Math. Appl. **11** (2022), 75–106
DOI: 10.13164/ma.2022.07

ON SOME SIMILARITIES AND DIFFERENCES BETWEEN DEEP NEURAL NETWORKS AND KERNEL LEARNING MACHINES

EDDIE PEI AND ERNEST FOKOUÉ

Abstract. This paper presents a thorough computational comparison of the predictive performances of deep neural networks and kernel learning machines. The work featured here successfully establishes that on both real-life datasets and artificially simulated ones, kernel learning machines tend to be just as good as deep neural networks, and quite often outperform them predictively. It turns out from the findings of this paper that while deep neural networks might have worked well on tasks for which millions of observations are available, kernel learning machines just happen to be predictively better on a wide variety of tasks with the kind of sample size that one should realistically expect to have in practice.

1. Introduction

Deep learning allows computational models that are composed of multiple processing layers to learn representations of data with various levels of abstraction. These methods have dramatically improved the state-of-the-art in speech recognition, visual object recognition, object detection, and many other domains such as drug discovery and genomics [33]. It is a fact that deep neural networks have recently gained tremendous popularity thanks mainly to their huge success in applications such as computer vision, image processing, object detection, face recognition, automatic speech recognition, speech synthesis, self driving cars, just to name a few [33, 43]. As a matter of fact, since their inception in the mid-1980s, neural networks have helped with solving a wide variety of significant problems [1,13,19,23,30,31,33,36,43,45,46]. From a methodological perspective, deep neural networks have also infused new energy into the statistical machine learning and artificial intelligence fields, specifically inspiring the inventions and discoveries of new methods and techniques. Despite all the above instances of success of deep neural networks and deep learning, it is important to remember that many other methods predated deep learning, with kernel learning machines being the most prominent of those. With the tremendous popularity of deep learning causing some practitioners and researchers to mistakenly surmise that deep neural networks are the holy grail of data science and artificial intelligence, we have deemed it of great interest to conduct a comprehensive comparison of predictive performances of deep

MSC (2020): primary 68T07, 68T09, 46E22.
Keywords: deep neural networks, kernel learning machines, single hidden layer neural networks, support vector machine, cross validation.

neural networks against kernel learning machines. Specific questions inspired and guided our comparisons, namely:

- What are the similarities and dissimilarities between neural networks and kernel learning machines?
- What is the computational and methodological price deep neural networks need to pay to achieve what is known as their spectacular superiority in a wide range of applications?
- Even more crucially, is it the case that deep neural networks outperform Kernel machines across all possible datasets?

Throughout this paper, we focus solely on supervised learning, with equal emphasis on classification and regression. Specifically, we consider an input space $\mathscr{X}$ and an output space $\mathscr{Y}$, and we seek to build learning machines

$$f : \mathscr{X} \longrightarrow \mathscr{Y}$$

that capture the relationship between the elements of $\mathscr{X}$ and those of $\mathscr{Y}$. Typically, a huge part of the statistical machine learning process consists of choosing/selecting function space or hypothesis space $\mathscr{H}$ from which the learning machine f is drawn, i.e., $f \in \mathscr{H}$, with $\mathscr{H} \subset \mathscr{Y}^{\mathscr{X}}$. Note here that $\mathscr{Y}^{\mathscr{X}}$ is the universal space of all possible functions (mappings) from $\mathscr{X}$ to $\mathscr{Y}$. Using the nomenclature from [16, 17], the learning process proceeds by using random sample

$$\mathscr{D}_n = \{(\mathbf{x}_i, y_i) \overset{iid}{\sim} p_{xy}(\mathbf{x}, y),\ \mathbf{x}_i \in \mathscr{X},\ y_i \in \mathscr{Y},\ i = 1, \ldots, n\}, \tag{1.1}$$

along with the suitably chosen function space $\mathscr{H}$ to empirically construct estimators $\widehat{f}_{\mathscr{H},n} \in \mathscr{H}$ of f. It is important to note that all datasets throughout this paper will be assumed to be random samples generated/drawn according to the joint probability density/mass function $p_{\mathrm{xy}}(\mathbf{x}, y)$ appearing in (1.1). Now, the process of building $\widehat{f}_{\mathscr{H},n}$ from $\mathscr{D}_n$ via some optimization method or algorithm $\mathscr{A}(\mathscr{D}_n; \mathscr{H}, \Lambda)$ is the learning process. In a sense, the actual realized learning machine $\widehat{f}_{\mathscr{H},n}$ is the output of the algorithm used, so that

$$\widehat{f}_{\mathscr{H},n} := \mathscr{A}(\mathscr{D}_n; \mathscr{H}, \Lambda),$$

where Λ at this point generically denotes the collection of all hyperparameters. The theoretical framework of which $\mathscr{A}(\mathscr{D}_n; \mathscr{H}, \Lambda)$ is a manifestation is the so-called theoretical risk minimization principle which is based on the definition of the theoretical risk functional $R(f)$ representing the population error made by a learning machine (function) $f \in \mathscr{Y}^{\mathscr{X}}$. The theoretical risk functional is

$$R(f) = \mathbb{E}\big[\mathcal{L}(Y, f(X))\big] = \int_{\mathscr{X} \times \mathscr{Y}} \mathcal{L}(\mathbf{x}, y) p_{xy}(\mathbf{x}, y)\, \mathrm{d}\mathbf{x}\mathrm{d}y, \tag{1.2}$$

where $\mathcal{L}(\cdot, \cdot)$ is the loss function. The loss function plays a central role in statistical machine learning, as it helps with the evaluation of how well the machine learning model fits the data. More rigorously, a loss function $\mathcal{L}(\cdot, \cdot)$ is a nonnegative bivariate function $\mathcal{L} : \mathscr{Y} \times \mathscr{Y} \longrightarrow \mathbb{R}_+$ such that, given $a, b \in \mathscr{Y}$, the value of $\mathcal{L}(a, b)$ measures the discrepancy between a and b, or the deviance of a from b, or the loss incurred from using b in place of a. For classification learning, the natural loss

function is the so-called zero One Loss defined as follows:

$$\mathcal{L}(y, f(\mathbf{x})) = \mathbb{1}(y \neq f(\mathbf{x})) = \begin{cases} 0 & \text{if} \quad y = f(\mathbf{x}), \\ 1 & \text{if} \quad y \neq f(\mathbf{x}). \end{cases}$$

For regression learning, the most commonly used loss function is the so-called squared error loss, defined as follows:

$$\mathcal{L}(y, f(\mathbf{x})) = (y - f(\mathbf{x}))^2.$$

Although these two functions are the most commonly used in their respective contexts when it comes to evaluating learning machines, many other loss functions exist and are used for a wide variety of specific machine learning scenarios deemed more suitable. Given a suitable chosen loss function along with the risk functional defined in (1.2), one ideally wants to find the universal best function $f^\star \in \mathscr{Y}^{\mathscr{X}}$ that minimizes the risk over all possible functions, i.e.,

$$f^* = \arg\inf_{f \in \mathscr{Y}^{\mathscr{X}}} \big\{ R(f) \big\} = \arg\inf_{f \in \mathscr{Y}^{\mathscr{X}}} \big\{ \mathbb{E}[\mathcal{L}(Y, f(\mathbf{x}))] \big\}. \tag{1.3}$$

Partly due to the fact that the universal space $\mathscr{Y}^{\mathscr{X}}$ is infinitely large, and also crucially to the fact that $p_{xy}(\mathbf{x}, y)$ is never known in practice, the ideal universe best $f^\star$ of (1.3) is never known in practice. Instead, a slightly less intangible way to compare functions (learning machines) is to define the risk function within the function class $\mathscr{H}$, namely

$$R_{\mathscr{H}}(f) = \mathbb{E}\big[\mathcal{L}(Y, f(X)) | f \in \mathscr{H}\big] = \int_{\mathscr{X} \times \mathscr{Y}} \mathcal{L}(\mathbf{x}, y) p_{xy}(\mathbf{x}, y) \, \mathrm{d}\mathbf{x}\mathrm{d}y,$$

and seek $f^\star_{\mathscr{H}} \in \mathscr{H}$ such that

$$f^\star_{\mathscr{H}} = \arg\inf_{f \in \mathscr{H}} \big\{ R_{\mathscr{H}}(f) \big\}. \tag{1.4}$$

Note that $f^\star_{\mathscr{H}} \in \mathscr{H}$ is not as universal as $f^\star \in \mathscr{Y}^{\mathscr{X}}$ because the former is restricted to the function space $\mathscr{H}$ while the latter has no restriction, being universal. Throughout this paper, (1.4) will play a central role, because we will consider different function spaces, and it will be of interest to find out which one performs better. For clarity, given two function spaces $\mathscr{H}_1$ and $\mathscr{H}_2$, we would prefer $\mathscr{H}_1$ over $\mathscr{H}_2$ as the realized best function from $\mathscr{H}_1$ has smaller theoretical risk than the realized best function from $\mathscr{H}_2$. In other words,

$$\text{If} \quad R^\star_{\mathscr{H}_1} < R^\star_{\mathscr{H}_2} \quad \text{choose} \quad \mathscr{H}_1, \tag{1.5}$$

where $R^\star_{\mathscr{H}_1} = R_{\mathscr{H}_1}(f^\star_{\mathscr{H}_1})$ and $R^\star_{\mathscr{H}_2} = R_{\mathscr{H}_1}(f^\star_{\mathscr{H}_2})$. *Specifically, in this paper, we are interested in two function spaces, namely the function space $\mathscr{H}_{\texttt{DNN}}$ of deep neural networks and the function space $\mathscr{H}_{\texttt{KLM}}$ of kernel learning machines. As stated earlier, our overarching goal in this paper is to find out if*

$$R^\star_{\mathscr{H}_{\texttt{DNN}}} < R^\star_{\mathscr{H}_{\texttt{KLM}}}. \tag{1.6}$$

It turns out that checking the inequality in (1.6) as it is constitutes a gigantic theoretical task far beyond the scope of this paper, partly due to the fact that the theoretical manipulations involved are either not even known yet or are very complex. Rather than seeking to compare the true risks for each of the function spaces, the paper adopts the comparison of various statistics around the test error

computed from the given data $\mathscr{D}_n$. Recall that $R_{\mathscr{H}}(f)$ for a function space $\mathscr{H}$ is the generalization error or true error of f in space $\mathscr{H}$. Stochastic Hold Out is widely used for estimating the generalization error of statistical machine learning models as discussed in [17]. As depicted in Algorithm 1, the given dataset $\mathscr{D}_n$ is randomly split into a training and test repeated, and summaries of the replicas of the test error are used as estimates of the generalization error.

Algorithm 1: Stochastic Hold Out for Generalization

for $s = 1$ **to** S **do**

Generate the s^{th} random split of $\mathscr{D}_n$ into $\mathscr{D}_{\texttt{tr}}^{(s)}$ and $\mathscr{D}_{\texttt{te}}^{(s)}$ such that and $|\mathscr{D}_{\texttt{te}}^{(s)}| = (1-\tau)|\mathscr{D}_n|$
and $\mathscr{D}_n = \mathscr{D}_{\texttt{tr}}^{(s)} \cup \mathscr{D}_{\texttt{te}}^{(s)}$, and $n = |\mathscr{D}_n| = |\mathscr{D}_{\texttt{tr}}^{(s)}| + |\mathscr{D}_{\texttt{te}}^{(s)}|$

for $m = 1$ **to** M **do**

Build and refine the m^{th} learning machine $\widehat{f}_m^{(\mathscr{D}_{\texttt{tr}}^{(s)})}(\cdot)$ using $\mathscr{D}_{\texttt{tr}}^{(s)}$
Compute predictions $\widehat{f}_m^{(\mathscr{D}_{\texttt{tr}}^{(s)})}(\mathbf{x}_i)$ for $\mathbf{z}_i \in \mathscr{D}_{\texttt{te}}^{(s)}$
Compute the test error for the m^{th} learning machine

$$\widehat{\varepsilon}_{sm} = \widehat{R}_{\texttt{te}}(\hat{f}_m^{(s)}) = \frac{1}{|\mathscr{D}_{\texttt{te}}^{(s)}|} \sum_{i=1}^{n} \mathbb{1}(\mathbf{z}_i \in \mathscr{D}_{\texttt{te}}^{(s)}) \mathcal{L}(\mathrm{y}_i, \widehat{f}_m^{(\mathscr{D}_{\texttt{tr}}^{(s)})}(\mathbf{x}_i))$$

Essentially then, given a dataset $\mathscr{D}_n$ and a collection of potential function spaces $\mathscr{H}_1, \mathscr{H}_2, \ldots, \mathscr{H}_M$, we set a number S of replications (splits), along with a proportion $\tau \in (0, 1/2)$ of the data $\mathscr{D}_n$, to be allocated to the test set at each split. Upon splitting S times using $\mathscr{D}_n^{(\texttt{s})} = \mathscr{D}_{\texttt{tr}}^{(\texttt{s})} \cup \mathscr{D}_{\texttt{te}}^{(\texttt{s})}$, we create for each dataset $\mathscr{D}_n$, the matrix E of realizations of the test error as seen in (1.7).

