

Andrews, D. (1989). A unified theory of estimation and inference for nonlinear dynamic models a.r. gallant and h. white. *Econometric Theory 5*(01), 166–171.

Baillie, R. T. and G. Kapetanios (2007). Testing for neglected nonlinearity in long-memory models. *Journal of Business & Economic Statistics 25*(4), 447–461.

Bartlett, P., N. Harvey, C. Liaw, and A. Mehrabian (2017). Nearly-tight vc-dimension bounds for piecewise linear neural networks. *CoRR abs/1703.02930*.

Breiman, L. (1996, Aug). Bagging predictors. *Machine Learning 24*(2), 123–140.

Breiman, L. (2001). Statistical Modeling: The Two Cultures (with comments and a rejoinder by the author). *Statistical Science 16*(3), 199–231.

Carvalho, C. M., H. Lopes, O. Aguilar, and M. Mendoza (2012, 01). Dynamic stock selection strategies: A structured factor model framework. *Bayesian Statistics 9 9*.

Chen, L., M. Pelger, and J. Zhuz (2019, March). Deep learning in asset pricing. Technical report, Stanford University.

Dimopoulos, Y., P. Bourret, and S. Lek (1995, Dec). Use of some sensitivity criteria for choosing networks with good generalization ability. *Neural Processing Letters 2*(6), 1–4.

Dixon, M. (2018a). A high frequency trade execution model for supervised learning. *High Frequency 1*(1), 32–52.

Dixon, M. (2018b). Sequence classification of the limit order book using recurrent neural networks. *Journal of Computational Science 24*, 277 – 286.

Dixon, M., D. Klabjan, and J. H. Bang (2016). Classification-based financial markets prediction using deep neural networks. *CoRR abs/1603.08604*.

Dixon, M. F., N. G. Polson, and V. O. Sokolov (2018). Deep learning for spatio-temporal modeling: Dynamic traffic flows and high frequency trading. *Applied Stochastic Models in Business and Industry 0*(0).

Fama, E. and J. D. MacBeth (1973). Risk, return, and equilibrium: Empirical tests. *Journal of Political Economy 81*(3), 607–36.

Fama, E. F. and K. R. French (1992). The cross-section of expected stock returns. *The Journal of Finance 47*(2), 427–465.

Fama, E. F. and K. R. French (1993). Common risk factors in the returns on stocks and bonds. *Journal of Financial Economics 33*(1), 3 – 56.

Fama, E. F. and K. R. French (2015). A five-factor asset pricing model. *Journal of Financial Economics 116*(1), 1–22.

Fan, J., L. Xue, and J. Yao (2017). Sufficient forecasting using factor models. *Journal of Econometrics 201*(2), 292 – 306. Theoretical and Financial Econometrics: Essays in honor of C. Gourieroux.

Feng, G., J. He, and N. G. Polson (2018, Apr). Deep Learning for Predicting Asset Returns. *arXiv e-prints*, arXiv:1804.09314.

Friedman, J. H. (1991, 03). Multivariate adaptive regression splines. *Ann. Statist. 19*(1), 1–67.

Gallant, A. and H. White (1988, July). There exists a neural network that does not make avoidable mistakes. In *IEEE 1988 International Conference on Neural Networks*, pp. 657–664 vol.1.

Garson, G. D. (1991, April). Interpreting neural-network connection weights. *AI Expert 6*(4), 46–51.

Greenwell, B. M., B. C. Boehmke, and A. J. McCarthy (2018, May). A Simple and Effective Model-Based Variable Importance Measure. *arXiv e-prints*, arXiv:1805.04755.

Gu, S., B. T. Kelly, and D. Xiu (2018). Empirical asset pricing via machine learning. Chicago Booth Research Paper 18-04.

Harvey, C. R., Y. Liu, and H. Zhu (2015, 10). ... and the Cross-Section of Expected Returns. *The Review of Financial Studies 29*(1), 5–68.

Heaton, J. B., N. G. Polson, and J. H. Witte (2017). Deep learning for finance: deep portfolios. *Applied Stochastic Models in Business and Industry 33*(1), 3–12.

