Randomness in Neural Networks: An Overview

Simone Scardapane[*], Dianhui Wang[†]

Article Type:

Overview

Abstract

Neural networks, as powerful tools for data mining and knowledge engineering, can learn from data to build feature-based classifiers and nonlinear predictive models. Training neural networks involves the optimization of non-convex objective functions, and usually the learning process is costly and infeasible for applications associated with data streams. A possible, albeit counter-intuitive alternative is to randomly assign a subset of the networks' weights, so that the resulting optimization task can be formulated as a linear least-squares problem. This methodology can be applied to both feedforward and recurrent networks, and similar techniques can be used to approximate kernel functions. Many experimental results indicate that such randomized models can reach sound performance compared to fully adaptable ones, with a number of favourable benefits, including (i) simplicity of implementation, (ii) faster learning with less intervention from human beings, and (iii) possibility of leveraging over all linear regression and classification algorithms (e.g., ℓ_1 norm minimization for obtaining sparse formulations). All these points make them attractive and valuable to the data mining community, particularly for handling large scale data mining in real-time. However, the literature in the field is extremely vast and fragmented, with many results being reintroduced multiple times under different names. This overview aims at providing a self-contained, uniform introduction to the different ways in which randomization can be applied to the design of neural networks and kernel functions. A clear exposition of the basic framework underlying all these approaches helps to clarify innovative lines of research, open problems and, most importantly, foster the exchanges of well-known results throughout different communities.

[*]Department of Information Engineering, Electronics and Telecommunications (DIET), Sapienza University of Rome, Via Eudossiana 18, 00184 Rome, Italy. Email: simone.scardapane@uniroma1.it

[†]Department of Computer Science and Information Technology, La Trobe University, Melbourne, VIC 3086, Australia. Email: dianhui.wang@latrobe.edu.au

INTRODUCTION

Training neural networks (NNs) is a process involving the optimization of a highly non-convex landscape, where local minima are widespread, and for which the computation of gradients poses fundamental challenges.[1] Alternative kernel-based approaches result instead in well-posed convex problems, one of the key reasons for their success in the last decade.[2] Kernel methods, however, struggle to scale efficiently to large datasets since, among other things, they require the computation of pairwise kernel values over the entire dataset. In addition to this, it is known that optimal testing accuracy achieved by recent deep generations of NNs[3] is only one among many conflicting objectives in data mining applications which, depending on the problem at hand, might include short training time, the possibility of hardware implementation, model complexity, stability of the software libraries, robustness to noise, and so on.[4] This is why, even in our era of deep learning and big data, it remains essential to understand the limits and the potentialities of simpler (i.e., single-layer) neural architectures, allowing a satisfactory level of solution within one-hundredth (or even one-millionth) of the time required by larger, more complex models to be reached, while at the same time possessing strong optimality guarantees in their optimization process, and which can be customized and implemented easily.

One successful approach in this sense is the use of '*randomization*', i.e. the stochastic assignment of a subset of the NNs weights in order to derive a simpler (often linear) optimization problem to solve. This idea has been applied countless times over the years, and it has resulted in three broad families of NN models which we classify as follows: (i) feedforward networks with random weights (RW-FFN),[5] (ii) recurrent neural networks with random weights (i.e. reservoir computing, RC),[6] and (iii) randomized kernel approximations.[7] Although the literature on these three methods is generally kept separate, they all share two fundamental ideas which contribute to their success. First, in all three cases randomization is used to define a (generally data independent) feature map, which transforms the input into a highly-dimensional space where learning is supposed to be simpler. Secondly, the resulting optimization problem is cast as a standard linear least-squares, which is by far the simplest, most studied and scalable learning procedure to date. In a sense, many of

these methods are direct descendants of the seminal work of Cover on the linear separability of patterns,[8] and they are elegantly summarized in the recent quote from Rahimi and Recht that "*randomization is [...] cheaper than optimization*".[9]

Both of the aforementioned aspects can help to explain the remarkable successes obtained by randomized NN models in realistic data mining problems. State-of-the-art libraries for linear regression are able to handle very large feature spaces, where the probability of finding a hyperplane with good generalization capabilities can be extremely high. In addition to this, drawing inspiration from the vast literature on linear regression (which goes as far back as Gauss himself[10]), it is possible to customize the training procedure in a straightforward way with a vast number of dedicated algorithms. As an example, sparsity on the adaptable parameters can be obtained immediately by incorporating a sparse penalty and solving the resulting linear problem in all families.[11] At the same time, the literature on these topics is vast and fragmented, so that equivalent ideas are reintroduced time and again, and it becomes difficult to appreciate the fundamental unity (both theoretical and practical) underlying all these methods. Going back to our previous example, sparse linear regression has been derived independently in all three areas, sometimes more than once in each case.[12–14]

Our aim in this overview is to provide a concise, self-contained, and uniform introduction to these three families of methods. Apart from providing a single entry point into a growing (and decades-spanning) body of literature, clearly stating the basic framework underlying them can help to clarify new lines of research and, more importantly, to foster the exchanges of well-known results throughout different communities. Two broad themes emerge from our treatment, which are interesting to briefly discuss here. First and foremost, there is a clear indication that random weight assignment is not just a 'cheap trick' to solve an otherwise difficult problem. On the contrary, the algorithms discussed here have continuously achieved remarkable accuracy, sometimes comparable to deep architectures,[15] in a large number of applications, ranging from short-term forecasting to image recognition, biomedical classification, and many others. This apparently counter-intuitive insight warrants more attention, as its reasons can point to a deeper understanding of the working behavior of NNs themselves. As discussed in the work of Saxe *et al.*,[16] for example, it appears that "*a sizeable component of a system's performance can come from the intrinsic properties of the*

architecture", so that a good architecture with a subset of weights assigned randomly can easily outperform a poorer architecture with finely-tuned weights. This also explains why research in this field, far from withering away, has continued to grow relentlessly as available computational power increased.

The second important aspect highlighted here is that randomized methods can lead the way to some intriguing theoretical analyses on their behavior, in terms of accuracy, dynamics and memory. As an example, RW-FFNs are shown to have error bounds comparable to that of networks with completely adaptable parameters. One additional aim of our overview is to summarize some of the most fundamental properties evidenced by these studies, along with the open problems and challenges that need to be resolved.

Structure of this overview

Following our categorization of randomized NN models, this overview is organized in three parts. The first section, '*Feedforward networks with random weights*', introduces the simplest randomized NN model, the RW-FFN. The content of this section is the foundation for the remaining of the paper: in fact, all training methods and most approximation properties that we describe are common to *all* families in the overview. The following sections describe the two remaining families of models, namely random features for kernel methods and the RC framework. In both cases, we first show how the learning problem is equivalent to that of RW-FNN, before focusing on some aspects that are specific to each model, such as the theoretical analysis of the reservoir in RC networks. Finally, the last section draws some general conclusions from our overview.

Feedforward networks with random weights

Network architecture

The basic model is defined for a generic input $\mathbf{x} \in \mathbb{R}^d$ as the linear combination of B nonlinear transformations (see Fig. 1):

$$f(\mathbf{x}) = \sum_{m=1}^{B} \beta_m h_m(\mathbf{x}; \mathbf{w}_m) = \boldsymbol{\beta}^T \mathbf{h}(\mathbf{x}; \mathbf{w}_1, \ldots, \mathbf{w}_B)\,, \tag{1}$$

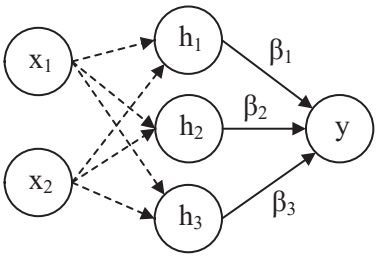

Figure 1: A RW-FNN architecture with two inputs, three hidden functions, and one output. Fixed connections are shown as dashed lines, whilst trainable connections as fixed lines.

where the mth transformation is parameterized by the vector $\mathbf{w}_m$. In (1), the weights $\mathbf{w}_m$ are drawn randomly from a predefined probability distribution, while the linear coefficients $\boldsymbol{\beta}$ are adapted based on some training data. Although the model is derived for a single output only, everything that follows can be extended straightforwardly to the case of multiple outputs. For readability, in the following we drop the dependence of h_m with respect to $\mathbf{w}_m$.

There are two basic families of methods falling under this scheme, depending on the type of hidden functions $h_m(\cdot)$. In the *additive* case, each function applies a predefined nonlinearity on a random linear combination of its input vector:

$$h_m(\mathbf{x}) = g(\mathbf{a}_m^T \mathbf{x} + b_m), \tag{2}$$

with $\mathbf{w}_m = \left[\mathbf{a}_m^T \; b_m\right]^T$. A typical choice for $g(\cdot)$ in this case is the sigmoid function:

$$h_m(\mathbf{x}) = \frac{1}{1 + \exp\left\{-\left(\mathbf{a}_m^T \mathbf{x} + b_m\right)\right\}}. \tag{3}$$

If we stack row-wise all vectors $\mathbf{a}_m$ and scalars b_m into a matrix $\mathbf{A}$ and vector $\mathbf{b}$, respectively, the first layer can be written compactly as $g(\mathbf{A}\mathbf{x} + \mathbf{b})$, where $g(\cdot)$ now operates elementwise. The linear projection $\mathbf{A}\mathbf{x}$ is loosely connected to the well-known method of random projections (RP),[17,18] a fact that might engender confusion. In RP, the matrix $\mathbf{A}$ is also randomly generated, but the aim is to reduce the dimensionality of the data while at the same time approximately preserving the Euclidean distances among the projected points. Since the aims are different, there are two broad differences with respect to the RW-FNN that have to be kept in mind: (i) in order to have sufficiently expressive power, in our case B is typically much larger than the number of input dimensions (even by one order of magnitude) and,

(ii) the linear transformation $\mathbf{A}$ is not required to preserve distances.[17] In fact, it is standard practice to randomly sample its values from a predefined distribution (e.g., the uniform distribution in $[-1, +1]$), which might not respect the theory of RP.