$$E = \begin{bmatrix} \varepsilon_{11} & \varepsilon_{12} & \cdots & \varepsilon_{1m} & \cdots & \varepsilon_{1M} \\ \varepsilon_{21} & \varepsilon_{22} & \cdots & \varepsilon_{2m} & \cdots & \varepsilon_{2M} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \varepsilon_{s1} & \varepsilon_{s2} & \cdots & \varepsilon_{sm} & \cdots & \varepsilon_{sM} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \varepsilon_{S1} & \varepsilon_{S2} & \cdots & \varepsilon_{Sm} & \cdots & \varepsilon_{SM} \end{bmatrix} \tag{1.7}$$

For further clarity, it is important to see for the matrix E that

$$\begin{aligned} \varepsilon_{sm} &= \widehat{R}_{\texttt{te}}(\hat{f}_m^{(s)}) = \mathrm{te}(\widehat{f}_m^{(\mathscr{D}_{\texttt{tr}}^{(\texttt{s})})}) \\ &= \text{Error of } \widehat{f}_m^{(\mathscr{D}_{\texttt{tr}}^{(\texttt{s})})}(\cdot) \text{ on } \mathscr{D}_{\texttt{te}}^{(\texttt{s})}. \end{aligned}$$

where

$$\widehat{R}_{\texttt{te}}(f) = \frac{1}{|\mathscr{D}_{\texttt{te}}|} \sum_{j=1}^{n} \mathcal{L}(\mathbf{y}_j, f(\mathbf{x}_j)) \mathbb{1}(\mathbf{z}_j \in \mathscr{D}_{\texttt{te}}),$$

and crucially, $\widehat{f}_m^{(\mathscr{D}_{\texttt{tr}}^{(\texttt{s})})}$ is the best learning machine from function space $\mathscr{H}_m$, learned using the sth training split $\mathscr{D}_{\texttt{tr}}^{(\texttt{s})}$ along with refinement and model selection via

V-fold cross validation.

$$\widehat{f}_m^{(\mathscr{D}_{\mathtt{tr}}^{(\mathtt{s})})} := \underset{\widehat{f}\in\mathscr{H}_m}{\arg\min}\Big\{\mathrm{CV}(\widehat{f}; \mathscr{D}_{\mathtt{tr}}^{(\mathtt{s})})\Big\}$$

It is worth repeating here that ε_{sm}, which is the realized test error of the mth learning machine on the sth replicate (split) of the data $\mathscr{D}_n$, is the most important ingredient for our overarching goal. In order to perform as thorough and complete a comparison of the estimated generalization errors as possible, we consider several statistical summaries of the values of ε_{sm}, $s = 1, \ldots, S$, $m = 1, \ldots, M$. As will be seen in Section 5, we will consider, for each data set, an empirical counterpart of the theoretical generalization errors defined in Equation (1.5), namely, we will define

$$\mathrm{score}(\mathscr{H}_m|\mathscr{D}_n) := \widehat{R}^{\star}_{\mathscr{H}_m} := \text{Empirical counterpart of } R^{\star}_{\mathscr{H}_m} \text{ given } \quad \mathscr{D}_n.$$

For regression learning tasks, we end up considering the following scores. Since we do consider several different datasets in the spirit of testing the so-called no free lunch theorem (NFLT), it makes sense to reveal the dataset in the expression of ε_{sm}, maybe by writing $\varepsilon_{sm}^{(\mathscr{D}_n)}$.

(1) Average Test Error Criterion

$$\mathrm{score}(\mathscr{H}_m|\mathscr{D}_n) := \frac{1}{S}\sum_{\mathtt{s}=1}^{S} \varepsilon_{sm}^{(\mathscr{D}_n)}.$$

(2) Median Test Error Criterion

$$\mathrm{score}(\mathscr{H}_m|\mathscr{D}_n) := \underset{s=1,\ldots,S}{\mathrm{median}}\{\varepsilon_{sm}^{(\mathscr{D}_n)}\}.$$

(3) Maximum Test Error Criterion

$$\mathrm{score}(\mathscr{H}_m|\mathscr{D}_n) := \max_{s=1,\ldots,S}\{\varepsilon_{sm}^{(\mathscr{D}_n)}\}.$$

(4) Minimum Test Error Criterion

$$\mathrm{score}(\mathscr{H}_m|\mathscr{D}_n) := \min_{s=1,\ldots,S}\{\varepsilon_{sm}^{(\mathscr{D}_n)}\}.$$

For any given data set, we will use the above scores as our criteria, and we will declare as winner the hypothesis/function $\mathscr{H}_m$ with the smallest $\mathrm{score}(\mathscr{H}_m|\mathscr{D}_n)$. For classification learning tasks, we will use the same criteria as in regression learning, but this time the test accuracy will replace the test error, namely

$$\pi_{sm}^{(\mathscr{D}_n)} := 1 - \varepsilon_{sm}^{(\mathscr{D}_n)}.$$

Of course, this time, the winner be will the hypothesis space $\mathscr{H}_m$ with the highest score. Equipped with all the above scores and criteria, this paper ultimately performs an empirical comparison of the generalization performances of kernel learning machines against deep neural networks. We use a wide variety of datasets differing in size, shape, and data types, with the finality of finding out if there is any evidence that deep neural networks are predictively better across all datasets as claimed by some practitioners and researchers. For thoroughness and completeness, we include both simulated data and real-world data; we also consider datasets covering many different areas, including medicine, biology, economics

and transportation, just to name a few. This paper does not focus on the methodological and technical subtleties of deep neural networks and their corresponding estimation and prediction mechanisms of deep learning. However, our thorough empirical comparisons of the predictive performances of the machines do provide useful insights into their respective strengths and weaknesses, which we deem of paramount importance to practitioners and researchers. The rest of this paper is organized as follows: Section 2 provides a standard description of the architecture of deep neural networks along with some elements of their fundamental characteristics. Section 3 touches on the general description of the kernel machines explored in this paper, focusing on Gaussian process-inspired learning machines and support vector machines. Section 4 features some theoretical results and relationships mentioned earlier between deep neural networks and kernel learning machines. Section 5 gives a detailed description of the experimental setup of our extensive computational comparisons, focusing on the effect of the ratio of the sample size to the dimension of the input space, on the one hand, and the sheer diversity of entire data sets on the other. Section 5 presents the metrics used in our comparisons of predictive performances. Section 5 also presents the detailed computations with the actual comparisons of predictive performances and all the relevant corresponding comments. Section 6 presents our discussion and conclusion and the points we intend to explore in our future work on this fascinating and fast-developing theme.

2. Neural networks learning machines

According to the comparison criterion defined in Section 1, namely (1.5), we need to define the function space that characterizes deep neural networks (DNN). Using the input space $\mathscr{X}$ and the output space $\mathscr{Y}$, the function space that characterizes a generic DNN is given by (2.1) below:

$$\mathscr{H}_{\text{dnn}} := \Big\{\mathbf{x} \mapsto \psi(\boldsymbol{W}^{(L+1)}\psi(\boldsymbol{W}^{(L)}\psi(\cdots \boldsymbol{W}^{(2)}\psi(\boldsymbol{W}^{(1)}\mathbf{x} + \mathbf{w}_0^{(1)}) + \mathbf{w}_0^{(2)})\cdots) + \mathbf{w}_0^{(L+1)})\Big\}, \tag{2.1}$$

where $\psi(\cdot)$ is the so-called activation function, herein applied vector-wise on the output from the previous layer. These activation functions, especially the ones appearing in the hidden layers, are intended to help capture nonlinearities. The activation vector $\boldsymbol{a}^{(\ell)} = (1, a_1^{(\ell)}, a_2^{(\ell)}, \ldots, a_m^{(\ell)}, \ldots, a_{d_\ell}^{(\ell)})^\top$ in the ℓ^{th} layer allows the component-wise transformation

$$\mathrm{z}_m^{(\ell)} = \mathrm{w}_{m,0}^{(\ell-1)} + \sum_{c=1}^{d_{\ell-1}} \mathrm{w}_{m,c}{}^{(\ell-1)} a_c^{(\ell-1)},$$

which is simply written in vector form as

$$\boldsymbol{z}^{(\ell)} = \boldsymbol{W}^{(\ell-1)}\boldsymbol{a}^{(\ell-1)}, \quad \text{where} \quad a_m^{(\ell)} = h^{(\ell)}(\mathrm{z}_m^{(\ell)}).$$

More succinctly, $h^{(\ell)}(z^{(\ell)}) = \psi(\boldsymbol{W}^{(\ell)}\boldsymbol{z}^{(\ell)} + \mathbf{w}_0^{(\ell)})$. These latent quantities play a central role in the neural networks learning paradigm. It turns out that one of the most challenging bottlenecks with deep neural networks stems from the sheer large numbers of weights to be learned/estimated. In fact, in (2.1), the

collection $\boldsymbol{\theta} := \{\boldsymbol{W}^{(\ell)} : \ \ell = 1, \ldots, L+1\}$ of all the weight matrices forms the set of parameters for the corresponding DNN model.

$$\boldsymbol{W}^{(\ell)} = \begin{bmatrix} \mathrm{w}_{1,0}^{(\ell)} & \mathrm{w}_{1,1}^{(\ell)} & \mathrm{w}_{1,2}^{(\ell)} & \cdots & \mathrm{w}_{1,c}^{(\ell)} & \cdots & \mathrm{w}_{1,d_{\ell-1}}^{(\ell)} \\ \mathrm{w}_{2,0}^{(\ell)} & \mathrm{w}_{2,1}^{(\ell)} & \mathrm{w}_{2,2}^{(\ell)} & \cdots & \mathrm{w}_{2,c}^{(\ell)} & \cdots & \mathrm{w}_{2,d_{\ell-1}}^{(\ell)} \\ \vdots & \vdots & \vdots & \cdots & \ddots & \cdots & \vdots \\ \mathrm{w}_{m,0}^{(\ell)} & \mathrm{w}_{m,1}^{(\ell)} & \mathrm{w}_{m,2}^{(\ell)} & \cdots & \mathrm{w}_{m,c}^{(\ell)} & \cdots & \mathrm{w}_{m,d_{\ell-1}}^{(\ell)} \\ \vdots & \vdots & \vdots & \cdots & \ddots & \cdots & \vdots \\ \mathrm{w}_{d_\ell,0}^{(\ell)} & \mathrm{w}_{d_\ell,1}^{(\ell)} & \mathrm{w}_{d_\ell,2}^{(\ell)} & \cdots & \mathrm{w}_{d_\ell,c}^{(\ell)} & \cdots & \mathrm{w}_{d_\ell,d_{\ell-1}}^{(\ell)} \end{bmatrix}.$$

Considering the "parameter" set $\boldsymbol{\theta}$ for deep neural networks, one of the most immediate remarks about deep neural networks has to do with their complexity or, more simply put, their size or number of parameters needed to be estimated from the data. Clearly, given that each weight matrix $\boldsymbol{W}^{(\ell)}$ is potentially quite large even for modest tasks, it is often the case that $\boldsymbol{\theta}$ has a typically very large number of entries even for very moderate values of L since each $\boldsymbol{W}^{(\ell)}$ is itself already a large $d_\ell \times (d_{\ell-1}+1)$ matrix. If $\dim(\mathscr{X}) = q$, and $\dim(\mathscr{Y}) = r$, and $\dim(\mathscr{Z}_\ell) = d_\ell$, then, for a DNN with L hidden layers, $\boldsymbol{\theta}$ will have $p = q + r + \sum_{\ell=1}^{L} d_\ell \times (d_\ell + 1)$ entries, at least a priori. Now, given a data set

$$\mathscr{D}_n = \{(\mathbf{x}_i, y_i) \overset{iid}{\sim} p_{xy}(\mathbf{x}, y), \ \mathbf{x}_i \in \mathscr{X}, \ y_i \in \mathscr{Y}, \ i = 1, \ldots, n\},$$

the only hope of learning a unique DNN rests on having $n > p$, i.e., more observations than parameters needed to be estimated. In fact, p is typically very large for DNN it is important to have astronomically large amounts of data in deep learning. It is crucial to note that for many tasks of great interest to practitioners, one will typically have situations where $n \lll p$, putting a dent on the potential for DNN to be the right method for such tasks. The activation function is a critical part of deep neural networks. Numerous activation functions have been traditionally used. The logistic sigmoid $\psi(\mathrm{u}) = 1/(1+e^{-\mathrm{u}})$, which is one of the most commonly used activation functions. One of the newest activation functions called the ReLU is defined by [38]:

$$\psi(\mathrm{u}) = \max(0, \mathrm{u}) = (\mathrm{u})_+,$$

preferred because of the efficiency of computation inherent in its use. Now, for a given regression training data set

$$\mathscr{D}_n = \{(\mathbf{x}_i, y_i) \overset{iid}{\sim} p_{xy}(\mathbf{x}, y), \ \mathbf{x}_i \in \mathscr{X}, \ y_i \in \mathscr{Y}, \ i = 1, \ldots, n\}$$

and the loss function $\mathcal{L}(y, f(\mathbf{x}; \boldsymbol{\theta}))$, we want to solve

$$\underset{\boldsymbol{\theta}}{\text{minimize}} \Big\{ \frac{1}{n} \mathcal{L}[y_i, f(\mathbf{x}_i, \boldsymbol{\theta}))] + \lambda J(\boldsymbol{\theta}) \Big\},$$

where λ is a tuning/regularization hyperparameter, and $J(\boldsymbol{\theta})$ is a regularization/penalty term. Dropout is a form of regularization that is heuristically implemented by randomly setting some of the weights to zero to avoid overfitting. *Dropout is more effective than other standard computationally inexpensive regularizers, such as weight decay, filter norm constraints and sparse activity regularization. Dropout may also be combined with other forms of regularization to yield*

a further improvement [21]. Even though dropout appears to work quite well for many DNN architectures, [37] just recently established that dropout fails to regularize for nonparametric learners. Partly in response to the challenges paused by the complexity of deep neural networks, especially in training, different types of neural network architectures have been invented including reservoir computing learning machines, of which echo state networks are a well known example. [51] provide a quick review of the fundamentals of echo state networks as well as a characterization of the statistical and probabilistic properties of the weights of the hidden layers.