Hornik, K., M. Stinchcombe, and H. White (1989, July). Multilayer feedforward networks are universal approximators. *Neural Netw. 2*(5), 359–366.

Hutchinson, J. M., A. W. Lo, and T. Poggio (1994). A nonparametric approach to pricing and hedging derivative securities via learning networks. *The Journal of Finance 49*(3), 851–889.

Kuan, C.-M. and H. White (1994). Artificial neural networks: an econometric perspective. *Econometric Reviews 13*(1), 1–91.

Lee, T.-H., H. White, and C. W. Granger (1993). Testing for neglected nonlinearity in time series models: A comparison of neural network methods and alternative tests. *Journal of Econometrics 56*(3), 269 – 290.

Lo, A. (1994). Neural networks and other nonparametric techniques in economics and finance. In *AIMR Conference Proceedings*, Number 9.

Martin, C. H. and M. W. Mahoney (2018). Implicit self-regularization in deep neural networks: Evidence from random matrix theory and implications for learning. *CoRR abs/1810.01075*.

Mhaskar, H., Q. Liao, and T. A. Poggio (2016). Learning real and boolean functions: When is deep better than shallow. *CoRR abs/1603.00988*.

Mohammadi, S. (2018). A new test for the significance of neural network inputs. *Neurocomputing 273*, 304 – 322.

Montúfar, G., R. Pascanu, K. Cho, and Y. Bengio (2014, Feb). On the Number of Linear Regions of Deep Neural Networks. *arXiv e-prints*, arXiv:1402.1869.

Moritz, B. and T. Zimmermann (2016). Tree-based conditional portfolio sorts: The relation between past and future stock returns.

Nielsen, F. and J. Bender (2010). The fundamentals of fundamental factor models. Technical Report 24, MSCI Barra Research Paper.

Olden, J. D. and D. A. Jackson (2002). Illuminating the 'black box': a randomization approach for understanding variable contributions in artificial neural networks. *Ecological Modelling 154*(1), 135 – 150.

Poggio, T. (2016). Deep Learning: Mathematics and Neuroscience. *A Sponsored Supplement to Science Brain-Inspired intelligent robotics: The intersection of robotics and neuroscience*, 9–12.

Polson, N. and V. Rockova (2018, Mar). Posterior Concentration for Sparse Deep Learning. *arXiv e-prints*, arXiv:1803.09138.

Racine, J. (2001). On the nonlinear predictability of stock returns using financial and economic variables. *Journal of Business & Economic Statistics 19*(3), 380–382.

Rosenberg, B. and V. Marathe (1976). Common factors in security returns: Microeconomic determinants and macroeconomic correlates. Research Program in Finance Working Papers 44, University of California at Berkeley.

Sirignano, J., A. Sadhwani, and K. Giesecke (2016, July). Deep Learning for Mortgage Risk. *ArXiv e-prints*.

Swanson, N. R. and H. White (1995). A model-selection approach to assessing the information in the term structure using linear models and artificial neural networks. *Journal of Business & Economic Statistics 13*(3), 265–275.

Telgarsky, M. (2016). Benefits of depth in neural networks. *CoRR abs/1602.04485*.

Tishby, N. and N. Zaslavsky (2015). Deep learning and the information bottleneck principle. *CoRR abs/1503.02406*.

A Other Interpretability Methods

Partial Dependency Plots (PDPs) evaluate the expected output w.r.t. the marginal density function of each input variable, and allow the importance of the predictors to be ranked. More precisely, partitioning the data X into an *interest* set, $\boldsymbol{X}_s$, and its complement, $\boldsymbol{X}_c = X \setminus \boldsymbol{X}_s$, then the "partial dependence" of the response on $\boldsymbol{X}_s$ is defined as

$$f_s\left(\boldsymbol{X}_s\right) = E_{\boldsymbol{X}_c}\left[\widehat{f}\left(\boldsymbol{X}_s, \boldsymbol{X}_c\right)\right] = \int \widehat{f}\left(\boldsymbol{X}_s, \boldsymbol{X}_c\right) p_c\left(\boldsymbol{X}_c\right) d\boldsymbol{X}_c, \tag{38}$$