In the second family of methods, each function is chosen instead as a radial basis function (RBF), typically of Gaussian shape:

$$h_m(\mathbf{x}) = \exp\left\{-\alpha_m \left\|\mathbf{x} - \mathbf{c}_m\right\|_2^2\right\}, \tag{4}$$

with $\mathbf{w}_m = \left[\mathbf{c}_m^T \; \alpha_m\right]^T$, and $\|\cdot\|_2$ is the standard Euclidean norm. In this case, the network can be seen as an RBF network with randomly chosen centers $\mathbf{c}_m$ and scaling factors α_m, corresponding to a superposition of B Gaussian distributions centered at random places in the input space.[19] Alternative, less common choices for $g(\cdot)$ are also possible, including multiplicative univariate functions,[20] convolutive filters with random weights (for 2D inputs),[21] and others. Random features for approximating kernels, discussed in the next section, also fit the basic model in (1). However, they derive from an intermediate kernel representation, and for this reason we discuss them separately.

Training the network

Given a model as in (1), we need efficient algorithms to adapt its parameters. Again, we stress that the content of this section apply directly to the training of *all* models considered in the next sections. Suppose we are provided with a set of N samples of the desired function to be approximated, that is $S = \{\mathbf{x}_i, y_i\}$, $i = 1, 2, \ldots, N$. Denoted by $\mathbf{H}$ the *hidden* matrix:

$$\mathbf{H} = \begin{pmatrix} h_1(\mathbf{x}_1) & \cdots & h_B(\mathbf{x}_1) \\ \vdots & \ddots & \vdots \\ h_1(\mathbf{x}_N) & \cdots & h_B(\mathbf{x}_N) \end{pmatrix}, \tag{5}$$

Choosing the optimal $\boldsymbol{\beta}$ can obviously be formulated as a standard regularized least-squares problem:

$$\boldsymbol{\beta}^* = \underset{\boldsymbol{\beta} \in \mathbb{R}^B}{\arg\min} \left\{ \frac{1}{2} \left\|\mathbf{H}\boldsymbol{\beta} - \mathbf{y}\right\|_2^2 + \frac{\lambda}{2} \left\|\boldsymbol{\beta}\right\|_2^2 \right\}, \tag{6}$$

where $\mathbf{y}$ is a column vector containing all the desired outputs and the user-defined scalar λ weights the regularization term. The problem in (6) is strictly convex, so its solution can be

found by setting its gradient to 0, that is:

$$\boldsymbol{\beta}^* = \left(\mathbf{H}^T\mathbf{H} + \lambda\mathbf{I}\right)^{-1}\mathbf{H}^T\mathbf{y}\,, \tag{7}$$

where $\mathbf{I}$ is the identity matrix of suitable dimensionality. The fact that the training of the model (more generally, of all the models described in this overview) can be performed in a single analytical step that is implemented efficiently in most linear algebra libraries is the major advantage of these methods. The computational complexity of least-squares in this case is mostly influenced by the $B \times B$ matrix inversion in (7). In cases where $N \ll B$, standard algebra also allows for an alternative formulation, where the matrix to be inverted is reduced to dimensionality $N \times N$:

$$\boldsymbol{\beta}^* = \mathbf{H}^T\left(\mathbf{H}\mathbf{H}^T + \lambda\mathbf{I}\right)^{-1}\mathbf{y}\,. \tag{8}$$

For large-scale problems, parallel versions of (6) exist in most parallel computing frameworks. For example, matrix-matrix and matrix-vector products in (6) can be computed in sub-groups using a MapReduce environment,[22] and summed up before inversion in a reducer node. A streaming parallel implementation is also available on the popular Apache Spark library,[1] and matrix decompositions for computing the Moore-Penrose inverse $\left(\mathbf{H}^T\mathbf{H} + \lambda\mathbf{I}\right)^{-1}\mathbf{H}^T$ can be parallelized using GPU or hybrid CPU/GPU clusters.[23]

Additionally, the problem can be addressed using convex optimization routines such as the conjugate gradient,[5] or primal-dual strategies,[24] which in the case of quadratic problems such as (6) possess very fast convergence guarantees. The problem can also be solved efficiently in the case of streaming data (with possibly time-varying characteristics) by the use of the well-known recursive least-squares algorithm.[25] For binary classification problems (i.e., when the output can only take binary values), the simplest approach is to combine the least-squares criterion in (6) with subsequent binarization of the RW-FNN outputs. More efficient loss functions (e.g. hinge loss) are possible, but do not admit a closed-form solution anymore; a review on recent progress for large-scale linear classification is given by Yuan *et al.*[26]

While this is enough to implement a working RW-FNN in most situations, it only scratches the surface of all possible modifications for its training algorithm, which can leverage in a

[1] `http://spark.apache.org/docs/latest/mllib-linear-methods.html#regression`

straightforward fashion over all available literature on linear regression/classification. Some of these have also been explicitly considered in the case of RW-FNN. As a representative example, ℓ_1 minimization can be used to achieve *sparse* output layers:[12]

$$\boldsymbol{\beta}^* = \arg\min_{\boldsymbol{\beta}\in\mathbb{R}^B} \left\{ \frac{1}{2} \left\| \mathbf{H}\boldsymbol{\beta} - \mathbf{y} \right\|_2^2 + \lambda \left\| \boldsymbol{\beta} \right\|_1 \right\}, \tag{9}$$

where $\|\boldsymbol{\beta}\|_1 = \sum_{m=1}^{B} |\beta_m|$. This is the much celebrated LASSO problem, for which extremely efficient optimization procedures are available.[11] Other possible variations are the absolute penalization of errors for the robust handling of noise and outliers,[27], ensembling for increasing accuracy and lowering variance,[28] and so on. A complete overview of least-squares theory goes outside the scope of this paper, however, and we refer the reader to standard textbooks on the topic.[29]

Historical overview on RW-FNN models

The basic idea represented by (1) is simple, yet it provides a very efficient nonlinear model. For this reason, similar ideas have been proposed many times, in different forms, over the last several decades. Here, we provide a brief historical overview of some representative works.

The perceptron

An historical predecessor to RW-FNNs can be found in the perceptron, one of the first adaptable connectionist models explored by Frank Rosenblatt from 1957 onwards.[30] In the perceptron, a sparse, random layer of weights connects the sensory perceptions to an adaptable threshold layer (see for example Fig. 13 in Widrow and Lehr[31]). This architecture was designed to work only for binary/ternary activations and, most importantly, its original training rule does not converge in the case of patterns which are not linearly separable: "*While it will eventually visit an optimal set of weights, it will not converge to any set of weights. Even worse, the algorithm can go from an optimal set of weights to a worst-possible set in one iteration, regardless of how many iterations have been taken previously.*"[32] This aspect was at the base of the fundamental historical controversy surrounding this model.[33] Later modifications to the perceptron came closer to the RW-FNN as described previous-

ly, most notably the variation of the 'pocket algorithm' proposed by Gallant and Smith in 1987.[32] However, they failed to gain additional popularity at the time.

Random vector functional-link networks

In the 1990s, RW-FNNs were introduced in their current form by a research group led by Y.-H. Pao, under the name random vector functional-link (RVFL) networks.[5,34,35] The only difference with respect to our treatment is the straightforward inclusion of direct links from the input to the output layer, which does not change the successive mathematical derivation. Apart from proving their universal approximation property (see next section), they showed their usefulness on a number of real-world problems, including in particular handwritten script recognition.[36] To this end, they also investigated how RVFL networks can be customized to offer a practical tool for density estimation.[37] RVFL networks have gained increasing popularity in the last few years, and numerous additional applications were explored, which are briefly summarized in Section 1 of Martínez-Villena *et al.*[38] An extensive evaluation of different RVFL variants is given instead in Zhang and Suganthan,[39] showing, in particular, significant improvements by the inclusion of the input/output links. RVFL networks have also been explored in semi-supervised scenarios, wherein only a part of the training data is labeled,[40] and more recently in a multi-layered configuration using a variation of the autoencoder network.[41]

RBF networks with random centers

In their RBF form (4), RW-FNNs were already envisioned in the seminal work of Lowe and Broomhead on RBF networks,[19] in which the random sampling of centers from the training data is a good alternative to more sophisticated clustering procedures. Lowe and Broomhead, in a footnote, hinted at the possibility of choosing RBF centers that do not correspond to any of the training points. This strategy was followed by a few authors, including Looney,[42] but never became mainstream because of the possibility that random centers might not represent the actual distribution of points in the training dataset at hand.