3. Kernel learning machines

In the late 1990s, kernel methods and kernel learning machines gained tremendous popularity thanks to Support Vector Machines, Gaussian Process and Radial basis function networks and relevance vector machines a bit later. Kernels and kernel learning machines became so popular that researchers sought to "kernelize" as many existing methods as possible, leading to powerful machines as kernel PCA, kernel kMeans, kernel CCA, kernel regression, just to name a few. In fact, kernels have now touched both supervised and unsupervised learning very deeply, extending even to reinforcement learning and beyond. Kernels have served as the backbone of many data science methods since their very inception [18]. It is worth noting that kernel learning has also been applied to ranking regression [22]. The key lies in the fact that kernel methods consist internally of a transformation of the data into some feature space that usually has a relatively larger dimension. Even though the dimension gets larger, [4] shows that the capacity of a generalization depends on the geometrical characteristics of the training data, and not on the dimension of the input space. Interestingly and somewhat crucially, the direct use of a kernel function reduces the complexity of finding the mapping function [7, 15, 29]. This is known as the kernel trick. Kernel functions allow the implicit computation of the feature space calculations with the function defined in the input space. Kernels are used such that a point in the dataset will affect the nearby points more than it affects the further away points. This is the reason why kernels in the sense that we use them in this paper are essentially *measures of similarities*. For the purposes of the overarching goal of comparing the predictive performances of learning machines, it turns out that, with suitable geometrical characteristics, the generalization error could get smaller with the suitable kernel even though the feature space has a large dimension. What do we really mean by a kernel? It is important to clarify this because the word kernel means different things even in different areas of the same field of Mathematics. Throughout this paper, a kernel $\mathcal{K}$ is a bivariate function defined on the input space. For our purposes, it measures the similarity between two given elements from $\mathscr{X}$. In that sense, the kernel $\mathcal{K}(\cdot, \cdot)$ is defined as

$$\begin{aligned} \mathcal{K}: \quad & \mathscr{X} \times \mathscr{X} \longrightarrow \mathbb{R}_+ \\ & (\mathbf{x}_l, \mathbf{x}_m) \longmapsto \mathcal{K}(\mathbf{x}_l, \mathbf{x}_m). \end{aligned}$$

The so-called Gaussian radial basis function kernel is given by

$$\mathcal{K}(\mathbf{x}_l, \mathbf{x}_m) = \exp\left(-\gamma\|\mathbf{x}_l - \mathbf{x}_m\|^2\right) = \exp\left(-\gamma(\mathbf{x}_l - \mathbf{x}_m)^\top(\mathbf{x}_l - \mathbf{x}_m)\right).$$

This is the most used kernel, viewed as a general purpose kernel typically resorted to as the default kernel by practitioners. γ, in this case, is a function of the bandwidth. Many other kernels exist, some of which will be used in Section 5. Once a kernel $\mathcal{K}$ is chosen for the task of interest, the so-called Gram matrix $\boldsymbol{K} = (\mathcal{K}(\mathbf{x}_l, \mathbf{x}_m))$, $l, m = 1, \ldots, n$ is formed and constitutes the most important object from then on. It is easy to see that the Gram matrix $\boldsymbol{K}$ plays a role similar to the design matrix or data matrix $\boldsymbol{X}$ in linear models.

$$\boldsymbol{K} := \begin{bmatrix} \mathcal{K}(\mathbf{x}_1, \mathbf{x}_1) & \mathcal{K}(\mathbf{x}_1, \mathbf{x}_2) & \cdots & \mathcal{K}(\mathbf{x}_1, \mathbf{x}_n) \\ \mathcal{K}(\mathbf{x}_2, \mathbf{x}_1) & \mathcal{K}(\mathbf{x}_2, \mathbf{x}_2) & \cdots & \mathcal{K}(\mathbf{x}_2, \mathbf{x}_n) \\ \vdots & \vdots & \cdots & \vdots \\ \mathcal{K}(\mathbf{x}_i, \mathbf{x}_1) & \mathcal{K}(\mathbf{x}_i, \mathbf{x}_2) & \cdots & \mathcal{K}(\mathbf{x}_i, \mathbf{x}_n) \\ \vdots & \vdots & \cdots & \vdots \\ \mathcal{K}(\mathbf{x}_n, \mathbf{x}_1) & \mathcal{K}(\mathbf{x}_n, \mathbf{x}_2) & \cdots & \mathcal{K}(\mathbf{x}_n, \mathbf{x}_n) \end{bmatrix}.$$

Some learning machines like support vector machines do indeed require the kernel to be positive definite in order to guarantee many convergence and stability properties of the learning process. Just like we did before with neural networks in Section 2, it is important to specify what the function space looks like when one is dealing with kernel learning machines. Given a dataset

$$\mathscr{D}_n = \{(\mathbf{x}_i, y_i) \overset{iid}{\sim} p_{xy}(\mathbf{x}, y),\ \mathbf{x}_i \in \mathscr{X},\ y_i \in \mathscr{Y},\ i = 1, \ldots, n\}$$

and a suitable chosen kernel $\mathcal{K}$, the corresponding function space is given by

$$\mathscr{H}_{\texttt{klm}} := \mathscr{H}_{\mathcal{K}} := \left\{\mathbf{x} \mapsto \varphi\Big(\sum_{j=1}^{n} \mathrm{w}_j \mathcal{K}(\mathbf{x}, \mathbf{x}_j) + \mathrm{w}_0\Big),\ \mathrm{w}_0 \in \mathbb{R},\ \mathrm{w}_j \in \mathbb{R},\ j \in [n]\right\},$$

$\varphi(\mathrm{v}) = \mathrm{v}$ for regression or $\varphi(\mathrm{v}) = \mathrm{sign}(\mathrm{v})$ for binary classification with labels $\{-1, +1\}$. From a pure representation point of view, regression and classification are not structurally different when it comes to kernel learning machines. Of course, the actual mechanics of learning can differ quite substantially in some cases, especially because the change of loss functions can result in substantial changes in the learning process. The actual development of kernel learning machines is beyond the scope of this paper. We will simply evoke and use all the kernel learning machines of interest in the final forms. It is worth noting immediately that for any $f \in \mathscr{H}_{\texttt{klm}}$, we have

$$f(\mathbf{x}) = \varphi\left(\sum_{j=1}^{n} \mathrm{w}_j \mathcal{K}(\mathbf{x}, \mathbf{x}_j) + \mathrm{w}_0\right), \tag{3.1}$$

so that any kernel learning machine in the sense intended in this work has at most $n + 1$ weights playing the role of "parameters". All the kernel learning machines used throughout this research are of the form depicted in Equation (3.1), regardless of whether the task is regression or binary classification. Multiclass classification is handled slightly differently but with exactly the same main foundational form

as its chief building block. Kernel methods provide very powerful tools in the portfolio of statistical machine learning techniques. As a matter of fact, many algorithms can operate and indeed have operated with kernels as we mentioned earlier. It is well-known that the very popular support vector machines (SVM) are kernel learning machines. However, for our purposes, we will also consider Gaussian processes that are powerful kernel learning machines in their own right. Both of these will ultimately be compared to deep neural networks in Section 4 and Section 5. Support Vector Machines (SVM) are popular machine learning tools for both classification and regression. They were invented by Vladimir Vapnik and Alexis Chervonenkis, and later developed and expanded by several other researchers around the world, after they were found to be extremely powerful on several difficult problems, especially in high dimensional tasks [8, 14, 44]. For a binary classification problem, support vector machines are expressed as follows:

$$\widehat{f}_{\texttt{SVM}}(\mathbf{x}) = \text{sign}\left(\sum_{i=1}^{n} \widehat{\boldsymbol{\alpha}}_i \mathbf{y}_i \mathcal{K}(\mathbf{x}_i, \mathbf{x}) + \widehat{\text{w}}_0\right)$$

$\widehat{\boldsymbol{\alpha}}_i$, $i = 1, \ldots, n$ are the Lagrange multipliers used on the constrained optimization task that defines the celebrated Support Vector Machine. The $\widehat{\boldsymbol{\alpha}}_i$, $i = 1, \ldots, n$ are estimated via quadratic programming. Those vectors $\mathbf{x}_i$ for which $\widehat{\boldsymbol{\alpha}}_i \neq 0$ correspond to the support vectors that gave the name to the machine. $\widehat{f}_{\texttt{SVM}}(\mathbf{x}) \in \{-1, +1\}$ is the kernelized binary classifier's predicted label for the input $\mathbf{x}$ whose true label $\mathbf{y}$ is being predicted. The support vector regression learning machine is of the following form:

$$\widehat{f}_{\texttt{SVM}}(\mathbf{x}) = \sum_{i=1}^{n} (\widehat{\boldsymbol{\alpha}}_i - \widehat{\boldsymbol{\alpha}}_i^{\star}) \mathcal{K}(\mathbf{x}_i, \mathbf{x}) + \widehat{\text{w}}_0,$$

where $\widehat{\boldsymbol{\alpha}}_i, \widehat{\boldsymbol{\alpha}}_i^{\star}$, $i = 1, \ldots, n$ are once again the Lagrange multipliers just like before and are estimated via quadratic programming. A Gaussian Process (GP) is a powerful statistical learning machine [42], that is built from several convenient properties of the multivariate normal (Gaussian) distribution. Gaussian Process models are non-parametric probabilistic models. Gaussian Process models can be naturally expressed in the Bayesian inference framework by using kernel functions or other covariance functions. Just like with Support Vector Machines, it turns out that Gaussian Processes can be conveniently expressed using the form of Equation (3.1). Specifically, for Gaussian Process we have:

$$\widehat{f}_{\texttt{gpr}}(\mathbf{x}) = \varphi\left(\sum_{j=1}^{n} \widehat{\text{w}}_j \mathcal{K}(\mathbf{x}_j, \mathbf{x}) + \widehat{\text{w}}_0\right).$$

For regression under the Gaussian noise model with homoscedastic variance σ^2, $\varphi(\cdot) = \text{id}(\cdot)$, and the weights are given by

$$\widehat{\text{w}}_j = [(\boldsymbol{K} + \sigma^2 \boldsymbol{I})^{-1} \boldsymbol{Y}]_j,$$

where $\boldsymbol{Y} = (y_1, y_2, \ldots, y_n)^{\top}$ is the n-dimensional vector of all the response values. It is important to note that the actual derivation of w_j for Gaussian process regression is far more complex than it appears here [15, 42].

