Integer Echo State Networks: Efficient Reservoir Computing for Digital Hardware

Denis Kleyko, *Member, IEEE*, Edward Paxon Frady, Mansour Kheffache, and Evgeny Osipov, *Associate Member, IEEE*

***Abstract*— We propose an approximation of echo state networks (ESNs) that can be efficiently implemented on digital hardware based on the mathematics of hyperdimensional computing. The reservoir of the proposed integer ESN (intESN) is a vector containing only n-bits integers (where $n < 8$ is normally sufficient for a satisfactory performance). The recurrent matrix multiplication is replaced with an efficient cyclic shift operation. The proposed intESN approach is verified with typical tasks in reservoir computing: memorizing of a sequence of inputs, classifying time series, and learning dynamic processes. Such architecture results in dramatic improvements in memory footprint and computational efficiency, with minimal performance loss. The experiments on a field-programmable gate array confirm that the proposed intESN approach is much more energy efficient than the conventional ESN.**

***Index Terms*— Dynamic systems modeling, echo state networks (ESNs), hyperdimensional computing (HDC), memory capacity, reservoir computing (RC), time-series classification, vector symbolic architectures.**

I. Introduction

RECENT work in reservoir computing (RC) [1], [2] illustrates how a recurrent neural network with fixed connectivity can memorize and generate complex spatiotemporal sequences. RC has been shown to be a powerful tool for modeling and predicting dynamic systems, both living [3] and technical [4], [5]. Recently, it has been shown in [6] that RC is able to predict large chaotic systems.

Manuscript received April 13, 2019; revised February 10, 2020 and November 20, 2020; accepted November 25, 2020. Date of publication December 22, 2020; date of current version April 5, 2022. This work was supported in part by the Swedish Research Council under Grant 2015-04677. The work of Denis Kleyko was supported by the European Union's Horizon 2020 Research and Innovation Programme under the Marie Skłodowska-Curie Individual Fellowship Grant 839179 and in part by the Defense Advanced Research Projects Agency (DARPA's) Virtual Intelligence Processing (VIP) (Super-HD Project) and the Artificial Intelligence Exploration (AIE) (HyDDENN Project) programs. *(Corresponding author: Denis Kleyko.)*

Denis Kleyko is with the Redwood Center for Theoretical Neuroscience, University of California at Berkeley, Berkeley, CA 94720 USA, and also with the Intelligent Systems Lab, Research Institutes of Sweden, 164 40 Kista, Sweden (e-mail: denis.kleyko@ri.se).

Edward Paxon Frady is with the Neuromorphic Computing Lab, Intel Labs, Santa Clara, CA 95054 USA, and also with the Redwood Center for Theoretical Neuroscience, University of California at Berkeley, Berkeley, CA 94720 USA (e-mail: epaxon@berkeley.edu).

Mansour Kheffache is with Netlight Consulting AB, 111 53 Stockholm, Sweden (e-mail: mansour.kheffache@netlight.com).

Evgeny Osipov is with the Department of Computer Science Electrical and Space Engineering, Luleå University of Technology, 971 87 Luleå, Sweden (e-mail: evgeny.osipov@ltu.se).

This article has supplementary material provided by the authors and color versions of one or more figures available at https://doi.org/10.1109/TNNLS.2020.3043309.

Digital Object Identifier 10.1109/TNNLS.2020.3043309

Recent work on feedforward networks shows that the binarization of filters in convolutional neural networks can lead to enormous gains in memory and computational efficiency [7]. Reducing the memory allocated to each neuron or synapse from a 32-bit float to a few bits or binary saves computation with minimal loss in performance (see [8], [9]). The increase in efficiency broadens the range of applications for these networks.

This article addresses two important research directions in RC: training reservoir networks and implementing networks efficiently. We discovered several direct functional similarities between the operations in RC and those of hyperdimensional computing (HDC) [10]. HDC [11] or more generally vector symbolic architectures [12] are frameworks for neural symbolic representation, computation, and analogical reasoning. The distinction from the traditional computing is that all entities (objects, phonemes, and symbols) are represented by random vectors of very high dimensionality—several thousand dimensions. Complex data structures and analogical reasoning are implemented by simple arithmetical operations (binding, addition/bundling, and permutation) and a well-defined similarity metric [11]. Specifically, RC and HDC are connected by the following core principles.

1) Random projections of input values onto a reservoir (which in essence is a high-dimensional vector) match random HDC representations stored in a superposition.
2) The update of the reservoir by a random recurrent connection matrix is similar to HDC binding/permutation operation.
3) The nonlinearity of the reservoir can be approximated with the thresholded addition of integers in HDC.