Additional considerations

In a 1992 conference publication, Schmidt *et al.* presented a network which closely resembled the additive version of the RW-FNN.[43] The network, however, had a completely different significance. Far from being a practical learning algorithm, it was used as a paradoxical criterion to explore some bizarre aspects of NNs, namely the possibility of obtaining high accuracies even when the number of parameters far exceeded the available training data, a result which goes against intuitions from learning theory. In this sense, their conclusion that "*the parameters found in the hidden layer do not add very much functionality to the classifier*" was not a justification for randomization, but must be read in a wider line of skepticism toward NNs themselves, whose application is sometimes viewed as a black-box approach without a 'real' understanding of the problem.[44] The significance of the work of Pao and colleagues can be found exactly in their acceptance that the RW-FNN is not only efficient but also *practical*, and this is clear insofar as the RVFL is presented as a "*computational approach compatible with our goal of devising a general-purpose artificial neural-net computer*".[34] This comment is echoed in a discussion by Widrow,[45] who replied to some general criticisms after having introduced an algorithm very similar to the RW-FNN termed "NoProp algorithm"[46]: "*we [...] have independently discovered that it is not necessary to train the hidden layers of a multi-layer neural network. Training the output layer will be sufficient for many applications.*" It should be pointed out that some statements on the full-rank property of the hidden output matrix in this work are not entirely justified, and some further research is to be expected.

Before concluding this section, we remark that some interesting work has also been done to justify the RW-FNN approach from a biological perspective, as shown in the following quote by Tapson and van Schaik: "*we are starting to see some evidence from neurophysiology that structures embodying the [random weights] principle may exist in the brain. For example, recent work [...] shows that complex rule-based tasks require both sensory stimuli and internal representation of states; and that a significant number of random connections placed between input sources and a hidden interlayer, and random recurrent connections between interlayer neurons, are necessary for optimal performance. [...] these interlayer*

neurons [are] equivalent to increasing the dimensionality of the state representation."[47] We will return on this point later on.

Universal approximation properties

The basic theoretical result on RW-FNNs was proved in 1995 by Igelnik and Pao,[35] with later corrections made by Li and Chow.[48] They showed that, for sufficiently smooth functions, and under a proper choice of $h_m(\cdot)$ and the range from which parameters are extracted, the network possesses a universal approximation capability. Particularly, the order of approximation error is $\mathcal{O}(C/\sqrt{B})$, where the constant C is independent of B. Thus, a desired level of accuracy can, in principle, be reached by increasing the number of hidden nodes. The result is extremely significant because this error bound is, in fact, comparable to that of a network with fully adaptable connections. It also goes against the worst-case bound $\mathcal{O}(d^{-1}/B^{1/d})$ presented in the seminal work by Barron for a generic linear combination of fixed functions.[49] This apparent discrepancy is explored by Gorban *et al.*,[50] who underline the probabilistic nature of the result by Igelnik and Pao: "*[...] there is always a non-zero probability of an unlucky draw from the probability distribution which will require re-initialization at some stage of approximation. Furthermore, [...] this rapid convergence of order* $1/\sqrt{B}$ *is assured only up to a given and fixed tolerance.*" From this, they draw the following warning: "*These features of randomized approximators should thus be considered with special care in applications.*" In this sense, how to properly choose a range of parameters and a sampling distribution remains an open problem in the literature, so that the advantages given by RW-FNN approaches are paid by the possibility of sampling a network with extremely poor accuracy. Similar statements are made in a recent overview by Principe and Chen[51]: "*[random-weights models] still suffer from design choices, translated in free parameters, which are difficult to set optimally with the current mathematical framework, so practically they involve many trials and cross validation to find a good projection space, on top of the selection of the number of hidden [processing elements] and the nonlinear functions.*". It should be pointed out that the theoretical work established by Igelnik and Pao[35] does not address the learning algorithm or implementation issues. From algorithmic perspective, Li and Wang[52] investigated the universal approximation property for RVFL networks and reported a result on the infeasibility

of such a learner model, if it is built incrementally with random input weights and biases extracted from a fixed scope, and its convergence rate satisfies some certain condition. These limitations should always be kept in mind.

Recently, more refined theoretical bounds have been obtained by a number of authors in the kernel approximation literature, most notably Rahimi and Recht in 2008-2009 (who ironically referred to the model as 'sums of random kitchen sinks'),[9,53] and Rudi *et al.* in 2016.[54] The latter work also puts forth the idea that randomization can serve as a form of 'computational regularization' during the training process. The theory behind these works is the subject of the next section.

Random features for kernel approximation

Network architecture

The second family of methods that we investigate is a special case of the model in (1), but its understanding is built on the notion of kernel functions. Due to this, we first provide a brief introduction to kernel methods, before analyzing their drawbacks as motivation for introducing random features for their approximation.

A brief primer on kernel methods

A common alternative to the stochastic assignment of weights in (1) is to consider a *deterministic* B-dimensional feature mapping $\mathbf{h}(\cdot)$, e.g. by the use of polynomial functions of the input. In some cases, however, this strategy requires the use of an intractable number of bases that can grow cubically or exponentially in the number of inputs. Kernel functions, as a way to circumvent this problem, were popularized by Cortes and Vapnik,[55] and became the foundation for a wide array of algorithms in machine learning.[2] Informally, it can be shown that for a large number of mappings $\mathbf{h}(\cdot)$, there exists a function $\mathcal{K} : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ such that:

$$\mathcal{K}(\mathbf{x_1}, \mathbf{x}_2) = \mathbf{h}(\mathbf{x}_1)^T \mathbf{h}(\mathbf{x}_2)\,. \tag{10}$$

The function $\mathcal{K}(\cdot,\cdot)$ is called a kernel, and it allows for an efficient evaluation of the dot product in the transformed space without the need for computing the mapping itself. One basic result, called the representer's theorem, shows that the solution to a large number of regularized supervised learning problems when using kernels can be expressed as:

$$f^*(\mathbf{x}) = \sum_{i=1}^{N} \alpha_i \mathcal{K}(\mathbf{x}, \mathbf{x}_i) \tag{11}$$

so that an optimization over a possibly infinite-dimensional space of functions is reduced to a finite-dimensional problem over the coefficients $\{\alpha_i\}_{i=1}^N$. Similarly, most problems in this case need to work with the so-called *kernel matrix* $\mathbf{K} \in \mathbb{R}^{N\times N}$ defined as:

$$K_{ij} := \mathcal{K}(\mathbf{x}_i, \mathbf{x}_j)\,. \tag{12}$$

As an example, the optimal solution $\boldsymbol{\alpha}^* = [\alpha_1, \ldots, \alpha_N]^T$ to the kernel-based version of (6) (called kernel ridge regression) is written as:

$$\boldsymbol{\alpha}^* = (\mathbf{K} + \lambda \mathbf{I})^{-1}\,\mathbf{y}\,. \tag{13}$$

Drawbacks of kernel methods for large-scale problems

The previous discussion, albeit short, already clarifies most of the drawbacks encountered during the application of any kernel method to a large-scale application:

1. First and foremost, the computation of $\mathbf{K}$ scales quadratically with N, which might be prohibitive both in time and in memory, even when using sophisticated linear algebra libraries.

2. Secondly, the computational complexity of solving the optimization problems involved in computing the optimal weight vector $\boldsymbol{\alpha}^*$ are between quadratic and cubic in N, even for methods which are provably sparse in $\boldsymbol{\alpha}^*$ (such as support vector machines).

3. A third, less visible problem appears in online applications, wherein the number of terms in (11) grows unbounded. This problem is particularly daunting in adaptive filtering applications that exhibit stringent real-time requirements.[56]

4. Finally, (11) is defined in terms of the original data, thus being problematic also for applications with privacy concerns, which are common in big data scenarios.[57]

Kernel approximation techniques

Over the past years, a large number of approximation techniques have been developed to overcome the previously mentioned drawbacks, in order to make kernel methods affordable even in large-scale situations. These methods use randomization to approximate several aspects of the training process, for example by appropriately sampling elements of the kernel matrix,[58] computing a low-rank approximation of $\mathbf{K}$,[59] or designing a stochastic approximation to the kernel function $\mathcal{K}(\cdot,\cdot)$ itself.[7]

The most well-known example of the second class is the Nyström method, originally proposed by Williams and Seeger,[60] and refined by Drineas and Mahoney.[59] The Nyström method is a two-step process, which can be simplified as follows: (i) we first build a matrix $\mathbf{G}$ by randomly sampling $M \ll N$ columns of $\mathbf{K}$ according to a predefined strategy, and scaling them appropriately; (ii) then, we compute a low-rank approximation of $\mathbf{K}$ by $\hat{\mathbf{K}} = \mathbf{G}\mathbf{K}_k^+\mathbf{G}^T$, where $\mathbf{K}_k^+$ is the best k-rank approximation of the square part of $\mathbf{K}$ corresponding to the indexes sampled in the step before (also scaled appropriately, see Drineas and Mahoney[59] for more details). While the Nyström method makes use of a randomization step, it does not result in a proper linear problem, and for this reason we do not discuss it further here (although we will use it for comparison later on).