4. Conceptual, methodological and theoretical comparisons

As stated several times up to this point, the motivating theme for the paper is the exploration of similarities and dissimilarities between deep neural networks and kernel learning machines, with the ultimate finality of finding out if deep neural networks just might be the holy grail of statistical machine learning and artificial intelligence. It is important to mention that, along with neural networks, other paradigms have continued to arise, many of which have consistently challenged the claims of superiority attributed to neural networks. Thanks to [41], radial basis function networks are formulated as neural networks with one single hidden layer and later shown to be kernel learning machines, with kernels of the specific type known as radial basis function kernels. It is important to note that many other kernel learning machines exist apart from radial basis function networks, developed from paradigms entirely foreign in principle to neural networks, and many of those kernel learning machines have proven to be excellent learning machines built on very solid and unshakable foundations. While the single hidden layer Neural network enjoys the perch of the universal approximation theorem as its supporting backbone and foundation, it is important to note that kernel learning machines are supported by the equally powerful result known as the representer theorem [9,20,47,48]. It is interesting and somewhat profoundly thought-provoking to note that most of the earlier successes of neural networks all came thanks to the one single hidden layer incarnation, with the so-called multiLayer perceptron (MLP), a single hidden layer feedforward neural network leading the charge. If we denote by $\mathscr{H}_{\texttt{1HLNN}}$, the function space of all one single hidden layer neural networks, then $\forall f \in \mathscr{H}_{\texttt{1HLNN}}$, we must have $\forall \mathbf{x} \in \mathscr{X} \in \mathbb{R}^d$,

$$f(\mathbf{x}) = \varphi\Big(\boldsymbol{W}^{(2)}\psi\Big(\boldsymbol{W}^{(1)}\mathbf{x} + \mathbf{w}_0^{(1)}\Big) + \mathbf{w}_0^{(2)}\Big). \tag{4.1}$$

In fact, it was around this single hidden layer ($L = 1$) neural network that some of the most impressive theoretical foundations of neural networks were set and firmly established. Along with all the successes in a wide variety of applications, various versions and incarnations of the so-called *universal approximation theorem* sprang up, all establishing single hidden layer (1HLNN) neural networks as formidable learning machines capable of learning any smooth function whether in classification or in regression. [3] opened up to the statistics community the desirable functional analysis properties of neural networks as a means of universal approximation of functions. In fact, [2] even provides an earlier foray in connection with insights into the power of neural networks, interestingly based on just a single hidden layer. Arguably one of the earliest if not the absolute earliest account on the universal approximation power of single hidden layer neural networks is provided by [10] and [11]. [32] revisits the universal approximation theorem featuring once again the Single hidden layer multilayer perceptron neural networks, further strengthening the perception and acceptance of single hidden layer neural networks as formidable learning machines. Interestingly, similarities have long been established between single hidden layer neural networks and some kernel learning machines, which begs the question as to whether it even makes sense to fuss about the superiority of one of the learning paradigms over the other. This paper will explore as thoroughly

as possible all the similarities and differences between deep neural networks and kernel learning machines. In their heyday, kernel learning machines, spearheaded by radial basis function networks, also made similar claims of being the holy grail in statistical machine learning and artificial intelligence. Which of the two is better then? Can that question even be answered definitively? In the continuum ranging from rigid parametric models to nonparametric to the so-called semi-parametric models, it might be interesting to revisit the late 1990s theorem that establishes Gaussian processes as the limit of a neural network with an infinite number of nodes in the single hidden layer. Since Gaussian process learning machines are pure nonparametric models, this connection may shed some light. What is the function space/class complexity of these good candidate models? For radial basis function networks, the learning machine is of the form given in Equation (4.2)

$$f(\mathbf{x}) = \varphi\left(\sum_{m=1}^{M} \mathrm{w}_m \psi(\|\mathbf{x} - \mathbf{c}_m\|)\right), \tag{4.2}$$

where $\varphi(\mathrm{v}) = \mathrm{v}$ for regression or $\varphi(\mathrm{v}) = \mathrm{sign}(\mathrm{v})$ for binary classification with labels $\{-1, +1\}$. Interestingly, and somewhat crucially, the basis function $\psi(\cdot)$ admits a reformulation in terms of a suitably chosen kernel $\mathcal{K}(\cdot, \cdot)$, namely

$$\psi(\|\mathbf{x} - \mathbf{c}_m\|) = \mathcal{K}(\mathbf{x}, \mathbf{c}_m) = \exp\left(-\frac{1}{2\tau^2}\|\mathbf{x} - \mathbf{c}_m\|_2^2\right). \tag{4.3}$$

The corresponding radial basis function network is then expressed as

$$f(\mathbf{x}) = \psi\left(\sum_{j=1}^{n} \mathrm{w}_j \mathcal{K}(\mathbf{x}, \mathbf{x}_j) + \mathrm{w}_0\right). \tag{4.4}$$

Note that the summation in (4.4) has n terms and uses the $\mathbf{x}_j$'s, whereas the one in (4.2) uses M terms with $\mathbf{c}_m$ learned from the data. The form, however, is the same, which makes our point of similarity of two learning machines. Through a formulation like (4.3), the radial basis function network which is cast in the context of neural networks finds itself being a bona fide member of the family of kernel learning machines. Note also that Equation (4.1) is similar to Equation (4.2). Interestingly, the similarities do not stop here. In fact, many of the early theoretical results on the universality of neural networks could interchangeably be expressed through radial basis function networks or through the kernel learning machines equivalent, or counterparts.

Theorem 4.1. [11] *Let $f : \mathscr{X} \longrightarrow \mathscr{Y}$ be a function of interest to be estimated. Assume that for all $\mathbf{x} \in \mathscr{X}$, $\mathbb{E}[f(\mathbf{x})] < \infty$, and further assume that we have a function $\psi : \mathbb{R} \longrightarrow \mathbb{R}$ that is continuous with*

$$\lim_{\mathbf{x} \to +\infty} \psi(\mathbf{x}) \to 1 \quad \textit{and} \quad \lim_{\mathbf{x} \to -\infty} \psi(\mathbf{x}) \to 0.$$

Then, for any $\varepsilon > 0$, there exists $n = n(\varepsilon)$ such that

$$\inf_{\{(a_i, b_i, \mathbf{w}_i)\}} \mathbb{E}\left\{\left|f(\mathbf{x}) - \frac{1}{n}\sum_{i=1}^{n} a_i \psi(\langle \mathbf{w}_i, \mathbf{x}\rangle + b_i)\right|\right\} \leq \varepsilon.$$

Several other researchers like [26] and [25] have contributed theoretical results of the same type around the approximation capabilities of multilayer feedforward networks, typically featuring the single hidden layer architecture. At this point, it might appear (and rightly so) more interesting to compare single hidden layer neural networks to kernel learning machines, thanks in part to intrinsic similarities but also because both paradigms are supported by strong theoretical foundations. Such comparisons will be carried out computationally later. Two natural questions arise in the presence of such an emphatic result as the universal approximation power of single hidden layer neural networks, namely:

(a) How can anyone justify the need/importance to study or develop any other learning paradigm if one exists that is a kind of panacea?
(b) Even within the neural networks paradigm, why should anyone use more than one hidden layer when one hidden layer has all the approximation power?

Owing to the rise to prominence of deep neural networks, i.e., neural networks with more than one hidden layer, it is important to extend our comparison to deep neural networks. Hence our overarching question, namely: *Are Deep Neural Networks (DNN) really predictively better than kernel learning machines (KLM) across the board?* This thought-provoking question, inspired by the surge in interest around Deep Neural Networks, has led many researchers to seriously investigate the predictive performances of deep neural networks relative to other learning paradigms. Specifically, in recent months, a number of authors have contributed several accounts (most of them of a computational nature) of the predictive comparisons between deep neural networks and kernel learning machines. [39] provides the seminal account through which Gaussian processes are linked to single hidden layer neural networks with an infinite number of nodes in the hidden layer. He specifically established that when the number of nodes in the single hidden layer of a neural network is allowed to grow to infinity, then one can use a Gaussian process prior over such an infinite network and derive Gaussian process learning machine as a regularized Neural Network. Considering the fact that Gaussian process learning machines are kernel learning machines, this result is quite arguably the first and most foundational foray into the now prevalent and widespread effort aimed at comprehending deep neural networks via Gaussian processes. Amongst others [5] and [24] are noteworthy. [5] declares that *to understand deep learning we need to understand kernel learning.* Taking a computational approach, [24] provide a comparison that is very similar in structure and in spirit to ours. Their very title *Deep neural networks and kernel regression achieve comparable accuracies for functional connectivity prediction of behavior and demographics* presents the very same conclusion we end up arriving at, namely that the two learning paradigms are very similar in terms of predictive performances. More recent papers like [35] centered around Wide Neural Networks and [12] discussing Gaussian process behavior in Wide deep neural networks, can be rightly viewed as extensions of [39] this time adapted to Deep Neural Network architectures. Authors like [28] are actively continuing the burgeoning work around the so-called neural tangent kernel (NTK) used as one of the most promising ways of establishing the link between deep neural networks and kernel learning machines, with the hope of establishing with DNN the kind of firm similarity and analogy enjoyed by single

hidden layer neural networks. Very early on, [6] proposed *Kernel Methods for Deep Learning*, establishing somehow a strong connection between the two paradigms. Both [27] and [28] introduce the neural tangent kernel, which provides one of the strongest similarity between deep neural networks and kernel learning machines. This work is truly revolutionary and transformative as it established the strongest connection yet between the two paradigms. [35] proposes the exploration of *Wide Neural Networks of Any Depth Evolve as Linear Models Under Gradient Descent*, following [34], who formulated *Deep Neural Networks as Gaussian Processes*. Even more recently, [52] emphasizes that *Understanding Deep Learning (Still) Requires Rethinking Generalization*. From a theoretical and methodological perspective, it appears that deep neural networks need kernel learning machines more than the other way around, since we see from many authors that the formulation of DNN via kernels is the pathway towards understanding DNN. In that sense, one may declare the upper hand to kernel learning machines, although the equivalence of formulation makes it hard to give the upper hand to any of the two paradigms. Maybe the key lies with the kernel itself, in the sense that while the neural tangent kernel might be a suitable device/mechanism for formulating a DNN as a kernel learning machines, it might not be straightforward to compute it. Therefore, one could see a methodological advantage in simply adopting traditional shallow kernels rather than using the neural tangent kernel. Of course, since choosing the suitable kernel is akin to model selection, which in turn has to do with the typically unknown geometry of the pattern underlying the data, one could conceive of situations where the neural tangent kernel is the optimal choice, albeit with the challenges of its computation. Finally, it is crucial to mention that classical kernel learning machines commonly used typically require the estimation of at most $n + 1$ weights, whereas the number of weights needed for deep neural networks can quickly grow to extremely large numbers. Each of the weight matrices $\boldsymbol{W}^{(\ell)}$ is itself already a large $d_\ell \times (d_{\ell-1} + 1)$ matrix. If $\dim(\mathscr{X}) = q$, and $\dim(\mathscr{Y}) = r$, and $\dim(\mathscr{Z}_\ell) = d_\ell$, then, for a DNN with L hidden layers, $\boldsymbol{\theta}$ will have $p = q + r + \sum_{\ell=1}^{L} d_\ell \times (d_\ell + 1)$ entries, at least a priori. This makes a generic DNN far more complex than a classical kernel learning machine. We will see in Section 5 that this complexity often hurts DNN when there is not enough data to learn all the weights consistently.

5. Computational explorations and demonstrations

As stated earlier, the chief goal of this paper is to perform computational comparisons on the predictive performances of both deep neural networks and kernel learning machines. A rather huge challenge in this regard comes from the fact that, due to the complex structure and the typically large number of parameters inherent in deep neural networks, tuning the parameters is very time-consuming and requires plenty of computational power, sometimes preventing the comparison altogether. As we mentioned earlier, we use both simulated and real-world datasets in this paper. We typically randomly split the data into 70% for training with the remaining 30% for testing the predictive performance. For each data set, we run the random split 50 times, and then compute the predictive measures of interest based on the 50 replications. To make sure it is objective to all the

machines, we normalize the data before we run the methods. We compare the test errors among five machines: Gaussian process with radial basis function kernel, support vector machine with the linear kernel; support vector machine with the polynomial kernel; support vector machine with radial basis function kernel, and deep neural networks. In the rest of this section, we will show the details of the data sets, experiments, and results. As we indicated right from Section 1, all the learning machines considered are tuned and selected via cross validation. Cross validation is a well-known and widely used method for tuning the hyperparameters in statistical learning and data mining. It divides the data into V chunks, and each chunk has almost equal portions, $\mathscr{D}_n = \bigcup_{v=1}^{V} \mathscr{D}_v$, holding out the portion and fitting the model from the rest of the data. We then use the fitted model to predict the holdout samples and the average measure of predictive performance over the V different fits to get the cross validation score, which is given by

$$CV(\hat{f}) = \frac{1}{V} \sum_{v=1}^{V} \hat{\xi}_v,$$

where

$$\xi_v = \frac{1}{|\mathscr{D}_v|} \sum_{i=1}^{n} 1(\mathbf{z}_i \in \mathscr{D}_v) L(\mathbf{y}_i, \hat{f}^{(-\mathscr{D}_v)}(\mathbf{x}_i))$$

Algorithm 2: V-fold cross validation

Input: Training data $\mathscr{D}_n = \{\mathbf{z}_i = (\mathbf{x}_i^\top, \mathbf{y}_i), i = 1, 2, \ldots, n\}$,where $\mathbf{x}_i^\top \in \mathscr{X}, \mathbf{y}_i \in \mathscr{Y}$, the function of interest is denoted by f, V : the number of fold, n: sample size.

Output: Cross Validation score $CV(\hat{f})$

for v=1 to V do

Extract the validation set
$\mathscr{D}_v = \mathbf{z}_i \in \mathscr{D}_n : i \in [1 + (v-1) \times m, v \times m]$
Extract the training set $\mathscr{D}_v^c := \mathscr{D}_n / \mathscr{D}_v$
Build the estimator $\hat{f}^{(-\mathscr{D}_v)}(\cdot)$ using $\mathscr{D}_v^c$
Compute predictions $\hat{f}^{(-\mathscr{D}_v)}(\mathbf{x}_i)$ for $\mathbf{z} \in \mathscr{D}_v$
Compute the validation error for the vth chunk

$$\xi_v = \frac{1}{|\mathscr{D}_v|} \sum_{i=1}^{n} 1(z_i \in \mathscr{D}_v) L(y_i, \hat{f}^{(-\mathscr{D}_v)}(x_i))$$

Compute the output CV score

$$CV(\hat{f}) = \frac{1}{V} \sum_{v=1}^{V} \hat{\xi}_v$$

Cross validation is one of the main techniques we used in this research to tune the parameters for the machines. For instance, in Figure 1, it shows tuning the parameters for the support vector machine with Gaussian kernel, we could directly visualize the best parameter in the plot, which is the lowest point.