We exploited these findings to design integer echo state networks (intESNs), which performs like echo state networks (ESNs) but with smaller memory footprint and computational cost.

In the proposed architecture, the reservoir of the network contains only constrained integers for each neuron, reducing the memory of each neuron from a 32-bit float to only a few bits. The recurrent matrix multiply update is replaced by a permutation (or even a cyclic shift), which results in the dramatic boosting of the computational efficiency. We validate the architecture on several tasks common in the RC literature. All examples demonstrate the satisfactory approximation of performance of the conventional ESN, while the implementation on field-programmable gate

array (FPGA) confirms the amenability of intESN for digital hardware.

This article is structured as follows. Background and related work are presented in Section II. The main contribution—intESNs—is described in Section III. The performance evaluation follows in Section IV. Section V presents the experiments on digital hardware. Sections VI and VII present a discussion and conclusions, respectively.

II. Background and Related Work

There are many practical tasks that require the history of inputs to be solved. In the area of artificial neural networks (ANNs), such tasks require working memory. This could be implemented by recurrent connections between neurons of an RNN. Training RNNs is much harder than that of feedforward ANNs (FFNNs) due to the vanishing gradient problem [13].

The challenge of training RNNs was addressed from two approaches. One approach eliminates the vanishing gradient problem through neurons with special memory gates [14]. Another approach is to reformulate the training process by learning only connections to the last readout layer while keeping the other connections fixed. This approach originally appeared in two similar architectures: liquid state machines [15] and ESNs [16], now referred to as RC [1].

It is interesting to note that similar ideas were conceived in the area of FFNNs, which can be seen as an RNN without memory and are known under the name of random vector functional link (RVFL) [17] or extreme learning machines (ELMs) [18]. RVFLs/ELMs are used to solve various machine learning problems, including classification, clustering, and regression [19].

Important applications of RC are the modeling and predicting of complex dynamic systems. Generating and predicting chaotic systems was an important use case from the beginning [20], for example, ESNs were used for chaotic time series from low-order aberrations caused by turbulence [21]. A thorough study on emulating chaotic systems was recently presented in [22]. It was also shown that ESNs can be used for forecasting electroencephalography signals and for solving classification problems in the context of brain–computer interfaces [23]. There are different classification strategies and readout methods when performing classification of time series with ESN. In [24], three classification strategies and three readout methods were explored under the conditions that testing data are purposefully polluted with noise. Interestingly, different readout methods are preferable in different noise conditions. Recent work in [25] also studied the classification of multivariate time series with ESN using standard benchmarking data sets. The work covered several advanced approaches, which extend the conventional ESN architecture, for generating a representation of a time series in a reservoir.

Another recent research area is binary RC with cellular automata (CARC) that started as an interdisciplinary research within three areas: cellular automata, RC, and HDC. CARC was initially explored in [26] for projecting binarized features into high-dimensional space. Furthermore, in [27], it was applied for modality classification of medical images. An alternative classifier based on cellular automata and HDC was presented in [28]. Cellular automata can also be used to form weight matrices for RC and HDC [29], [30]. The usage of CARC for symbolic reasoning is explored in [31]. The memory characteristics of a reservoir formed by CARC are presented in [32]. Work [33] proposed the usage of coupled cellular automata in CARC. Examples of recent RC developments also include advanced architectures, such as deep RC [34], [35], Laplacian ESN [36], learning of reservoir's size and topology [37], new tools for investigating reservoir dynamics [38], and determining its edge of criticality [39].

The design of ESNs has been an important research area (see [40]–[42]). One important aspect of the design is, of course, the choice of network's parameters for a given task. Another important aspect considered in this study is the computational complexity. One of the ways of reducing computational costs would be to use quantized reservoir states. It was explored in fractal prediction machines [43] and neural prediction machines [44], [45] RC models. Another way of reducing computational costs involves modifications of network's connectivity structure. In this respect, the approach that is ideologically closest to our intESN, was presented in [46]. The authors demonstrated that a simple cycle reservoir (referred to as the ring-based ESN) can be used to achieve a performance similar to the conventional ESN. Similar conclusions about the ring-based ESN were obtained in [42] when studying different design strategies for reservoir connection matrices in four typical RC tasks. While the ring-based ESN explored reservoir update solution, which is similar to one of our optimizations, the technical side is very different from our approach as intESN strives at using only integers as neurons activation values.