The third class of algorithms, instead, is interested in finding a good, low-dimensional approximation of the kernel function, so as to obtain a feature mapping in the form (1) which is cheap to compute in real-time. The idea was originally introduced by Rahimi and Recht for a particular class of kernels called shift-invariant,[7] and later generalized by a number of authors for other classes, including group-invariant kernels,[61] γ-homogeneous kernels,[62] and dot-product kernels,[63] among others. The idea can be described in a general form as follows. Denote by v a random variable with probability density $P(v)$, and suppose the kernel can be expressed as an expectation form:

$$\mathcal{K}(\mathbf{x}_1, \mathbf{x}_2) = \mathrm{E}_{v\sim P(v)}\left[\phi_v(\mathbf{x}_1)\phi_v(\mathbf{x}_2)\right] , \tag{14}$$

where $\phi_v(\cdot) : \mathbb{R}^d \to \mathbb{R}$ are a family of functions indexed by the random variable v, and $v \sim P(v)$ denotes that the expectation is taken with respect to $P(v)$. Then, we can construct a low-dimensional approximation $\widetilde{\mathcal{K}}$ of $\mathcal{K}$ with a Monte Carlo approach, by randomly sampling

B values $\{v_1, \ldots, v_B\}$ from $P(v)$ to approximate (14):

$$\widetilde{K}(\mathbf{x}_1, \mathbf{x}_2) = \frac{1}{B} \sum_{m=1}^{B} \phi_{v_m}(\mathbf{x}_1) \phi_{v_m}(\mathbf{x}_2) \,. \tag{15}$$

The feature mapping $\widetilde{\boldsymbol{\phi}}(\cdot) = [\phi_{v_1}(\cdot), \ldots, \phi_{v_B}(\cdot)]$ can now be computed directly. Due to this, all training methods described in the previous section can be applied for optimizing these models, and their universal approximation properties can be analyzed in a similar fashion. In other words, we can apply linear models to approximate nonlinear models that use the original kernel $\mathcal{K}(\cdot, \cdot)$, which in turn indirectly define a nonlinear mapping over a possibly infinite-dimensional space $\mathbf{h}(\cdot)$. Standard arguments from probability theory guarantee an exponentially fast decrease of the error of (15) with respect to (14) as B increases.[7] It is important not to mistake the fixed feature mapping $\mathbf{h}(\cdot)$, which uniquely identifies the kernel function, with the randomized space $\widetilde{\boldsymbol{\phi}}(\cdot)$, which depends on the sampled vectors $\mathbf{v}_1, \ldots, \mathbf{v}_B$. This relation is schematically shown in Fig. 2.

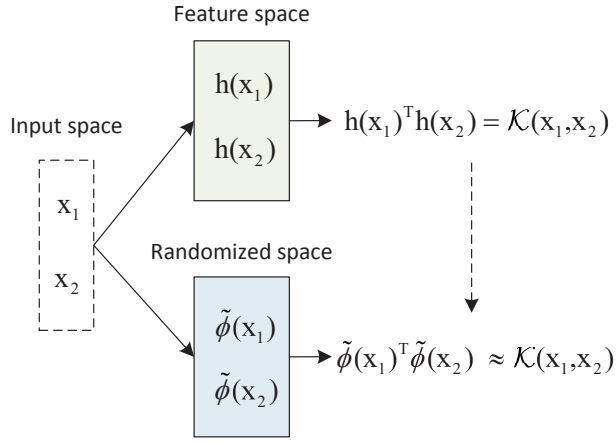

Figure 2: Schematic representation of the kernel approximation process. The original, *fixed* feature mapping is shown in light green, while the randomized space to approximate the kernel evaluation is shown in light blue.

A concrete example: random Fourier features

Random Fourier features (RFF), as originally introduced by Rahimi and Recht,[7] are the most common implementation of random features for kernel approximation. Here, we briefly introduce them and their extensions. RFFs are defined for any shift-invariant kernel, i.e. a

kernel which depends on its inputs $\mathbf{x}_1$ and $\mathbf{x}_2$ only by their difference $\mathbf{x}_1 - \mathbf{x}_2$. In this case, it can be shown that the kernel admits a representation similar to (14) in a Fourier basis with complex functions:

$$\mathcal{K}(\mathbf{x}_1 - \mathbf{x}_2) = \mathrm{E}_{\mathbf{v}\sim P(\mathbf{v})}\left[e^{j\mathbf{v}^T\mathbf{x}_1}e^{-j\mathbf{v}^T\mathbf{x}_2}\right], \tag{16}$$

where j is the imaginary number, and the existence of $P(\mathbf{v}), \mathbf{v} \in \mathbb{R}^d$ in closed form is guaranteed by Bochner's theorem from armonic theory. Substituting the exponential with sines and cosines allows to obtain real-valued features, so that the final feature vector for RFFs is written as:

$$\widetilde{\phi}(\mathbf{x}) = \frac{1}{\sqrt{B}}\left[\cos(\mathbf{v}_1^T\mathbf{x}), \sin(\mathbf{v}_1^T\mathbf{x}), \ldots, \cos(\mathbf{v}_B^T\mathbf{x}), \sin(\mathbf{v}_B^T\mathbf{x})\right]. \tag{17}$$

As a concrete example, consider the well-known Gaussian kernel, which is by far one of the most widely used kernels in practice thanks to its approximation properties. The Gaussian kernel is defined for a parameter $\gamma \in \mathbb{R}$ similarly to α_m in (4):

$$\mathcal{K}(\mathbf{x}_1, \mathbf{x}_2) = \exp\left\{-\frac{1}{\gamma^2}\left\|\mathbf{x}_1 - \mathbf{x}_2\right\|_2^2\right\}. \tag{18}$$

In this case, the probability density $P(\mathbf{v})$ is simply a Gaussian with mean 0 and covariance inversely proportional to γ given by $\gamma^{-1}\mathbf{I}$. RFFs have been studied extensively and have also given rise to a large number of approximation results which can be translated to the RW-FNN case, as reported in the previous section. Of the methods we review here, they are the only one possessing robust, stable software implementations, such as the one in the Python's scikit-learn library,[2] and they have been applied successfully to problems outside supervised learning, such as clustering.[64]

Before concluding this section, we review some of the most recent, promising lines of research devoted to random features:

- Vedaldi and Zisserman[62] investigated how RFF techniques can be extended to 'generalized' Gaussian kernels, wherein the ℓ_2 norm in (18) is substituted with a generic distance metric $D(\mathbf{x}_1, \mathbf{x}_2)$. If the distance metric has an additive form, i.e. it can be decomposed feature-wise, an efficient random approximation to the generalized kernel

[2] `http://scikit-learn.org/stable/modules/kernel_approximation.html`

can be obtained by combining random features for $D(\cdot, \cdot)$ (using the algorithm presented by the authors) with RFFs features over the kernel. This allows extremely good results to be obtained, particularly in computer vision problems.

- In order to further speedup computations, Le *et al.*[65] presented the *fastfood* algorithm, which allows RFFs for the Gaussian kernel to be computed in $\mathcal{O}(B \log(d))$ time instead of $\mathcal{O}(Bd)$, by the use of specially designed matrices. Their work is generalized in the framework presented by Yang *et al.*[66] Along the same lines, a state-of-the-art solver for kernel methods on large-scale scenarios is presented in Dai *et al.*,[67] by combining RFFs with a stochastic evaluation of the overall gradient.

- Instead of first choosing a kernel and then deriving an approximation, recently Sinha and Duchi showed that we can work with a generic random feature representation to learn a suitable kernel function for a given task.[68]

- Finally, RFFs have also been studied for solving the last two classes of problems previously detailed, namely the unbounded growth of terms in the kernel expansion,[69] and the possible concerns in privacy-critical settings.[70] Refined error bounds in the former case have been obtained by Lin *et al.*[71]

Drawbacks and research directions

Just like RW-FNNs, the basic problem of random features for kernel approximation is that their sampling strategy does not incorporate information from the training set. Yang *et al.*[72] summarize the problem as follows: "*[...] the basis functions used by random Fourier features are sampled from a Gaussian distribution that is independent from the training examples. In contrast, the basis functions used by the Nyström method are sampled from the training examples and are therefore data dependent.*" Basically, the results in terms of classification accuracy in this case will depend on the eigenstructure of the kernel matrix: "*In the case of large eigengap, i.e., the first few eigenvalues of the full kernel matrix are much larger than the remaining eigenvalues, the classification performance is mostly determined by the top eigenvectors. Since the Nyström method uses a data dependent sampling method, it is able*

to discover the subspace spanned by the top eigenvectors using a small number of samples. In contrast, since random Fourier features are drawn from a distribution independent from training data, it may require a large number of samples before it can discover this subspace."

One possibility to counter this problem is to draw a large number B of samples, and then linearly down-project the resulting feature mapping to a low-dimensional space via the use of a random projection.[73] The error bound in this case is investigated by Hamid *et al.*[73] Another, more principled line of research is to make the sampling strategy dependent on the training data, or less subject to variance. Yang *et al.*[72] introduced "*a rejection procedure that rejects the sample Fourier components when they do not align well with the top eigenfunctions estimated from the sampled data*" as a possible future line of research. Similar approaches can be found in the later work by the same authors,[66] or in the use of quasi Monte Carlo procedures for choosing the sequence of points.[74]

It is also worth mentioning here the work on sparse spectra in the Gaussian Processes (GPs) literature,[75] which exploits the idea of approximating the harmonic representation of the covariance function (the GP equivalent of the kernel function) inside a fully Bayesian framework. Due to this, the optimal frequencies can be chosen according to a data-dependent maximization, at the cost of the standard increase in computational burden requested by the Bayesian inference procedure.