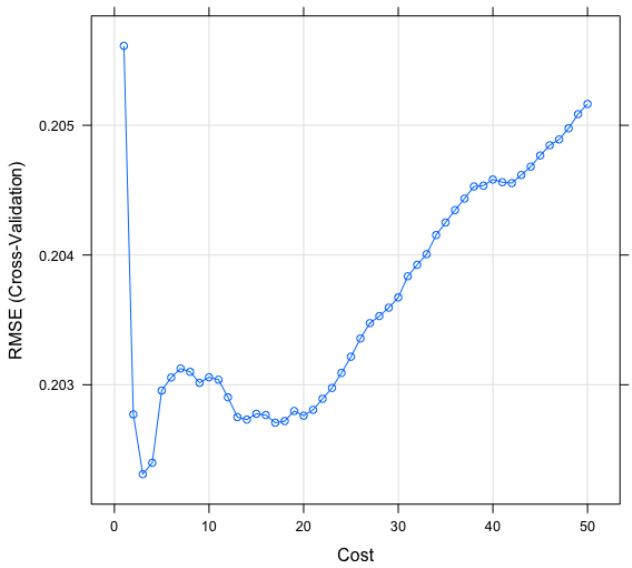


Figure 1. 5-fold cross validation to choose parameter for SVM with gaussian kernel

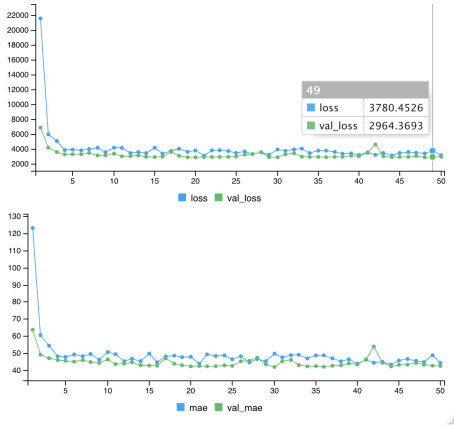
Grid Search is a naive but straightforward approach to trying every possible configuration. Since deep neural networks have many parameters (for example: learning rate, dropout rate, batch size, etc.), the challenge of this approach is "the curse of dimensionality". This means that the more dimensions we add, the more the search will explode in time complexity and computing power. For example, we want to create four hidden layers Neural Networks model, and for each layer, we add a dropout rate, also add a batch size to tune, and each parameter we choose from 2 potential good candidates, then we will need to run $2^9 = 512$ times. Because of this limitation the dimensions used in the experiments are less than or equal to 4, and the number of neurons in each layer goes from 16 to 256 in this research. The R package "tfruns" is a suite of tools for tracking, visualizing, and managing TensorFlow training runs. Figure 4a shows the tuning error; we want two losses as close as it could be. Figure 2b shows the saved hyperparameters. Using this table, we could find the best combination of the hyperparameters.

5.1. Simulation study

5.1.1. Regression example. For our first example, we consider 25-dimensional multiple linear regression model with Gaussian noise. Specifically, $\mathbf{Y} \in \mathbb{R}^{25}$ is given by

$$\mathbf{Y} = \mathbf{x}^\top \boldsymbol{\beta}^\star + \varepsilon$$

Let $\boldsymbol{\beta}^\star = (\beta_1, \beta_2, \ldots, \beta_p)^\top$. Let us consider an example where $\boldsymbol{\beta}^\star \in \mathbb{R}^p$, with $p = 25$, $\beta_j^\star = 0$ for $j \notin \{3, 5, 7, 11, 13, 17, 19\}$, except for $\beta_3^\star = 0.3$, $\beta_5^\star = 0.5$,



	metric_accuracy	metric_val_accuracy	flag_dense_units1	flag_dense_units2	flag_dense_units3	flag_dense_units4	flag_dense_units5	flag_drop
1	0.6513	0.5146	128	64	64	16	8	
2	0.633	0.4951	256	64	64	16	8	
3	0.6343	0.5243	128	128	64	16	8	
4	0.6501	0.4563	256	128	64	16	8	
5	0.6403	0.4612	128	64	32	16	8	
6	0.6391	0.5146	256	64	32	16	8	
7	0.6476	0.4272	128	128	32	16	8	
8	0.6209	0.4854	256	128	32	16	8	
9	0.6561	0.5146	128	64	64	32	8	
10	0.6593	0.432	256	64	64	32	8	
11	0.6659	0.432	128	128	64	32	8	
12	0.6719	0.4709	256	128	64	32	8	
13	0.6306	0.466	128	64	32	32	8	
14	0.633	0.4806	256	64	32	32	8	
15	0.6428	0.4515	128	128	32	32	8	
16	0.6756	0.4854	256	128	32	32	8	
17	0.6403	0.4903	128	64	64	16	8	
18	0.644	0.4757	256	64	64	16	8	
19	0.6051	0.5388	128	128	64	16	8	
20	0.6245	0.5097	256	128	64	16	8	
21	0.6221	0.4806	128	64	32	16	8	
22	0.6075	0.5	256	64	32	16	8	
23	0.6282	0.4612	128	128	32	16	8	
24	0.6379	0.4854	256	128	32	16	8	
25	0.6464	0.3786	128	64	64	32	8	

(a) DNN **(b)** hyperparameters Table

Figure 2. Tuning hyperparameters for the Deep Neural Networks

$\beta_7^\star = -0.7$, $\beta_{11}^\star = 0.11$, $\beta_{13}^\star = 0.13$, $\beta_{17}^\star = 0.17$, $\beta_{19}^\star = 0.19$. By changing the number of data points (table 1), we get six sets of data. We run five machines, then compare the test errors: (a) average test errors; (b) median test errors. Figure 3 shows the median and mean of 50 runs of the Mean Squared Error (MSE) on the test data; based on the plots, we could see all the methods improve tremendously when sample size n changes from 25 to 50, SVM with linear kernel, Gaussian kernel and polynomial kernel slightly improve after $n = 500$ compared to DNN and GP. All the SVM with kernel outperformed the DNN, GP is catching up with DNN at 250 data points, and starts to get better than DNN after that.

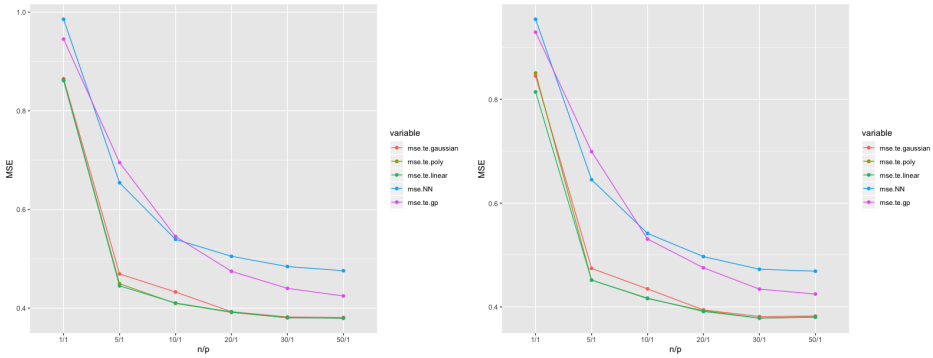
Table 1. Simulation data for regression Example

SN	partition	n	p	$\kappa = n/p$
1	regression-25	25	25	1/1
2	regression-125	125	25	5/1
3	regression-250	250	25	10/1
4	regression-500	500	25	20/1
5	regression-750	750	25	30/1
6	regression-1250	1250	25	50/1

5.1.2. Classification example. In the classification simulation study, we generate 100 variables. All variables are generated from a multivariate Gaussian distribution, and the response variable Y is generated from a Bernoulli distribution with the probability of success given by the function:

$$(Y|\mathbf{x}) \sim \text{Bernoulli}(\pi(\mathbf{x})) \quad \text{where} \quad \pi(\mathbf{x}) = \Pr(Y = 1|\mathbf{x}) = \frac{1}{1 + e^{-\mathbf{x}^\top \boldsymbol{\beta}}}.$$

Changing the number of data points from 25 to 500, which will give us different values of $\kappa = \frac{1}{4}, \frac{1}{2}, \frac{1}{1}, \frac{2}{1}, \frac{5}{1}$. κ is an essential parameter in this paper, and it



(a) Mean MSE plot **(b)** Median MSE plot

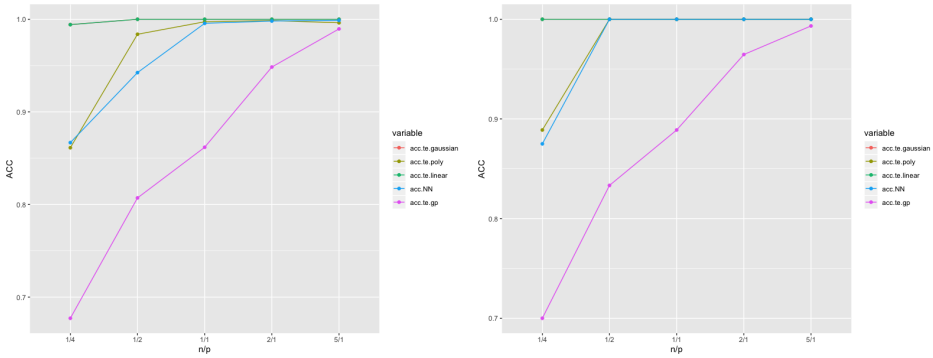
Figure 3. These two plots are the simulation regression results for 5 machines. The left image shows the mean MSE in the 50 runs. The right image shows the median MSE in the 50 runs.

shows the models' sensitiveness of the sample size. Table 2 shows the details of the data used in the simulation classification study. We compare the test error: Accuracy(ACC). We compare the median and mean of 50 runs of the accuracy on the test data. Figure 4 shows the results. Based on the plots, we could see most of the methods behaved much better when we increase the sample size from 25 to 50. SVM with linear kernel and Gaussian kernel could even reach 100 % accuracy nearly all the time. DNN is not a very competitive machine at first, but it starts to catch up when we increase the sample size to 100. SVM with linear kernel works very well on this data compared to the other machines. GP is not very competitive for this data set but eventually caught up when we increased the data to 500.

Table 2. Simulation data for classification example

SN	partition	n	p	$\kappa = n/p$
1	classification-25	25	100	1/4
2	classification-50	50	100	1/2
3	classification-100	100	100	1/1
4	classification-200	200	100	2/1
5	classification-500	500	100	5/1

In the simulation study, both the regression results and the classification results show that kernel machines have a better output than Deep Neural Networks most of the time. Compared to the kernel models, DNN's behavior depends more on the richness of the sample. In the next section, we present more results based on the real-world data.



(a) Mean ACC plot **(b)** Median ACC plot

Figure 4. These two plots are the simulation classification results for 5 machines. The left image shows the mean ACC in the 50 runs. The right image shows the median ACC in the 50 runs.

5.2. Computational explorations for real life data

For the real-world data, we used 10 regression datasets, and 16 classification data sets that included binary classification and multi-classification data. We thank [40] and MNIST (public data) for the contribution of the data sets. We are trying to cover as many areas as possible, for instance: medical, traffic, economic, etc. Tables 3 and 4 briefly describe the data sets, the sample size, the number of response variables and $\kappa = \frac{n}{p}$, which represents the informational richness. For the classification data there are three comparison groups: nine data sets for the binary classification, three data sets for the multi-classification, and four data sets for the parameter κ's influence.