While in this article the main focus is on reservoir states comprised of integers only, it is worth mentioning related works considering the general problem of reducing the computational complexity of the conventional ESN. For example, several optimizations were used in [47] in order to deploy an ESN on a resource-constrained device for anomaly detection. These optimizations include sparse matrix algebra via compressed row storage for weights of connections between the input layer neurons and the reservoir, single floating-point precision, and an activation function, which resembles tanh() function but has lower complexity. Similarly, in works [48], [49], ESNs were used in the context of user movements prediction on resource-constrained devices; therefore, the authors studied how the parameters of the network will affect computational costs and task performance. In particular, the varied parameters were sparsity of reservoir connection matrix, number of bits per weight, and number of neurons in reservoir.

A. Echo State Networks

This section summarizes the functionality of the conventional ESN, and it follows the description in [50] for a special case of leaky integration when $\alpha = 1$.[1] Fig. 1 shows the architectural design of the conventional ESN, which includes three layers of neurons. The input layer with K neurons represents the current value of input signal denoted as $\mathbf{u}(n)$.

[1]For the detailed tutorial on ESNs, diligent readers are referred to [50].

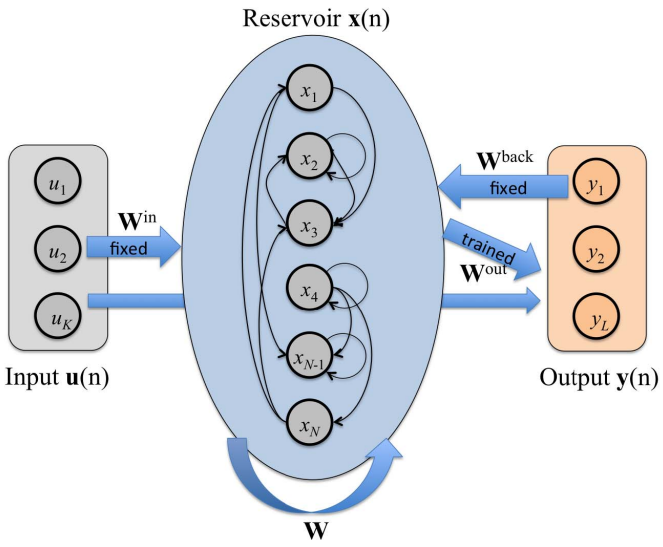


Fig. 1. Architecture of the conventional ESN.

The output layer (L neurons) produces the output of the network [denoted as $\mathbf{y}(n)$] during the operating phase. The reservoir is the hidden layer of the network with N neurons, with the state of the reservoir at time n denoted as $\mathbf{x}(n)$.

In general, the connectivity of ESN is described by four matrices. $\mathbf{W}^{\text{in}}$ describes connections between the input layer neurons and the reservoir, and $\mathbf{W}^{\text{back}}$ does the same for the output layer. Both matrices project the current input and output to the reservoir. The memory in ESN is due to the recurrent connections between neurons in the reservoir, which are described in the reservoir matrix $\mathbf{W}$. Finally, the matrix of readout connections $\mathbf{W}^{\text{out}}$ transforms the current activity levels in the input layer and reservoir ($\mathbf{u}(n)$ and $\mathbf{x}(n)$, respectively) into the network's output $\mathbf{y}(n)$. Note that three matrices ($\mathbf{W}^{\text{in}}$, $\mathbf{W}^{\text{back}}$, and $\mathbf{W}$) are randomly generated at the network initialization and stay fixed during the network's lifetime. Thus, the training process is focused on learning the readout matrix $\mathbf{W}^{\text{out}}$. There are no strict restrictions for the generation of projection matrices $\mathbf{W}^{\text{in}}$ and $\mathbf{W}^{\text{back}}$. They are usually randomly drawn from either normal or uniform distributions and scaled as shown next. The reservoir connection matrix, however, is restricted to possess the echo state property. This property is achieved when the spectral radius of the matrix $\mathbf{W}$ is less or equal than one. For example, $\mathbf{W}$ can be generated from a normal distribution and then normalized by its maximal eigenvalue. Unless otherwise stated, in this article, an orthogonal matrix was used as the reservoir connection matrix; such a matrix was formed by applying QR decomposition to a random matrix generated from the standard normal distribution. Also, $\mathbf{W}$ can be scaled by a feedback strength parameter [see (1)].