Recurrent networks with random weights

Network architecture for reservoir computing

All the algorithms we investigated up to this point are designed for *static* data, in the sense that the order of presentation of patterns to the network does not influence its behavior. However, many real-world problems require the analysis of *temporal* data that exhibits strong temporal dependencies among subsequent patterns. One possibility to handle them is to input a feedforward network, such as the RW-FNN previously discussed, with a time-delayed embedding of the temporal series, so that a memory of the previous instants is artificially forced by providing them as inputs. However, results in terms of accuracy might

be suboptimal and, more importantly, there is no principled way of choosing an embedding dimension of the input. An alternative, more powerful choice is to consider recurrent neural networks (RNNs), wherein connections between internal neurons create a temporal processing, making the network a highly nonlinear, flexible dynamic system. RNNs have also achieved remarkable breakthroughs in the last few years with the use of multilayered architectures, gated neuronal units,[76] efficient second-order optimization routines,[77] and so on. Still, the training of fully adaptable RNNs requires the flow of error's gradient information throughout temporal instants, increasing the likelihood of vanishing or exploding gradients, and leading to possibly unstable network behaviors. Even with the diffusion of powerful software libraries allowing automatic differentiation of a cost function over time, the adaptation of RNNs remains a challenging and demanding problem.

For tasks that do not require an extremely long memory of its inputs, an alternative to a fully adaptable RNN is obtained by extending the RW-FNN idea, allowing for a *recurrent* layer of fixed, randomly generated nonlinearities, followed by an adaptable linear layer in the output. This idea, schematically shown in Fig. 3, is known in the literature as reservoir computing (RC),[6] and the two layers are denoted as 'reservoir' and 'readout' respectively. There are many flavors of algorithms in the RC field, which are briefly summarized in the next section. Here, we follow the notation and formalism of the so-called echo state network (ESN), which is common in the machine learning literature.

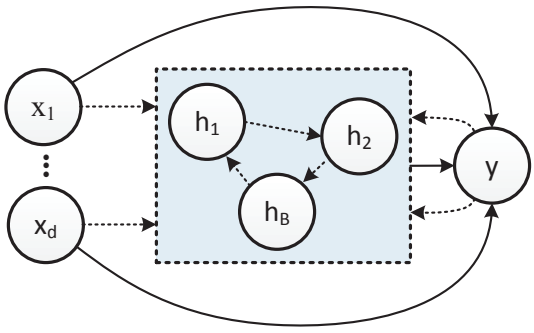

Figure 3: Depiction of an RC architecture with one output. Fixed connections are shown as dashed lines, whilst trainable connections are shown as undashed lines. The reservoir is highlighted with a light blue background.

Denote by $\mathbf{x}[n] \in \mathbb{R}^d$ the d-dimensional input of the ESN at the nth time instant, where now we use the square brackets to highlight the time dependency. Similar to before, the input is fed to a B-dimensional reservoir, whose internal state $\mathbf{h}[n-1] \in \mathbb{R}^B$ is updated according to the state equation:

$$\mathbf{h}[n] = h(\mathbf{W}_i^r \mathbf{x}[n] + \mathbf{W}_r^r \mathbf{h}[n-1] + \mathbf{w}_o^r y[n-1]) \,, \tag{19}$$

where $\mathbf{W}_i^r \in \mathbb{R}^{B \times d}$, $\mathbf{W}_r^r \in \mathbb{R}^{B \times B}$ and $\mathbf{w}_o^r \in \mathbb{R}^B$ are randomly generated matrices and vectors, $h(\cdot)$ is a suitably defined non-linear function (e.g. a sigmoid), and $y[n-1] \in \mathbb{R}$ is the previous scalar output of the network. To increase stability, it is possible to add a small uniform noise term to the state update before computing the non-linear transformation $h(\cdot)$.[78] In the second step, the current output is updated according to:

$$y[n] = (\mathbf{w}_i^o)^T \mathbf{x}[n] + (\mathbf{w}_r^o)^T \mathbf{h}[n] \,, \tag{20}$$

where $\mathbf{w}_i^o \in \mathbb{R}^{N_i}, \mathbf{w}_r^o \in \mathbb{R}^{N_r}$ are adapted based on the training data. An alternative update equation with respect to (19) is to have a reservoir that performs a leaky integration of its state (corresponding in this case to a moving average) for a given parameter $\alpha \in [0, 1]$:

$$\mathbf{h}[n] = \alpha \mathbf{h}[n-1] + (1-\alpha)\, h(\mathbf{W}_i^r \mathbf{x}[n] + \mathbf{W}_r^r \mathbf{h}[n-1] + \mathbf{w}_o^r y[n-1]) \,. \tag{21}$$

The update in (21) is the most common variation of leaky ESNs, as it only introduces one additional parameter to be tuned, but other choices (e.g., by inserting the leak factor inside the state update) are possible.[79]

ESN, and RC architectures in general, have achieved remarkable success in many applications, particularly when the problem at hand does not require very long memory processing. Representative examples include short-term load forecasting,[14] grammatical inference,[80] stock price prediction,[81] speech recognition,[82] robotic control,[83] and acoustic modeling,[84] among others. Lukoševičius and Jaeger[6] provide a complete overview of the field up to 2009. A more recent overview of the current trends in RC (some of which are detailed below) is instead provided by Goudarzi and Teuscher.[85]

Training RC networks

Due to the separation between reservoir and readout, during the training phase the internal states of the ESN can be computed for each element of the training sequence by using the desired output in (19) in place of the ESN output, a process known as 'state harvesting'. Once this is done, the training is again reduced to a standard linear regression over the concatenation of current input and internal states, allowing for an extremely fast training of a recurrent network architecture as an alternative to a full adaptation. Due to this, all training methods introduced for RW-FNN can be immediately applied also here, including training the network with sparse penalties[86], the hinge loss[87], and semi-supervised formulations.[88]

Historical overview of reservoir computing classes

The ESN as described in the previous section was formally introduced in 2001 in a technical report by Herbert Jaeger,[89] and it was designed explicitly to provide a simple, fast alternative to full adaptation of a RNN for machine learning and signal processing tasks. In this sense, the neurons in the reservoir act as dynamic (random) feature extractors that create several 'echoes' of the input sequence to be used for classification/regression purposes. For this reason, most of the literature on ESNs (a selection of which is reviewed in the next sections) has focused on creating features which are as useful as possible from a classification point of view. Along this line, some intriguing notions connecting the reservoir (seen as a randomized feature extractor), to infinite neural networks and kernels methods are explored by Hermans and Schrauwen.[90]

A very similar algorithm was derived a few years later by Steil,[91,92] under the name Backpropagation-Decorrelation (BPDC). While the BPDC is slightly more complex than the ESN described previously, it provides an interesting theoretical motivation for having a fixed reservoir. The BPDC algorithm was derived by analyzing the run-time behavior of a (at that time) state-of-the-art algorithm for RNN training, known as Atiya-Parlos.[93] The analysis, made by Schiller and Steil,[94] revealed how the algorithm results in a rapidly adapting output layer for the network, and a slowly adapting recurrent layer which, in some cases, also presents strong couplings. Based on this observation, it makes sense to directly

avoid the adaptation of the latter, resulting in a simplified linear problem and in the BPDC algorithm itself.

A similar idea to ESNs was proposed in 2002 by Maass *et al.* under the name liquid state machine (LSM)[95] from a completely different point of view, namely as a model for computational neuroscience. LSMs were put forth as a mathematical tool for analyzing the properties of neural microcircuits, showing that layers of neurons can learn to 'act' for different tasks on the information coming from a large (fixed) reservoir of neurons, exploiting the high dimensionality of the feature mapping, and thereby providing "*a computational model that does not require convergence to stable internal states or attractors*" of the recurrent neural part,[95] in contrast to the previous literature. The study of LSMs provide a direct link between artificial neural networks as used in machine learning, and neuroscience, in particular with respect to the study of random populations of neurons and their biological plausibility (e.g., by providing a connection to the decades-long investigation of P.F. Dominey and colleagues[96]). In the last decade, LSMs have been a fundamental tool for the understanding of specific neural models and their computational properties.[97,98]

Distinguished from ESNs, LSMs are designed in order to be biologically plausible, and for this reason they employ more sophisticated spiking models of neurons and, more importantly, the connectivity of the reservoir (called the 'liquid' in the LSM) follows biological principles of organization. Nonetheless, all the models discussed here are fundamentally equivalent, and in 2007, they were unified in the RC framework by the work of Verstraeten *et al.*[99] Recently, an evolution of the idea of ESNs with spiking neurons was proposed under the name NeuCube, in which temporal information for multiple time series is mapped to different parts of a 3D reservoir, based on their spatiotemporal properties.[100,101]

More generally, the idea of randomly selecting weights in RNNs was first explored by Schmidhuber and Hochreiter in a 1996 report,[102] where they showed how complete guessing was outperforming current state-of-the-art training methods in many simple problems. We briefly mention here also the 'random neural network' model, analyzed by Gelenbe in a number of works during the nineties.[103,104] In this network, randomness in introduced in the rate of firing among neurons, and the weights representing these rates are adapted by repeatedly solving two sets of linear and nonlinear equations for each input/output pair

presented to the network. Due to this, the model does not fall in the unified presentation given in this overview, although several of its idea could be exploited in connection to RC networks. A survey on the applications of these models is given by Bakırcıoğlu and Koçak[105]

Theoretical properties of RC networks

The reservoir is the central component of RC networks, and its presence is the major difference with respect to feedforward models. Although its parameters are not adapted in a supervised fashion, analyzing its dynamic behavior is essential to understanding the capabilities of RC networks. Intuitively, if we are able to avoid 'unstable' reservoirs, the universal approximation properties of RW-FNNs apply immediately: this is the topic of the first part of this section. In the rest of the section, we investigate some additional aspects related to the reservoir in terms of topology, computational power, and memory.