Table 3. Regression Datasets table

SN	Dataset	n	p	$\kappa = n/p$
1	Airfoil	1503	6	250.5
2	Auto mpg	392	8	49
3	computer hardware	209	7	29.86
4	Concrete strength	1030	9	114.44
5	Diabetes	442	11	40.18
6	Ducan MBA	203	7	29
7	GPA	141	5	28.2
8	hprice	506	6	84.33
9	Istanbul stock	536	8	67
10	Mortality	992	4	233.75

Table 4. Classification Datasets table

SN	Dataset	n	p	Number of classes	$\kappa = n/p$
1	Asthmatic	405	11	2	36.82
2	Breast cancer	569	10	2	56.90
3	Congressional voting	435	17	2	25.59
4	Cryotherapy	90	7	2	12.86
5	Social network	400	4	2	100
6	Gender voice	3168	21	2	150.86
7	Diabetic	1151	20	2	57.55
8	Sonar	208	61	2	3.41
9	Indian Liver Patient	538	11	2	53
10	Balance scale	625	5	3	125
11	Cars	1728	7	4	246.86
12	Seeds	210	8	3	26.25
13	MNIST-112	112	784	10	1/7
14	MNIST-300	300	784	10	0.38(appro 3/7)
15	MNIST-784	784	784	10	1
16	MNIST-1586	1586	784	10	2

5.3. Regression results

For the regression study, there are ten data sets to be used to compare the five machines. The distribution of test errors of all the machines over 50 replications is shown in Figure 5. Among the ten boxplots, we could see the SVM with kernels are mostly better than DNN and have a smaller range compared to DNN. DNN only outperformed GP once (the 3rd boxplot). DNN did slightly better than SVM with the polynomial kernel. In the regression comparison, we would evaluate the minimum, median, maximum and mean of the test error which is shown in Tables 5, 6, 8 and 7. Among all the tables, SVM with Gaussian kernel wins seven times in the minimum table, six times in the median table, seven times in the mean table, and seven times in the maximum table, which makes SVM with Gaussian kernel the winner among the five learning machines. GP wins two times in the median table and maximum table, and one time in the mean table. DNN only wins two times in the minimum table, and there is a tie with the SVM with linear kernel.

5.4. Classification results

5.4.1. Binary classification results. In this section, we test five machines on nine binary classification datasets, we compare the minimum, median, maximum and mean of the test error which shows in Tables 12, 10, 11 and 9. We also generate distributions of the test error over 50 replications, which is shown in Figure 6. The boxplots show that DNN is comparable with other machines. In the binary classification test, DNN performs better than in the regression test and has a relatively smaller range of test error. In the maximum comparison table, SVM-RBF wins four times; SVM-poly and SVM-linear each win three times; GP wins two times, and DNN wins one time, but it ties with GP in this time. In

Table 5. Regression minimum MSE table

Dataset	SVM-rbf	SVM-poly	SVM-linear	DNN	GP
Airfoil	0.335	0.378	0.370	**0.253**	0.346
Auto mpg	**2.33**	2.50	2.75	2.65	2.38
computer hardware	**28.41**	31.55	31.58	31.71	33.89
Concrete	**5.65**	6.09	9.56	6.72	6.45
Diabetes	49.66	50.33	**49.58**	50.28	50.59
Ducan MBA	0.168	0.177	**0.166**	**0.166**	0.174
GPA	**0.265**	0.270	0.272	0.350	0.271
hprice	**2839.00**	3075.46	4311.29	3613.33	3709.67
Istanbul stock	**0.0124**	**0.0124**	**0.0124**	0.0137	0.0143
Mortality	**0.00008**	**0.00008**	0.00045	0.00225	0.00012

Table 6. Regression mediam MSE table

Dataset	SVM-rbf	SVM-poly	SVM-linear	DNN	GP
Airfoil	0.49	0.53	0.53	0.54	**0.47**
Auto mpg	**2.84**	3.01	3.43	3.47	2.97
computer hardware	**55.37**	73.00	68.19	64.22	124.89
Concrete	**6.39**	7.89	10.82	8.87	7.23
Diabetes	**55.42**	56.48	56.32	58.87	56.30
Ducan MBA	**0.21**	**0.21**	**0.21**	0.36	0.22
GPA	0.36	**0.35**	**0.35**	0.49	0.37
hprice	5077.29	5058.32	5631.73	5291.71	**4966.83**
Istanbul stock	**0.01**	**0.01**	**0.01**	0.02	0.02
Mortality	10.12	9.70	**9.16**	12.18	9.66

Table 7. Regression mean MSE table

Dataset	SVM-rbf	SVM-poly	SVM-linear	DNN	GP
Airfoil	0.49	0.54	0.55	0.53	**0.47**
Auto mpg	**2.89**	3.03	3.44	3.80	2.95
computer hardware	**58.83**	79.93	71.27	68.96	127.09
Concrete	**6.42**	7.61	10.81	9.25	7.25
Diabetes	**55.18**	55.92	55.98	59.19	56.32
Ducan MBA	**0.21**	0.22	**0.21**	0.37	0.22
GPA	0.36	**0.35**	**0.35**	0.53	0.36
hprice	**4928.90**	4893.10	5577.50	5155.59	4989.48
Istanbul stock	**0.01**	**0.01**	**0.01**	0.02	0.02
Mortality	9.50	8.76	**8.30**	12.03	8.89

the medium comparison table, SVM-poly and GP each win three times SVM-RBF wins two times, SVM-linear wins one time, DNN wins 0 times. SVM-poly

Table 8. Regression maximum MSE table

Dataset	SVM-rbf	SVM-poly	SVM-linear	DNN	GP
Airfoil	0.67	0.75	0.75	0.74	**0.66**
Auto mpg	**3.69**	3.88	4.22	6.50	4.06
computer hardware	**112.15**	156.68	130.46	119.84	222.77
Concrete	**7.72**	8.70	12.73	16.26	8.21
Diabetes	**61.15**	62.05	62.05	80.75	61.89
Ducan MBA	0.29	0.29	**0.25**	0.77	0.26
GPA	**0.44**	0.46	0.45	0.97	0.43
hprice	**5907.38**	5918.57	6703.37	6278.60	5982.95
Istanbul stock	**0.02**	**0.02**	**0.02**	0.03	**0.02**
Mortality	18.95	18.64	**12.86**	39.97	13.33

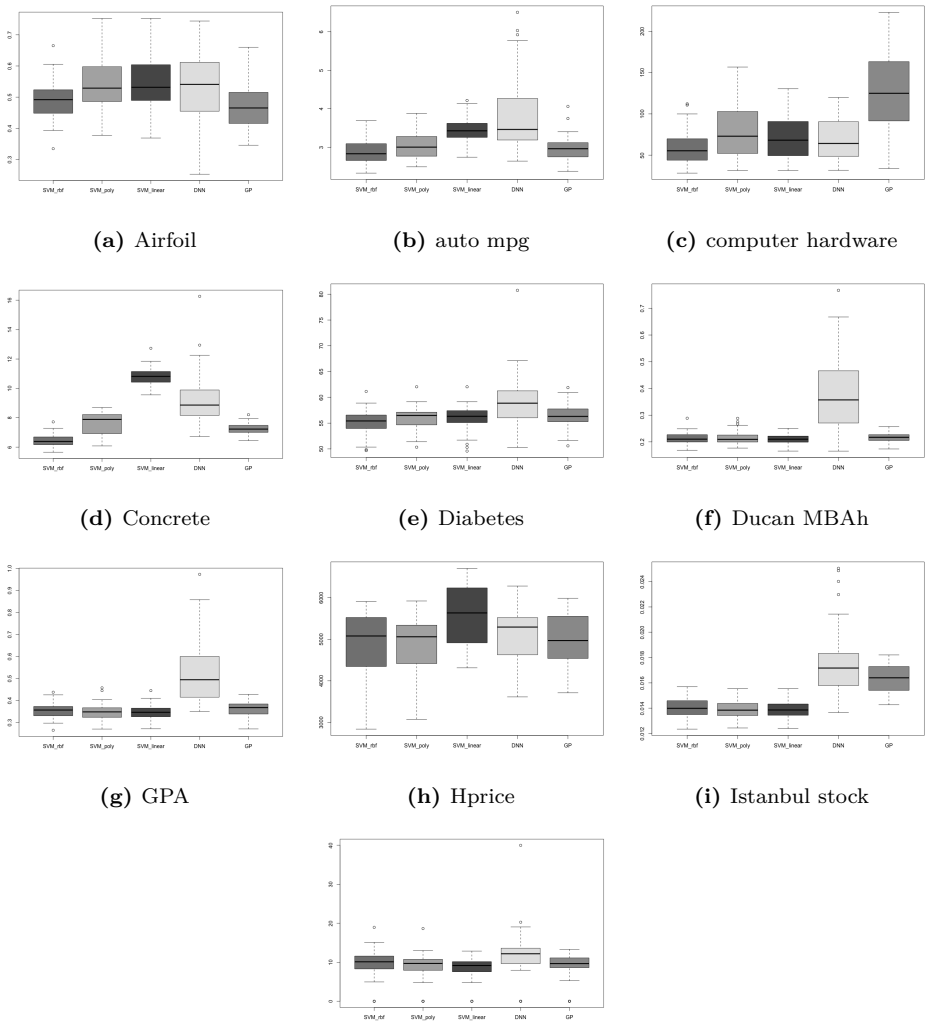
wins four times in the mean table, DNN once but ties with SVM-poly and SVM-RBF in this one time, GP wins three times in the mean table. For the minimum table, DNN only wins once, SVM-poly wins five times. So based on all the binary classification results, SVM-poly is the winner among the five machines; DNN does not outperform the other machines. Even though the Kernel machines still seem to give a better predictive performance based on the tables and the boxplot, DNN is still comparable with other machines in these cases.

Table 9. Binary Classification maximum accuracy results table

Dataset	SVM-rbf	SVM-poly	SVM-linear	DNN	GP
Asthmatic	**0.913**	0.898	**0.913**	0.891	0.896
breast cancer	**0.988**	0.981	0.981	0.987	0.982
congressional voting	**0.992**	**0.992**	**0.992**	0.986	0.976
cryotherapy	0.935	0.969	0.960	**1.000**	**1.000**
Social network	0.954	**0.969**	0.908	0.962	0.956
gender voice	0.984	**0.992**	0.981	0.981	0.984
Diabetic	0.785	0.787	**0.788**	0.771	0.764
Sonar	**0.948**	0.944	0.865	0.865	0.900
Indian Liver Patient	0.753	0.781	0.781	0.750	**0.803**

5.4.2. Multi classification results. In the multi-class cases, we use seven datasets, divided into two sections. The first section contains three datasets, and the second section contains four datasets that explicitly test the κ effect of the prediction performance.

In the first section, we use three datasets to compare the minimum, median, maximum, and mean of the test error. Tables 13, 14, 15 and 16 show the result. SVM-RBF wins two times out of three in the maximum and minimum comparison table and three times in the medium and mean comparison table, making it the winner in the multi-classification test. DNN and GP do not win even one time among the four comparison tables. However, based on the value of the 4 ACC



(a) Airfoil **(b)** auto mpg **(c)** computer hardware

(d) Concrete **(e)** Diabetes **(f)** Ducan MBAh

(g) GPA **(h)** Hprice **(i)** Istanbul stock

(j) Mortality

Figure 5. Comparison boxplots of MSE for 10 regression data sets, where x-axis is the name of the machines, y-axis is MSE.

tables, most machines have an above 95% ACC value, which is an excellent result for the multi-classification problem, and SVM-RBF has a couple 100% accuracy, which makes this machine excellent for these data sets.

5.4.3. Predictive performances on MNIST and a function of κ. Although the current study suggests that the kernel machines and DNN models perform very similarly, the predictive performance of DNN seems more dependent on the

Table 10. Binary Classification median accuracy results table

Dataset	SVM-rbf	SVM-poly	SVM-linear	DNN	GP
Asthmatic	0.837	0.838	**0.845**	0.823	0.830
breast cancer	0.961	**0.962**	0.958	0.957	0.958
congressional voting	**0.958**	0.955	0.957	0.953	0.947
cryotherapy	0.849	0.830	0.862	0.880	**0.895**
Social network	0.890	0.899	0.824	0.891	**0.902**
gender voice	0.976	**0.981**	0.975	0.970	0.972
Diabetic	**0.742**	0.734	0.741	0.707	0.692
Sonar	0.817	**0.857**	0.764	0.771	0.789
Indian Liver Patien	0.698	0.705	0.703	0.697	**0.708**