where $p_c\left(\boldsymbol{X}_c\right)$ is the marginal probability density of $\boldsymbol{X}_c$: $p_c\left(\boldsymbol{X}_c\right) = \int p\left(\boldsymbol{x}\right) d\boldsymbol{x}_s$. Equation (38) can be estimated from a set of training data by

$$\bar{f}_s\left(\boldsymbol{X}_s\right) = \frac{1}{n}\sum_{i=1}^{n}\widehat{f}\left(\boldsymbol{X}_s, \boldsymbol{X}_{i,c}\right), \tag{39}$$

where $\boldsymbol{X}_{i,c}$ $(i = 1, 2, \ldots, n)$ are the observations of $\boldsymbol{X}_c$ in the training set; that is, the effects of all the other predictors in the model are averaged out. There are a number of challenges with using PDPs for model interpretability. First, the interaction effects are ignored by the simplest version of this approach. While Greenwell et al. (2018) propose a methodology extension to potentially address the modeling of interactive effects, PDPs do not provide a 1-to-1 correspondence with the coefficients in a linear regression. Instead, we would like to know, under strict control conditions, how the fitted weights and biases of the MLP correspond to the fitted coefficients of linear regression. Moreover in the context of neural networks, by treating the model as a black box, it is difficult to gain theoretical insight in to how the choice of the network architecture effects its interpretability from a probabilistic perspective.

Garson (1991) partitions hidden-output connection weights into components associated with each input neuron using absolute values of connection weights. Garson's algorithm uses the absolute values of the connection weights when calculating variable contributions, and therefore does not provide the direction of the relationship between the input and output variables.

Olden and Jackson (2002) determines the relative importance, R_{ij}, of the i^{th} output to the j^{th} predictor variable of the model as a function of the weights, according to the expression

$$R_{ij} = W_{jk}^{(2)} W_{ki}^{(1)}. \tag{40}$$

The approach does not account for non-linearity introduced into the activation, which is the most critical aspects of the model. Furthermore, the approach presented was limited to a single hidden layer.

B Chernoff Bounds on the Variance of the Jacobian

We can also derive probabilistic bounds on the Jacobians. Let $\delta > 0$ and $a_1, \ldots, a_{n-1}$ be reals in $(0, 1]$. Let $X_1, \ldots, X_{n-1}$ be independent Bernoulli trials with $\mathbb{E}[X_k] = p_k$ so that

$$\mathbb{E}[J] = \sum_{k=1}^{n-1} a_k p_k = \mu. \tag{41}$$

The Chernoff-type bound exists on deviations of J above the mean

$$\Pr(J > (1+\delta)\mu) = \left[\frac{e^{\delta}}{(1+\delta)^{1+\delta}}\right]^{\mu}. \tag{42}$$

A similar bound exists for deviations of J below the mean. For $\gamma \in (0, 1]$:

$$\Pr(J - \mu < -\gamma\mu) < \left[\frac{e^{\gamma}}{(1+\gamma)^{1+\gamma}}\right]^{\mu}. \tag{43}$$

These bounds are generally weak and are suited to large deviations, i.e. the tail regions. The bounds are shown in the figure below for different values of μ. Here μ is increasing towards the upper right hand corner of the figure.

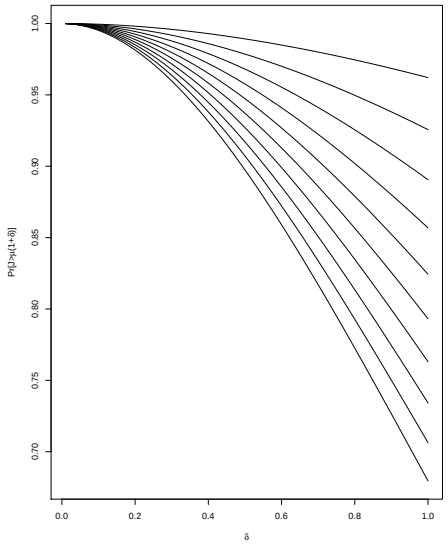


Figure 15: *The Chernoff-type bounds for deviations of J above the mean, μ. Various μ are shown in the plot, with μ increasing towards the upper right hand corner of the plot*