The update of the network's reservoir at time n is described by the following equation:

$$\mathbf{x}(n) = \tanh(\rho \mathbf{W}\mathbf{x}(n-1) + \beta \mathbf{W}^{\text{in}}\mathbf{u}(n) + \beta \mathbf{W}^{\text{back}}\mathbf{y}(n-1)) \tag{1}$$

where β and ρ denote the projection gain and the feedback strength, respectively. Note that it is assumed that the spectral radius of the reservoir connection matrix $\mathbf{W}$ is one. Note also that at each time step, neurons in the reservoir apply tanh() as the activation function. The nonlinearity prevents the network from exploding by restricting the range of possible values from -1 to 1. The activity in the output layer is calculated as

$$\hat{\mathbf{y}}(n) = g(\mathbf{W}^{\text{out}}[\mathbf{x}(n); \mathbf{u}(n)]) \tag{2}$$

where the semicolon denotes the concatenation of two vectors and $g()$ denotes the activation function of the output neurons, for example, linear or Winner-take-all.

1) Training Process: This article only considers training with supervised learning when the network is provided with the ground-truth desired output at each update step. The reservoir states $\mathbf{x}(n)$ are collected together with the ground truth $\mathbf{y}(n)$ for each training step. The weights of the output layer connections are acquired by solving the regression problem, which minimizes the mean square error between predictions (2) and the ground truth. While this article does not focus on the readout training task, it should be noted that there are many alternatives reported in the literature, including the usage of regression with regularization and online update rules [50].

B. Fundamentals of Hyperdimensional Computing

In a localist representation, which is used in all modern digital computers, a group of bits is needed in its entirety to interpret a representation. In HDC [11], [51], [52], all entities (objects, phonemes, symbols, and items) are represented by vectors of very high dimensionality—thousands of bits. The information is spread out in a distributed representation, which contrary to the localist representations, any subset of the bits can be interpreted. Computing with distributed representations utilizes statistical properties of vector spaces with very high dimensionality, which allows for approximate, noise-tolerant, highly parallel computations. Item memory (also referred to as clean-up memory) is needed to recover composite representations assigned to complex concepts. There are several flavors of HDC with distributed representations, differentiated by the random distribution of vector elements, which can be real numbers [51], [53]–[55], complex numbers [56], binary numbers [11], [57], or bipolar [53], [58].

We rely on the mathematics of HDC with bipolar distributed representations to develop intESN. Kanerva [11] proposed the use of distributed representations comprising $N = 10\,000$ binary elements (referred to as HD vectors). The values of each element of an HD vector are independent equally probable, and hence, they are also called densely distributed representations. The similarity between two binary HD vectors is characterized by Hamming distance, which (for two vectors) measures the number of elements in which they differ. In very high dimensions, Hamming distances (normalized by the dimensionality N) between any arbitrary chosen HD vector and all other vectors in the HD space are concentrated around 0.5. Interested readers are referred to [11] and [59] for comprehensive analysis of probabilistic properties of the high-dimensional representational space.

The binary HD vectors can be equivalently mapped to the case of bipolar representations, i.e., where each vector's element is encoded as "-1" or "$+1$." This definition is

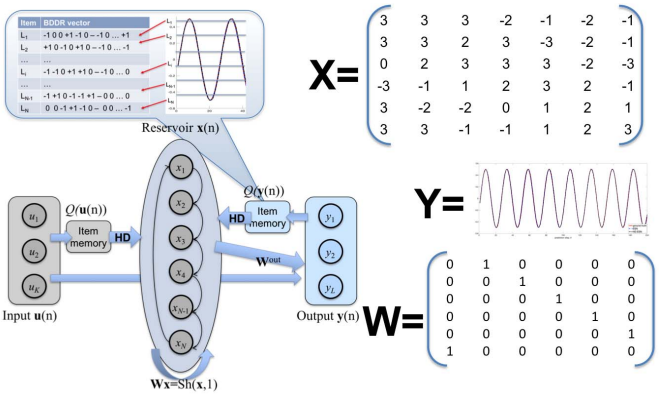


Fig. 2. Architecture of the proposed intESN.

sometimes more convenient for purely computational reasons. The distance metric for the bipolar case is a dot product

$$\text{dist} = \mathbf{x}^\top \mathbf{y}. \tag{3}$$

Basic symbols in HDC are referred to as atomic HD vectors. They are generated randomly and independently and, due to high dimensionality, will be nearly orthogonal with very high probability, i.e., similarity (dot product) between such HD vectors is approximately 0. An ordered sequence of symbols can be encoded into a composite HD vector using the atomic HD vectors, the permutation (e.g., cyclic shift as a special case of permutation), and bundling operations. This vector encodes the entire sequence history in the composite HD vector and resembles a neural reservoir.