Table 11. Binary Classification mean accuracy results table

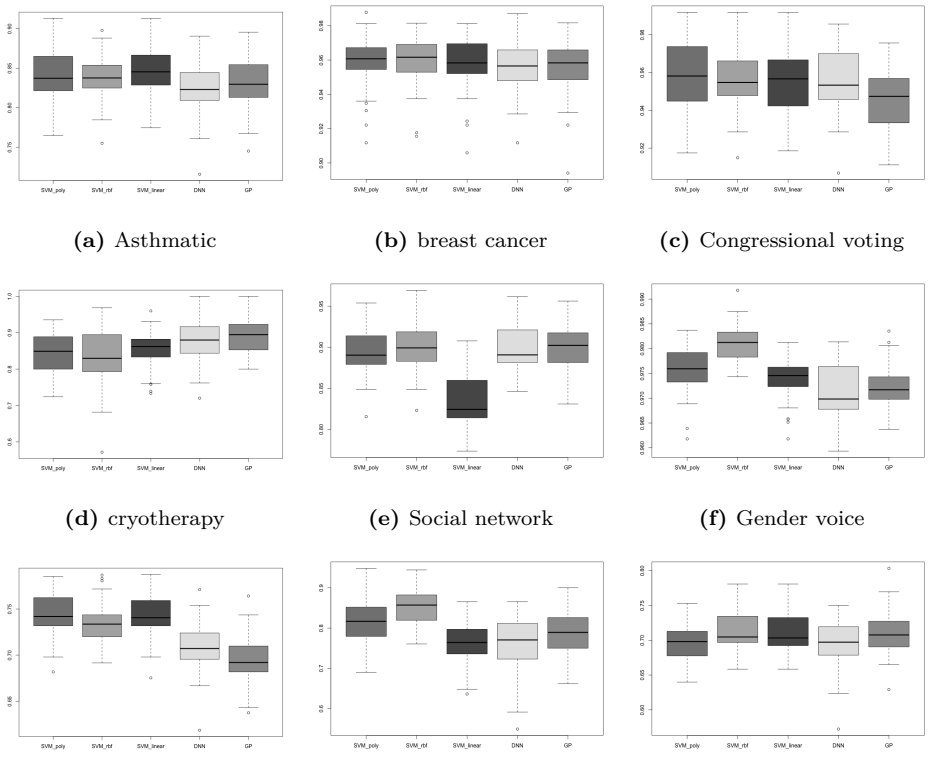
Dataset	SVM-rbf	SVM-poly	SVM-linear	DNN	GP
Asthmatic	0.843	0.840	**0.846**	0.822	0.830
breast cancer	**0.959**	**0.959**	0.958	0.957	0.956
congressional voting	**0.956**	**0.956**	0.955	**0.956**	0.945
cryotherapy	0.844	0.832	0.855	0.876	**0.893**
Social network	0.894	0.900	0.834	0.898	**0.901**
gender voice	0.976	**0.981**	0.974	0.971	0.972
Diabetic	**0.745**	0.734	0.744	0.708	0.695
Sonar	0.816	**0.854**	0.760	0.760	0.788
Indian Liver Patien	0.697	0.714	0.712	0.697	**0.710**

Table 12. Binary Classification minimum accuracy results table

Dataset	SVM-rbf	SVM-poly	SVM-linear	DNN	GP
Asthmatic	0.765	0.755	**0.775**	0.716	0.745
breast cancer	0.912	**0.916**	0.906	0.912	0.894
congressional voting	0.917	0.915	**0.919**	0.907	0.911
cryotherapy	0.724	0.571	0.733	0.720	**0.800**
Social network	0.815	0.823	0.773	**0.846**	0.831
gender voice	0.962	**0.974**	0.962	0.959	0.964
Diabetic	0.682	**0.692**	0.675	0.618	0.637
Sonar	0.690	**0.761**	0.636	0.549	0.662
Indian Liver Patient	0.640	**0.658**	**0.658**	0.572	0.629

Table 13. Multi Classification maximum accuracy results table

Dataset	SVM-rbf	SVM-poly	SVM-linear	DNN	GP
balance scale	**1.00**	0.99	0.97	0.98	0.94
cars	**1.00**	0.99	0.88	0.98	0.91
seeds	0.99	0.99	**1.00**	0.97	0.99



(a) Asthmatic **(b)** breast cancer **(c)** Congressional voting

(d) cryotherapy **(e)** Social network **(f)** Gender voice

(g) Diabetic **(h)** Sonar **(i)** Indian Liver Patient

Figure 6. Comparison boxplot of accuracy(ACC) for 9 binary classification data sets, where x-axis is the name of the machines, y-axis is ACC.

Table 14. Multi Classification median accuracy results table

Dataset	SVM-rbf	SVM-poly	SVM-linear	DNN	GP
balance scale	**1.00**	0.96	0.92	0.96	0.91
cars	**0.98**	**0.98**	0.85	0.96	0.89
seeds	**0.95**	0.93	**0.95**	0.93	0.93

Table 15. Multi Classification mean accuracy results table

Dataset	SVM-rbf	SVM-poly	SVM-linear	DNN	GP
balance scale	**0.99**	0.96	0.92	0.96	0.91
cars	**0.98**	**0.98**	0.85	0.96	0.89
seeds	**0.94**	0.92	**0.94**	0.92	0.92

Table 16. Multi Classification minimum accuracy results table

Dataset	SVM-rbf	SVM-poly	SVM-linear	DNN	GP
balance scale	**0.95**	0.91	0.88	0.90	0.85
cars	**0.96**	**0.97**	0.82	0.94	0.85
seeds	0.80	**0.84**	0.82	0.83	0.82

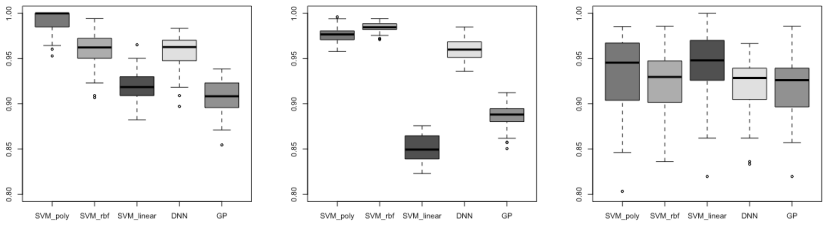


Figure 7. Multi Classification accuracy comparison boxplot, where x-axis is the name of the machines, y-axis is ACC

richness of the data. In this section, we carry out an empirical study of the richness of the data ($\kappa = n/p$) affecting the performance.

The MNIST database mentioned earlier is a large database of handwritten digits from USPS. It has been commonly used for image recognition processing, and training and testing in the field of machine learning. Figure 8 shows 16 sample digits of MNIST data. We randomly take samples from the MNIST classification data set; we use 112, 300, 784, and 1568 samples as training data (Table 17), and 100 samples as test data. We run each machine 50 times for each sample data and then compare the performance on the test data.

Table 17. MNIST datasets

SN	partition	n	p	Number of classes	$\kappa = n/p$
1	MNIST-112	112	784	10	1/7
2	MNIST-300	300	784	10	3/7
3	MNIST-784	784	784	10	1
4	MNIST-1586	1586	784	10	2

We compare the minimum, mean, maximum, and medium of the accuracy results. In Figure 10, we could tell SVM-RBF, SVM-poly and SVM-linear are excellent machines for this task when using four different sizes of data. Next, let us look at the DNN's performance. From box plot one to box plot four we notice that, as the sample sizes get larger, the range of DNN's accuracy gets smaller and catches up with SVM kernel machines. GP is falling behind all the time. In Figure 9, the performance of all the algorithms increases with more data. When using 112

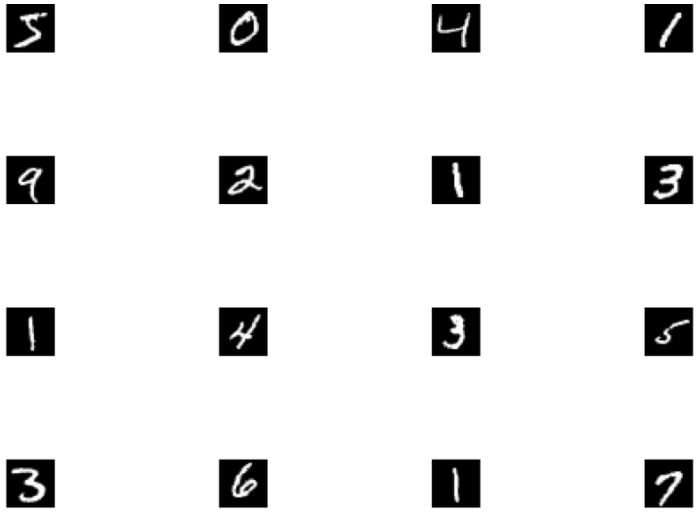


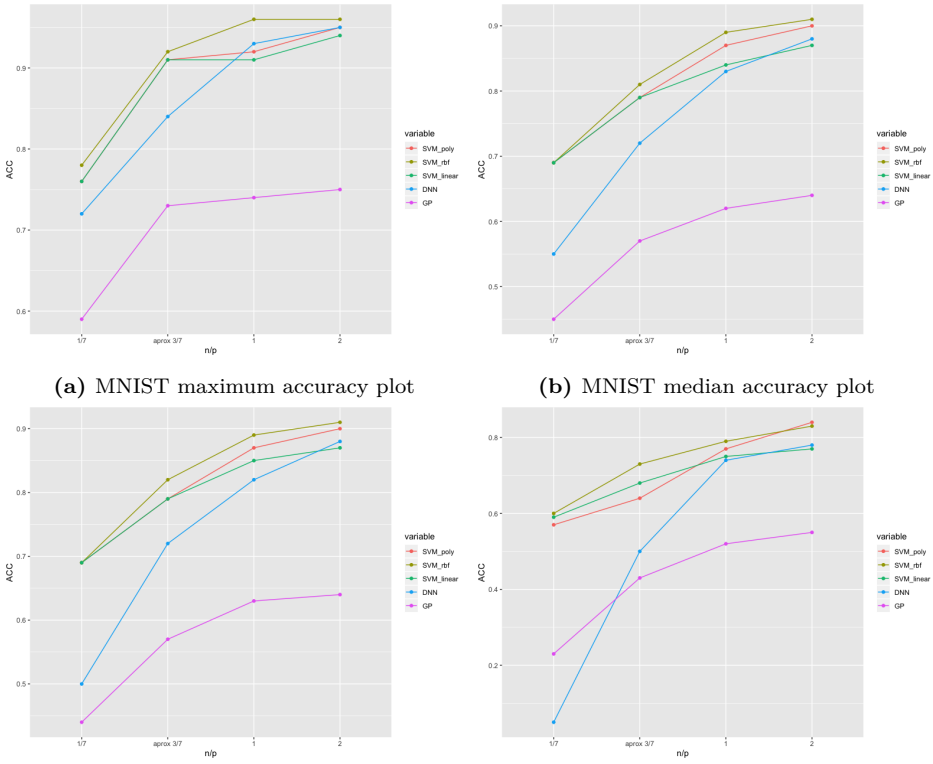
Figure 8. 16 sample digits from MNIST

data points where n/p is 1/7, the medium and maximum all start relatively low. When we increase n/p to approximately 3/7, all the machines have their accuracy increase tremendously, but the SVM kernel machines are still a lot better than DNN. From 3/7 to 1, the accuracy improves for all the machines by a lot, but the DNN model increases more than the other models, and it becomes more compatible with the SVM kernel machines. When increasing n/p to 2, DNN's output is almost as good as SVM-RBF, but the tuning time of the DNN model is a lot longer than that of the SVM-RBF machine. So, based on these results, we can say that DNN is a greedy machine. It needs a rich data set to do the job.

6. Conclusion and discussion

The whole purpose of this paper was to explore the similarities and differences between kernel learning machines and Deep Neural Networks.

Section 4 provided a discussion of the similarities and differences between kernel learning machines and Neural Networks from a conceptual, methodological and theoretical perspectives. It was found that there are some similarities between the two learning paradigms, some of those similarities making the two paradigms almost identical at times. For instance, Gaussian processes turned out to be equivalent to single hidden layer neural networks with an infinite number of nodes in the hidden layer. Radial Basis Function Networks are formulated as Neural Networks with one single hidden layer and later shown to be kernel learning machines, with kernels of the specific type known as radial basis function kernels. In fact, it



(a) MNIST maximum accuracy plot

(b) MNIST median accuracy plot

(c) MNIST mean accuracy plot

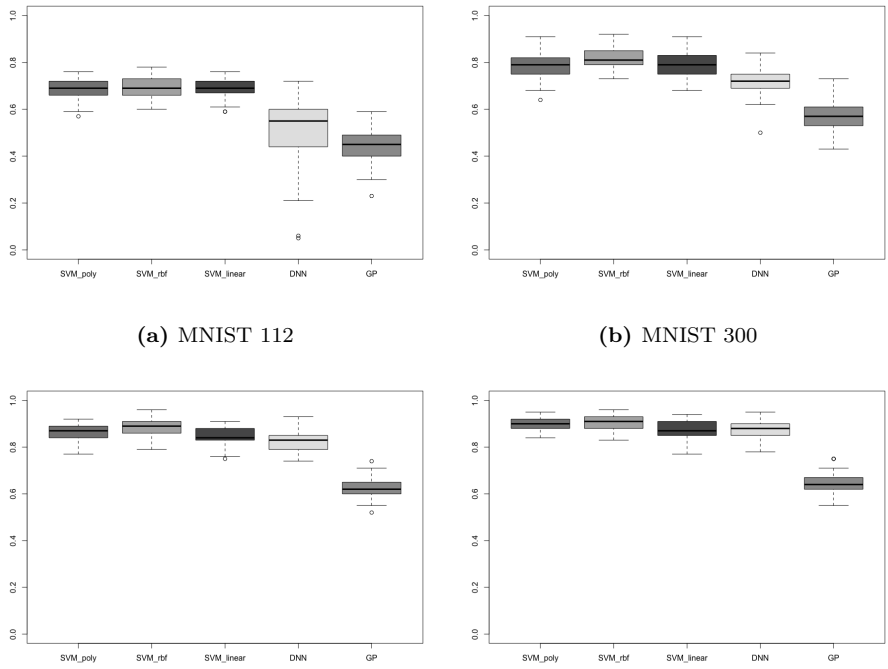
(d) MNIST minimum accuracy plot

Figure 9. MNIST accuracy(ACC) comparison plots, where x-axis is the name of the machines, y-axis is ACC

turns out that, for many versions of the so-called universal approximation theorem, single hidden layer neural networks are indistinguishable from kernel learning machines.