The echo state property

In RW-FNNs, connections in the first layer are generally extracted from a fixed probability distribution, typically a uniform distribution in $[-1, +1]$ although (as we have seen) this is not enough to guarantee its accuracy for any possible initialization. However, this approach is not sufficient for ESNs: reservoirs with random weights can result in dynamic systems which oscillate or, even worse, present chaotic behaviors. Clearly, the features extracted by these reservoirs would be of limited use in terms of discriminatory power. Basically, we should be able to guarantee that the effects of any given input and state vanish in a finite number of time-instants, without persisting indefinitely or becoming amplified exponentially. This property of reservoirs was called the echo state property (ESP) by Jaeger,[89] while a similar property is called the 'input separation property' in the LSM literature. It can be shown that, under these conditions, ESNs can approximate any time series with fading memory to the desired level of accuracy (whereas it cannot be used for tasks requiring unbounded memory).[95]

In practice, the ESP can be guaranteed *almost* always by rescaling the matrix $\mathbf{W}_r^r$ so that its spectral radius $\rho(\mathbf{W}_r^r)$ (defined as the maximum eigenvalue in absolute terms) is less

than 1. This rule originates from the work of Jaeger, which also proposed to have sparse connectivity in $\mathbf{W}_r^r$ so as to create heterogeneous clusters of features.[89] Together, these two rules represent the most common way of initializing the ESN architecture.

Nonetheless, this criterion is only heuristic and does not exactly guarantee the ESP. Jaeger[89] showed that, necessarily, the ESP is violated if $\rho(\mathbf{W}_r^r) > 1$ for networks with tanh nonlinearities and if the input space contains the zero sequence, but the criterion does not relate to ESNs driven by a specific input sequence, and the ESP might be lost (theoretically) even for $\rho(\mathbf{W}_r^r) \leq 1$. A sufficient condition in terms of the largest singular value was also obtained by Jaeger, but it is too restrictive to be useful in practice. Two equivalent (tighter) sufficient conditions were obtained by Buehner and Young in 2006 and by Yildiz *et al.* in 2012.[106,107] Taking into account the random initialization of the reservoir, Zhang *et al.*[108] instead exploited tools from random matrix theory to analyze the behavior of the network, showing that (asymptotically) the empirical criterion $\rho(\mathbf{W}_r^r) \leq 1$ is in fact justified. Some important results relating the ESP to the nature of the input signal were obtained in 2013 by Manjunath and Jaeger.[109]

We emphasize that the ESP property is not the only possibility to design stable reservoirs in a principled way. Ozturk *et al.*[110] proposed a metric based on information-theoretic arguments by analyzing the linearized dynamic of the reservoir. To date, this remains the most common alternative to the *a priori* selection of a desired spectral radius. In practice, both the input matrix and the feedback matrix have great impact on the prediction performance. Note that the criterion $\rho(\mathbf{W}_r^r) \leq 1$ can be regarded as a constraint on the distribution of the random weights of the feedback matrix, which can be characterised in the sense of ℓ_1 norm by the scope of random weights. Thus, a proper estimate on the scope setting for both the input and feedback matrices is critical to guarantee a success of time-series modelling (a typical data mining task).

Design of the reservoir

Since the training of the readout is formulated as a linear problem, most of the relevant literature can be applied also for this class of networks. As an example, ℓ_1 minimization to achieve sparse readouts was investigated independently in the context of ESNs by Ceperic

and Baric[13] and Bianchi *et al.*,[14] with some similar results obtained in a previous study by Butcher *et al.*[86] Additionally, it is possible to consider advanced optimization strategies,[111,112] support vector algorithms,[87] and many other variations (see Section 7 of Lukoševičius and Jaeger for an overview[6]).

In a more general sense, however, the isolation of the recurrent part of the network allows for a lot of freedom in its design, both by varying its initialization strategy (in terms of connectivity), and by adding unsupervised strategies for its optimization.[113,114] In the latter case, two state-of-the-art unsupervised strategies with a biological inspiration are submodular reservoirs with lateral inhibition[113] and intrinsic plasticity (IP).[114] Lateral inhibition is a biological mechanism in which the activity of a neuron might inhibit the response of another connected neuron. A way to realize this in ESNs is to have multiple (smaller) reservoirs, with pairwise negative connections to inhibit their neighbors.[113] As an example, consider two reservoirs described by matrices $\mathbf{W}_1, \mathbf{W}_2 \in \mathbb{R}^{\frac{B}{2} \times \frac{B}{2}}$, where each reservoir is now constituted by $\frac{B}{2}$ neurons. An overall reservoir with lateral inhibition can be constructed as:

$$\mathbf{W} = \begin{bmatrix} \mathbf{W}_1 & -\mathcal{P}\mathbf{I} \\ -\mathcal{P}\mathbf{I} & \mathbf{W}_2 \end{bmatrix} \tag{22}$$

where $\mathcal{P}$ is an operator that predicts the new subreservoir states in the absence of lateral inhibition, and we suppose that both inhibitory connections have the same strength.[113] Despite its simplicity, this scheme can have a significant impact on the prediction accuracy.[113] More in general, clustered architectures for the reservoir have been studied extensively, as they can also provide improvements on the overall stability of the system.[115]

IP, proposed by Steil[116] in 2007 and extended by Schrauwen *et al.*,[114] is another biologically-inspired rule which can be implemented easily. Practically, it allows for an increase in accuracy and a possible decrease in variance due to the random initialization. Suppose that the original reservoir's nonlinearity $h(s)$ is parameterized with two free adaptable values $a, b \in \mathbb{R}$ as $h_{\text{ip}} = h(as + b)$. The IP rule allows the fine-tuning of these two parameters in an online fashion, so that the distribution of the reservoirs' states approaches a predefined distribution, e.g. Gaussian. Simple update rules can be found in closed form for most nonlinearities used in practice.[114,116]

Another possibility for improving performance is to consider layered architectures in

which the outputs of one layer are provided as inputs to a subsequent layer. Triefenbach *et al.*[84] investigated this idea for acoustic modeling, in which every layer of the network is trained to distinguish between acoustic entities of increasing temporal duration. Alternative approaches, comprising multiple reservoirs connected to a *single* readout and/or between them are explored in two recent papers by Malik *et al.*[117] and Galicchio and Micheli.[118] Despite the good results obtained, multilayered models are uncommon in ESNs, and in fact they represent one of the most interesting open questions, as they might be able to process events at different time resolutions. An alternative line of research has focused on the construction of simpler, *deterministic* reservoirs. Specifically, Rodan and Tiňo[119,120] showed that cyclic reservoirs with possibly highly constrained weights can perform almost as well as large, random reservoirs. Similar insights are obtained by Strauss *et al.*[121] Together, these results raise some important questions, namely: are the random 'echoes' produced by ESNs actually helpful, or can we discard randomness and focus on the algebraic properties of the matrices? Furthermore, in this case, is this true for any possible class of problems?

Recently, Appeltant *et al.*[122] showed good performance of a 'virtual' reservoir composed of a single node receiving time-delayed versions of its input. This scheme is particularly interesting as the algorithm admits a very fast electronic implementation. Later, fully optoelectronic implementations of ESN schemes were independently presented in Paquot *et al.*[123] and Larger *et al.*,[124] paving the way to ultra-fast implementations of RC using non-conventional computing technologies.

Computational power and memory in RC architectures

Apart from the initialization strategy of the reservoir and its spectral radius, another fundamental parameter in the design of ESNs is the range in which the input connections $\mathbf{W}_i^r$ are drawn. Basically, larger connections drive the reservoir in more nonlinear regions for commonly used nonlinear functions. Together, these parameters determine the computational regime in which the ESN is operating, both in terms of computational power and memory capacity.[125,126] One important aspect that we briefly mention here is the concept of 'edge of instability' (or 'edge of chaos'): namely, a large body of literature suggests that ESNs, like other recurrent architectures and even cortical circuits, have the highest computational

power when they operate near an instable regime.[127,128] This notion is of paramount importance, as it provides a link between RC, neuroscience and the theory of complex systems. Open questions abound, particularly in order to determine *when* such processing power is required and, in this case, how we can detect the onset of criticality in order to achieve this aim.[129–131]

CONCLUSIONS

Randomly assigning a subset of parameters in a learner model is an efficient approach for training neural networks, which provides an effective solution for large scale data mining problems. Methods exploiting random mappings, such as RVFL, Random Kitchen Sinks and FastFood, commonly share a two step process, that is, random mapping using some weights (must be carefully scoped indeed for the universal approximation property) followed by resolving a linear model. Due to the formulation of optimization problems in the parameter space, all solutions can be obtained by leveraging over algebra routines and well-developed techniques for linear regression and classification. Throughout this overview, we have seen how this idea can be applied to feedforward neural networks, recurrent neural networks, and for approximating kernel methods defined over possibly infinite-dimensional feature spaces. A wealth of algorithms can be cast under a rather general framework, providing good trade-offs in terms of accuracy, training time, extensibility, and so on. Nonetheless, advancement in these models has been penalized by a fragmentation of the literature over different fields. In this paper, we provided a comprehensive overview of all of them, showing their history, strengths, weaknesses and, most importantly, open problems. As we have seen, many fundamental questions remain to be further explored, in order to bridge an existing gap between purely randomized architectures and fully adaptable networks. A particular daunting problem is to develop some practical methods for the design of sampling distributions in a data-dependent fashion, which will play a key role in dealing with streaming data mining problems.