We later created objective experimental comparisons between Deep Neural Networks and Kernel machines in Section 5. We used many simulated and real-world datasets to compare kernel machines and Deep Neural Networks. Based on our experimental results, Kernel machines offer very competitive predictive performances and typically outperform Deep Neural Networks on almost all the data sets explored. Kernel machines also save resources compared to DNN according to the number of parameters and the tuning time. Deep Neural Networks turned out to be inordinately demanding in tuning time because of there complex internal structures, requiring vast amounts of time and computer resources.

$\kappa = n/p$ is one of the essential characteristics in this research, representing the richness of the data. Just as we anticipated, our results clearly show that the performance of DNN strongly depends on the richness of the data. In conclusion, DNN does not appear to be ideal for use if the dataset is small and limited. This is clearly more evidence for neural networks with more than one hidden layer. In the



(a) MNIST 112 **(b)** MNIST 300

(c) MNIST 784 **(d)** MNIST 1568

Figure 10. MNIST accuracy comparison boxplot, where x-axis is the name of the machines, y-axis is ACC

event of data poverty, i.e., when $\kappa < 1$, kernel methods appear to outperform DNN substantially. This alone shows that the claim that DNN should be a panacea used on every conceivable task, is at best an unsubstantiated claim.

Numerous articles based on the "No Free Lunch Theorem" have been published that compare the performance of various machine learning algorithms [40, 49, 50]. The central point of the "No Free Lunch Theorem" is that the class of models for which a given learning technique is the adequate representation is limited. *In other words, no one technique will work well for all problems.* Even though DNN has been a success in a number of very complex real life problems like self driving cars and complex image classification tasks, they also seem not to work well on very many tasks of importance to practitioners from all walks of life. Besides, our computational explorations did show that DNN do typically require relatively powerful computer systems, for some users, which may not be an ideal choice; due to the complex structure of DNN, which is also easier to lead to overfitting problems. DNN also required large amounts of data in order to yield adequate predictive performances comparable to those of kernel learning machines.

Our investigation found that kernel methods may be better suited for typical practical applications. A natural sequel to our investigation would be using more

powerful computers to investigate how well DNN would perform with vast amounts of data.

Acknowledgment. The author Ernest Fokoué wishes to express his deepest gratitude to his Divine Holy Mother, Our Lady of Perpetual Help, for her continued inspiration and empowering support.

References

[1] F. Alfaro-Almagro, M. Jenkinson, N. K Bangerter, J. L. R. Andersson, L. Griffanti, G. Douaud, S. N. Sotiropoulos, S. Jbabdi, M. Hernandez-Fernandez, E. Vallee *et al.*, *Image processing and Quality Control for the first 10,000 brain imaging datasets from UK Biobank*, Neuroimage **166** (2018), 400–424.

[2] A. Barron, *Universal approximation bounds for superpositions of a sigmoidal function*, IEEE Trans. Inf. Theory **39** (1993), 930–945.

[3] A. R. Barron, A. Cohen, W. Dahmen and R. A. DeVore, *Approximation and learning by greedy algorithms*, Ann. Stat. **36** (2008), 64–94.

[4] G. Baudat and F. Anouar, *Kernel-based methods and function approximation*, in: IJCNN'01. International Joint Conference on Neural Networks, Proceedings (Cat. No.01CH37222) **2**, 2001, pp. 1244–1249.

[5] M. Belkin, S. Ma and S. Mandal, *To understand deep learning we need to understand kernel learning*, PMLR **80** (2018), 541–549.

[6] Y. Cho and L. Saul, *Kernel methods for deep learning*, in: Y. Bengio, D. Schuurmans, J. Lafferty, C. Williams and A. Culotta (eds.), Advances in Neural Information Processing Systems **22**, Curran Associates, Inc., 2009, 9 pp.

[7] B. Clarke, E. Fokoué and H. H. Zhang, *Principles and Theory for Data Mining and Machine Learning*, Springer-Verlag, New York, 2009.

[8] C. Cortes and V. Vapnik, *Support-vector networks*, Machine Learning **20** (1995), 273–297.

[9] P. Craven and G. Wahba, *Smoothing noisy data with spline functions: Estimating the correct degree of smoothing by the method of GCV*, Numer. Math. **31** (1979), 377–403.

[10] G. Cybenko, *Continuous Valued Neural Networks with Two Hidden Layers are Sufficient*, Center for Supercomputing Research and Development, University of Illinois at Urbana-Champaign, 1988.

[11] G. Cybenko, *Approximation by superpositions of a sigmoidal function*, Math. Control Signals Syst. **2** (1989), 303–314.

[12] A. G. de G. Matthews, M. Rowland, J. Hron, R. E. Turner and Z. Ghahramani, *Gaussian process behaviour in wide deep neural networks*, Published as a conference paper at ICLR 2018, 2018, 15 pp.

[13] R. Dechter, *Learning while searching in constraint-satisfaction-problems*, in: Proceedings of the Fifth AAAI National Conference on Artificial Intelligence, AAAI'86, AAAI Press, 1986, pp. 178–183.

[14] H. Drucker, C. J. C. Burges, L. Kaufman, A. J. Smola and V. Vapnik, *Support vector regression machines*, in: M. I. Jordan, T. Petsche (eds.), NIPS'96: Proceedings of the 9th International Conference on Neural Information Processing Systems, 1997, MIT Press, Cambridge, MA, 1996, pp. 155–161.

[15] E. Fokoué, *Kernel regression*, in: N. Balakrishnan, T. Colton, B. Everitt, W. Piegorsch, F. Ruggeri and J. L. Teugels (eds.), Wiley StatsRef: Statistics Reference Online, 2021.

[16] E. Fokoué, *On the ubiquity of the bayesian paradigm in statistical machine learning and data science*, Math. Appl. **8** (2019), 189–209.

[17] E. Fokoué, *Model selection for optimal prediction in statistical machine learning*, Notices Am. Math. Soc. **67** (2020), 155–168.

[18] E. Fokoué and B. Brimkov, *The multifaceted impact of statistical methodology and theory in data science*, Math. Appl. **7** (2018), 1–2.

[19] K. Fukushima. *Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position*, Biol. Cybernet. **36** (2004), 193–202.

[20] G. Wahba, *Support Vector Machines, Reproducing Kernel Hilbert spaces, and the Randomized GACV*, Tech. Rep. 984, Department of Statistics, University of Wisconsin, Madison, 1998.

[21] I. Goodfellow, Y. Bengio and A. Courville, *Deep Learning*, Adaptive Computation and Machine Learning, MIT Press, 2016.

[22] G. Guimaraes Olinto and E. Fokoué, *Kernelized cost-sensitive listwise ranking*, Math. Appl. **7** (2018), 31–40.

[23] M. H Hassoun, *Computational Capability of Artificial Neural Networks*, MIT Press, 1995.

[24] T. He, R. Kong, A. J. Holmes, M. Nguyen, M. R. Sabuncu, S. B. Eickhoff, D. Bzdok, J. Feng and B. T. T. Yeo, *Deep neural networks and kernel regression achieve comparable accuracies for functional connectivity prediction of behavior and demographics*, NeuroImage **206** (2020), Article No. 116276, 15 pp.

[25] K. Hornik, *Approximation capabilities of multilayer feedforward networks*, Neural Netw. **4** (1991), 251–257.

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

[27] A. Jacot, F. Gabriel and C. Hongler, *Neural tangent kernel: Convergence and generalization in neural networks*, in: S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett (eds.), Advances in Neural Information Processing Systems **31** (NeurIPS 2018), Curran Associates, Inc., 2018, 10 pp.

[28] A. Jacot, F. Gabriel and C. Hongler, *Neural tangent kernel: Convergence and generalization in neural networks*, ver. 4, arXiv:1806.07572v4, 2020.

[29] A. Karatzoglou, A. Smola, K. Hornik and A. Zeileis, *Kernlab – an S4 package for kernel methods in R*, J. Stat. Softw. **11** (2004), 1–20.

[30] J. Kawahara, C. J. Brown, S. P. Miller, B. G. Booth, V. Chau, R. E. Grunau, J. G. Zwicker and G. Hamarneh, *BrainNetCNN: Convolutional neural networks for brain networks; towards predicting neurodevelopment*, NeuroImage **146** (2017), 1038–1049.

[31] A. Krizhevsky, I. Sutskever and G. E. Hinton, *Imagenet classification with deep convolutional neural networks*, in: F. Pereira, C. J. C. Burges, L. Bottou and K. Q. Weinberger (eds.), Advances in Neural Information Processing Systems **25**, Curran Associates, Inc. (2012), pp. 1097–1105.

[32] V. Kůrková, *Universality and complexity of approximation of multivariable functions by feedforward networks*, in: R. Roy, M. Koeppen, S. Ovaska, T. Furuhashi and F. Hoffmann (eds.), Softcomputing and Industry: Recent Applications, Springer, London, 2002, pp. 13–24.

[33] Y. LeCun, Y. Bengio and G. Hinton, *Deep learning*, Nature **521** (2015), 436–444.

[34] J. Lee, Y. Bahri, R. Novak, S. S. Schoenholz, J. Pennington and J. Sohl-Dickstein, *Deep neural networks as Gaussian processes*, published as a conference paper at ICLR 2018, 2018, 17 pp.

[35] J. Lee, L. Xiao, S. S. Schoenholz, Y. Bahri, R. Novak, J. Sohl-Dickstein and J. Pennington, *Wide neural networks of any depth evolve as linear models under gradient descent*, 33rd Conference on Neural Information Processing Systems (NeurIPS 2019), Vancouver, Canada, 2019, 12 pp.

[36] W. Liu, Z. Wang, X. Liu, N. Zeng, Y. Liu and F. E. Alsaadi, *A survey of deep neural network architectures and their applications*, Neurocomputing **234** (2017), 11–26.

[37] R. W. Murray and E. Fokoué, *Dropout fails to regularize nonparametric learners*, J. Stat. Theory Pract. **15** (2021), Paper No. 23, 20 pp.

[38] V. Nair and G. E. Hinton, *Rectified linear units improve restricted Boltzmann machines*, in: Proceedings of the 27 th International Confer- ence on Machine Learning, Haifa, Israel, 2010, 8 pp.

[39] R. M. Neal, *Priors for infinite networks*, Lecture Notes in Statistics **118**, Springer, New York, 1996, pp. 29–53.

[40] E. Adebayo Ogundepo and E. Fokoué, *An empirical demonstration of the no free lunch theorem*, Math. Appl **8** (2019), 173–188.

[41] J. Park and I. W. Sandberg, *Approximation and radial-basis-function networks*, Neural Comput. **5** (1993), 305–316.

[42] C. E. Rasmussen and C. K. I. Williams, *Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning)*, MIT Press, 2005.
[43] J. Schmidhuber, *Deep learning in neural networks: An overview*, Neural Netw. **61** (2015), 85–117.
[44] B. Scholkopf and A. J. Smola, *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*, Adaptive Computation and Machine Learning, MIT Press, 2018.
[45] D. Snyder, D. Garcia-Romero, D. Povey and S. Khudanpur, *Deep neural network embeddings for text-independent speaker verification*, in: Proc. Interspeech 2017, 2017, pp. 999–1003.
[46] Y. Tian, K. Pei, S. Jana and B. Ray, *Deeptest: Automated testing of deep-neural-network-driven autonomous cars*, in: ICSE '18: Proceedings of the 40th International Conference on Software Engineering, 2018, pp. 303–314.
[47] G. Wahba, *Spline Models for Observational Data*, SIAM CBMS-NSF Regional Conference Series **59**, Philadelphia, 1990.
[48] G. Wahba, *An Introduction to Model Building With Reproducing Kernel Hilbert Spaces*, Tech. Rep. 1020, Department of Statistics, University of Wisconsin, Madison, 2000.
[49] D. H. Wolpert and W. G. Macready, *No free lunch theorems for optimization*, Trans. Evol. Comp. **1** (1997), 67–82.
[50] D. H. Wolpert, W. G. Macready et al, *No Free Lunch Theorems for Search*, Tech. Rep. SFI-TR-95-02-010, Santa Fe Institute, 1995.
[51] Q. Wu, E. Fokoué and D. Kudithipudi, *An ensemble learning approach to the predictive stability of echo state networks*, J. Inform. Math. Sci. **10** (2018), 181–199.
[52] C. Zhang, S. Bengio, M. Hardt, B. Recht and O. Vinyals, *Understanding deep learning (still) requires rethinking generalization*, Commun. ACM **64** (2021), 107–115.

Eddie Pei, Munsell Color Science Laboratory, Rochester Institute of Technology, 85 Lomb Memorial Drive, Rochester, New York 14623
e-mail: ep2667@rit.edu

Ernest Fokoué , School of Mathematical Sciences, Rochester Institute of Technology, 85 Lomb Memorial Drive, Rochester, New York 14623
e-mail: epfeqa@rit.edu