There are some interesting and primary questions on the randomized approaches, for instance, is randomness beneficial to general data mining problems, or is there a way of con-

structing informative, functional and cheap feature spaces without randomization techniques which can perform favorably?[119,120] Additional insights might be obtained by connecting the theory presented here to the broad field of randomness and stability in learning theory,[132] which investigates how randomness in an algorithm affects the resulting error bounds. In the end, we hope that this overview will attract more researchers to pay attention on the use of randomness in the field of machine learning and data mining, fostering innovative methods and the exchange of key results among communities.

FURTHER READING

For readers interested in the field of randomization for neural networks and kernel functions, we recommend the recent special issue on the topic published in Information Sciences. Specifically, the special issue contains an informative editorial by Wang[133] and a survey paper by Zhang and Suganthan.[134] Apart from the algorithms detailed here, the survey covers additional methods that go outside the definition of randomization we considered, such as probabilistic neural networks,[135] ensemble methods,[136] dropout regularization,[137] and others. As such, it provides an excellent alternative entry point into the growing literature of the field.

References

1. X. Glorot and Y. Bengio, in *13th International Conference on Artificial Intelligence and Statistics (AISTATS)* (2010), pp. 249–256.

2. T. Hofmann, B. Schölkopf, and A. Smola, The Annals of Statistics pp. 1171–1220 (2008).

3. J. Schmidhuber, Neural Networks **61**, 85 (2015).

4. S. Kotsiantis, Informatica **31** (2007).

5. Y.-H. Pao, G.-H. Park, and D. Sobajic, Neurocomputing **6**, 163 (1994).

6. M. Lukoševičius and H. Jaeger, Computer Science Review **3**, 127 (2009).

7. A. Rahimi and B. Recht, in *Advances in Neural Information Processing Systems* (2007), pp. 1177–1184.

8. T. Cover, IEEE Transactions on Electronic Computers pp. 326–334 (1965).

9. A. Rahimi and B. Recht, in *Advances in Neural Information Processing Systems* (2009), pp. 1313–1320.

10. S. Stigler, The Annals of Statistics pp. 465–474 (1981).

11. F. Bach, R. Jenatton, J. Mairal, and G. Obozinski, Foundations and Trends® in Machine Learning **4**, 1 (2012).

12. F. Cao, Y. Tan, and M. Cai, Expert Systems with Applications **41**, 2457 (2014).

13. V. Ceperic and A. Baric, in *2014 UKSim-AMSS 16th International Conference on Computer Modelling and Simulation (UKSim)* (2014), pp. 26–31.

14. F. Bianchi, S. Scardapane, A. Uncini, A. Rizzi, and A. Sadeghian, Neural Networks **71**, 204 (2015).

15. Z. Lu, A. May, K. Liu, A. Garakani, D. Guo, A. Bellet, L. Fan, M. Collins, B. Kingsbury, M. Picheny, et al., arXiv preprint arXiv:1411.4000 (2014).

16. A. Saxe, P. Koh, Z. Chen, M. Bhand, B. Suresh, and A. Ng, in *28th International Conference on Machine Learning ICML)* (2011), pp. 1089–1096.

17. E. Bingham and H. Mannila, in *Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (ACM, 2001), pp. 245–250.

18. D. Fradkin and D. Madigan, in *Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (ACM, 2003), pp. 517–522.

19. D. Lowe and D. Broomhead, Complex Systems **2**, 321 (1988).

20. B. Igelnik, Y.-H. Pao, S. LeClair, and C. Shen, IEEE Transactions on Neural Networks **10**, 19 (1999).

21. K. Jarrett, K. Kavukcuoglu, M. Ranzato, and Y. LeCun, in *2009 IEEE 12th International Conference on Computer Vision (ICCV)* (IEEE, 2009), pp. 2146–2153.

22. C. Chu, S. Kim, Y.-A. Lin, Y. Yu, G. Bradski, A. Ng, and K. Olukotun, in *Advances in neural information processing systems* (MIT Press, 2007), pp. 281–288.

23. E. Agullo, C. Augonnet, J. Dongarra, M. Faverge, H. Ltaief, S. Thibault, and S. Tomov, in *2011 IEEE International Parallel & Distributed Processing Symposium (IPDPS)* (IEEE, 2011), pp. 932–943.

24. S. Scardapane, D. Wang, M. Panella, and A. Uncini, Information Sciences **301**, 271 (2015).

25. C. Leung, G. Young, J. Sum, and W.-K. Kan, Neural Networks, IEEE Transactions on **10**, 1482 (1999).

26. G.-X. Yuan, C.-H. Ho, and C.-J. Lin, Proceedings of the IEEE **100**, 2584 (2012).

27. F. Cao, H. Ye, and D. Wang, Information Sciences **313**, 62 (2015).

28. M. Alhamdoosh and D. Wang, Information Sciences **264**, 104 (2014).

29. G. Seber and A. Lee, *Linear regression analysis* (John Wiley & Sons, 2012).

30. F. Rosenblatt, Psychological review **65**, 386 (1958).

31. B. Widrow and M. Lehr, Proceedings of the IEEE **78**, 1415 (1990).

32. S. Gallant and D. Smith, in *1987 IEEE International Conference on Neural Networks (IJCNN)* (1987).

33. M. Minsky and S. Papert, *Perceptrons* (MIT press, 1988).

34. Y. Pao and Y. Takefji, IEEE Computer Journal **25**, 76 (1992).

35. B. Igelnik and Y.-H. Pao, IEEE Transactions on Neural Networks **6**, 1320 (1995).

36. G.-H. Park and Y.-H. Pao, Neurocomputing **31**, 45 (2000).

37. D. Husmeier and J. Taylor, Neural Networks **11**, 89 (1998).

38. J. Martínez-Villena, A. Rosado-Muñoz, and E. Soria-Olivas, Applied intelligence **41**, 184 (2014).

39. L. Zhang and P. Suganthan, Information Sciences **367–368**, 1094 (2016).

40. S. Scardapane, D. Comminiello, M. Scarpiniti, and A. Uncini, Information Sciences **364**, 156 (2016).

41. H. Cecotti, in *2016 International Joint Conference on Neural Networks (IJCNN)* (IEEE, 2016), pp. 3628–3633.

42. C. Looney, Neurocomputing **48**, 489 (2002).

43. W. Schmidt, M. Kraaijveld, and R. Duin, in *11th IAPR International Conference on Pattern Recognition (ICPR)* (IEEE, 1992), pp. 1–4.

44. R. Duin, Pattern Recognition Letters **15**, 215 (1994).

45. B. Widrow, Neural Networks **48**, 204 (2013).

46. B. Widrow, A. Greenblatt, Y. Kim, and D. Park, Neural Networks **37**, 182 (2013).

47. J. Tapson and A. van Schaik, Neural Networks **45**, 94 (2013).

48. J.-Y. Li, W. Chow, B. Igelnik, and Y.-H. Pao, IEEE Transactions on Neural Networks **8**, 452 (1997).

49. A. Barron, IEEE Transactions on Information Theory **39**, 930 (1993).

50. A. Gorban, I. Tyukin, D. Prokhorov, and K. Sofeikov, Information Sciences **364–365**, 129 (2016).

51. J. Principe and B. Chen, IEEE Computational Intelligence Magazine **10**, 68 (2015).

52. M. Li and D. Wang, Information Sciences **382-383**, 170 (2017).

53. A. Rahimi and B. Recht, in *46th Annual Allerton Conference on Communication, Control, and Computing* (IEEE, 2008), pp. 555–561.

54. A. Rudi, R. Camoriano, and L. Rosasco, arXiv preprint arXiv:1602.04474 (2016).

55. C. Cortes and V. Vapnik, Machine learning **20**, 273 (1995).

56. S. Van Vaerenbergh, M. Lázaro-Gredilla, and I. Santamaría, IEEE Transactions on Neural Networks and Learning Systems **23**, 1313 (2012).

57. X. Wu, X. Zhu, G.-Q. Wu, and W. Ding, IEEE Transactions on Knowledge and Data Engineering **26**, 97 (2014).

58. D. Achlioptas, F. Mcsherry, and B. Schölkopf, in *Advances in Neural Information Processing Systems* (MIT Press, 2002), vol. 1, pp. 335–343.

59. P. Drineas and M. Mahoney, Journal of Machine Learning Research **6**, 2153 (2005).

60. C. Williams and M. Seeger, in *Advances in Neural Information Processing Systems* (MIT Press, 2001), pp. 682–688.

61. F. Li, C. Ionescu, and C. Sminchisescu, in *Pattern Recognition* (Springer, 2010), pp. 262–271.

62. A. Vedaldi and A. Zisserman, IEEE Transactions on Pattern Analysis and Machine Intelligence **34**, 480 (2012).

63. P. Kar and H. Karnick, in *15th International Conference on Artificial Intelligence and Statistics (AISTATS)* (2012).

64. R. Chitta, R. Jin, and A. Jain, in *2012 IEEE 12th International Conference on Data Mining (ICDM)* (IEEE, 2012), pp. 161–170.

65. Q. Le, T. Sarlós, and A. Smola, in *30th International Conference on Machine Learning (ICML)* (2013), pp. 244–252.

66. Z. Yang, A. Smola, L. Song, and A. Wilson, in *18th International Conference on Artificial Intelligence and Statistics (AISTATS)* (2015).

67. B. Dai, B. Xie, N. He, Y. Liang, A. Raj, M.-F. F. Balcan, and L. Song, in *Advances in Neural Information Processing Systems* (2014), pp. 3041–3049.

68. A. Sinha and J. C. Duchi, in *Advances In Neural Information Processing Systems* (2016), pp. 1298–1306.

69. J. Lu, S. Hoi, J. Wang, P. Zhao, and Z.-Y. Liu, Journal of Machine Learning Research **17**, 1 (2016).

70. K. Chaudhuri, C. Monteleoni, and A. Sarwate, Journal of Machine Learning Research **12**, 40 (2009), ISSN 1532-4435, 0912.0071.

71. M. Lin, S. Weng, and C. Zhang, ACM Transactions on Knowledge Discovery from Data (TKDD) **8**, 13 (2014).

72. T. Yang, Y.-F. Li, M. Mahdavi, R. Jin, and Z.-H. Zhou, in *Advances in Neural Information Processing Systems* (2012), pp. 476–484.

73. R. Hamid, Y. Xiao, A. Gittens, and D. Decoste, arXiv preprint arXiv:1312.4626 (2013).

74. J. Yang, V. Sindhwani, H. Avron, and M. Mahoney, in *31st International Conference on Machine Learning (ICML)* (2014), pp. 485–493.

75. M. Lázaro-Gredilla, J. Quiñonero-Candela, C. E. Rasmussen, and A. R. Figueiras-Vidal, Journal of Machine Learning Research **11**, 1865 (2010).

76. I. Sutskever, O. Vinyals, and Q. Le, in *Advances in Neural Information Processing Systems* (2014), pp. 3104–3112.

77. J. Martens and I. Sutskever, in *Neural Networks: Tricks of the Trade, Reloaded* (Springer Berlin Heidelberg, 2012), chap. 20, pp. 479–535.

78. H. Jaeger, in *Advances in Neural Information Processing Systems* (2002), pp. 593–600.

79. H. Jaeger, M. Lukoševičius, D. Popovici, and U. Siewert, Neural Networks **20**, 335 (2007).

80. M. Tong, A. Bickett, E. Christiansen, and G. Cottrell, Neural Networks **20**, 424 (2007).

81. X. Lin, Z. Yang, and Y. Song, Expert Systems with Applications **36**, 7313 (2009).

82. M. D. Skowronski and J. G. Harris, Neural Networks **20**, 414 (2007).

83. E. Aislan Antonelo and B. Schrauwen, IEEE Transactions on Neural Networks and Learning Systems **26**, 763 (2015).

84. F. Triefenbach, A. Jalalvand, K. Demuynck, and J.-P. Martens, IEEE Transactions on Audio, Speech, and Language Processing **21**, 2439 (2013).

85. A. Goudarzi and C. Teuscher, in *3rd ACM International Conference on Nanoscale Computing and Communication* (ACM, 2016), p. 13.

86. J. Butcher, C. Day, P. Haycock, D. Verstraeten, and B. Schrauwen, in *2010 IEEE International Workshop on Machine Learning for Signal Processing (MLSP)* (2010), pp. 250–255.

87. D. Li, M. Han, and J. Wang, IEEE Transactions on Neural Networks and Learning Systems **23**, 787 (2012).

88. S. Scardapane and A. Uncini, Cognitive Computation pp. 1–11 (2016).

89. H. Jaeger, German National Research Center for Information Technology GMD Technical Report **148** (2001).

90. M. Hermans and B. Schrauwen, Neural Computation **24**, 104 (2012).

91. J. Steil, in *2004 IEEE International Joint Conference on Neural Networks (IJCNN)* (IEEE, 2004), vol. 2, pp. 843–848.

92. J. Steil, Neurocomputing **69**, 642 (2006).

93. A. Atiya and A. Parlos, IEEE Transactions on Neural Networks **11**, 697 (2000).

94. U. Schiller and J. Steil, Neurocomputing **63**, 5 (2005).

95. W. Maass, T. Natschläger, and H. Markram, Neural Computation **14**, 2531 (2002).

96. P. Dominey, Biological Cybernetics **73**, 265 (1995).

97. T. Yamazaki and S. Tanaka, Neural Networks **20**, 290 (2007).

98. X. Hinaut and P. Dominey, PloS one **8**, e52946 (2013).

99. D. Verstraeten, B. Schrauwen, M. D'Haene, and D. Stroobandt, Neural Networks **20**, 391 (2007).

100. N. Kasabov and E. Capecci, Information Sciences **294**, 565 (2015).

101. E. Tu, N. Kasabov, and J. Yang, IEEE Transactions on Neural Networks and Learning Systems (2016), in press.

102. J. Schmidhuber and S. Hochreiter, *Guessing can outperform many long time lag algorithms* (Technical Note IDSIA-19-96, 1996).

103. E. Gelenbe, Neural Computation **2**, 239 (1990).

104. E. Gelenbe, Neural Computation **5**, 154 (1993).

105. H. Bakırcıoğlu and T. Koçak, European Journal of Operational Research **126**, 319 (2000).

106. M. Buehner and P. Young, IEEE Transactions on Neural Networks **17**, 820 (2006).

107. I. Yildiz, H. Jaeger, and S. Kiebel, Neural networks **35**, 1 (2012).

108. B. Zhang, D. Miller, and Y. Wang, IEEE Transactions on Neural Networks and Learning Systems **23**, 175 (2012).

109. G. Manjunath and H. Jaeger, Neural Computation **25**, 671 (2013).

110. M. Ozturk, D. Xu, and J. Príncipe, Neural Computation **19**, 111 (2007).

111. C. Sheng, J. Zhao, Y. Liu, and W. Wang, Neurocomputing **82**, 186 (2012).

112. S. Scardapane, D. Wang, and M. Panella, Neural Networks **78**, 65 (2016).

113. Y. Xue, L. Yang, and S. Haykin, Neural Networks **20**, 365 (2007).

114. B. Schrauwen, M. Wardermann, D. Verstraeten, J. Steil, and D. Stroobandt, Neurocomputing **71**, 1159 (2008).

115. M. Kaiser, C. Hilgetag, and R. Kötter, Frontiers in Neuroinformatics **4** (2010).

116. J. Steil, Neural Networks **20**, 353 (2007).

117. Z. Malik, A. Hussain, and Q. Jonathan Wu, IEEE Transactions on Cybernetics (2016), in press.

118. C. Galicchio and A. Micheli, in *ESANN 2016 - European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning* (2016), pp. 1–6.

119. A. Rodan and P. Tiňo, IEEE Transactions on Neural Networks **22**, 131 (2011).

120. A. Rodan and P. Tiňo, Neural Computation **24**, 1822 (2012).

121. T. Strauss, W. Wustlich, and R. Labahn, Neural Computation **24**, 3246 (2012).

122. L. Appeltant, M. Soriano, G. Van der Sande, J. Danckaert, S. Massar, J. Dambre, B. Schrauwen, C. Mirasso, and I. Fischer, Nature Communications **2**, 468 (2011).

123. Y. Paquot, F. Duport, A. Smerieri, J. Dambre, B. Schrauwen, M. Haelterman, and S. Massar, Scientific Reports **2** (2012).

124. L. Larger, M. Soriano, D. Brunner, L. Appeltant, J. Gutiérrez, L. Pesquera, C. Mirasso, and I. Fischer, Optics Express **20**, 3241 (2012).

125. H. Jaeger, *Short term memory in echo state networks* (German National Research Center for Information Technology GMD Technical Report, 2001).

126. D. Verstraeten, J. Dambre, X. Dutoit, and B. Schrauwen, in *2010 IEEE International Joint Conference on Neural Networks (IJCNN)* (IEEE, 2010), pp. 1–8.

127. N. Bertschinger and T. Natschläger, Neural Computation **16**, 1413 (2004).

128. J. Boedecker, O. Obst, J. Lizier, N. Mayer, and M. Asada, Theory in Biosciences **131**, 205 (2012).

129. A. Goudarzi and D. Stefanovic, Procedia Computer Science **41**, 176 (2014).

130. L. Livi, F. Bianchi, and C. Alippi, arXiv preprint arXiv:1603.03685 (2016).

131. F. Bianchi, L. Livi, and C. Alippi, arXiv preprint arXiv:1601.07381 (2016).

132. A. Elisseeff, T. Evgeniou, and M. Pontil, Journal of Machine Learning Research pp. 55–79 (2005).

133. D. Wang, Information Sciences **364–365**, 126 (2016).

134. L. Zhang and P. Suganthan, Information Sciences **364–365**, 146 (2016).

135. R. Salakhutdinov and G. Hinton, in *12th International Conference on Artificial Intelligence and Statistics (AISTATS)* (2009), pp. 448–455.

136. Y. Ren, L. Zhang, and P. Suganthan, IEEE Computational Intelligence Magazine **11**, 41 (2016).

137. N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, Journal of Machine Learning Research **15**, 1929 (2014).