Normally, in HDC, the recovery of component atomic HD vectors from a composite HD vector is performed by finding the most similar vectors stored in the item memory.[2] However, as more vectors are bundled together, there is more interference noise and the likelihood of recovering the correct atomic HD vector declines.

Our recent work [10] reveals the impact of interference noise and shows that different flavors of HDC have universal memory capacity. Thus, the different flavors of HDC can be interchanged without affecting performance. From these insights, we are able to design much more efficient networks for RC for digital hardware.

III. Integer Echo State Networks

This section presents the main contribution of the article—an architecture for intESN. The architecture is shown in Fig. 2. The proposed intESN is structurally identical to the conventional ESN (see Fig. 1) with three layers of neurons: input ($\mathbf{u}(n)$ and K neurons), output ($\mathbf{y}(n)$ and L neurons), and reservoir ($\mathbf{x}(n)$ and N neurons). It is important to note from the beginning that training the readout matrix $\mathbf{W}^{\text{out}}$ for intESN is the same as for the conventional ESN (see Section II-A1).

However, other components of intESN differ from the conventional ESN. First, activations of input and output layers are projected into the reservoir in the form of bipolar HD vectors [54] of size N [denoted as $\mathbf{u}^{\text{HD}}(n)$ and $\mathbf{y}^{\text{HD}}(n)$]. For problems where input and output data are described by finite alphabets and each symbol can be treated independently, the mapping to N-dimensional space is achieved by simply assigning a random bipolar HD vector to each symbol in the alphabet and storing them in the item memory [11], [60]. In the case with continuous data (e.g., real numbers), we quantized the continuous values into a finite alphabet. The quantization scheme (denoted as Q) and the granularity of the quantization are problem dependent. In addition, when there is a need to preserve similarity between quantization levels, distance preserving mapping schemes are applied (see [61], [62]), which can preserve, for example, linear or nonlinear similarity between levels. An example of a discretization and quantization of a continuous signal as well as its HD vectors in the item memory is shown in Fig. 2. Continuous values can also be represented in HD vectors by varying their density. For a recent overview of several mapping approaches, readers are referred to [63]. Also, an example of applying such mapping is presented in Section IV-A2. Another feature of intESN is the way the recurrence in the reservoir is implemented. Rather than a matrix multiply, recurrence is implemented via the permutation of the reservoir vector. Note that permutation of a vector can be described in matrix form, which can play the role of $\mathbf{W}$ in intESN. Note that the spectral radius of this matrix equals one. However, an efficient implementation of permutation can be achieved for a special case—cyclic shift (denoted as Sh()). It is important to note that we have shown in [10] that the recurrent weight matrix $\mathbf{W}$ creates key-value pairs of the input data. Note that $\mathbf{W}$ is chosen randomly and kept fixed, and this always leads to the same properties. Moreover, there is no advantage of the fully connected random recurrent weight matrix over the simple cyclic shift operation for storing the input history. Thus, the use of the cyclic shift in place of a random recurrent weight matrix does not limit intESN's ability to produce linearly separable representations. Fig. 2 shows the recurrent connections of neurons in a reservoir with recurrence by cyclic shift of one position. In this case, vector-matrix multiplication $\mathbf{W}\mathbf{x}(n)$ is equivalent to $\text{Sh}(\mathbf{x}(n), 1)$.

Finally, to keep the integer values of neurons, intESN uses different nonlinear activation function for the reservoir—clipping (4). Note that the simplest bundling operation is an elementwise addition. However, when using the elementwise addition, the activity of a reservoir (i.e., a composite HD vector) is no longer bipolar. From the implementation point of view, it is practical to keep the values of the elements of the HD vector in the limited range using a threshold value (denoted as κ)

$$f_\kappa(x) = \begin{cases} -\kappa, & x \le -\kappa \\ x, & -\kappa < x < \kappa \\ \kappa, & x \ge \kappa. \end{cases} \tag{4}$$

The clipping threshold κ is regulating the nonlinear behavior of the reservoir and limiting the range of activation values. Note that in intESN, the reservoir is updated only with integer bipolar vectors, and after clipping, the values of neurons are still integers in the range between $-\kappa$ and κ. Thus, each

[2]It is not common to do such decoding in RC. Normally, in the scope of RC, a readout matrix is learned. In this article, we follow this standard RC approach to extracting information back from a reservoir.

neuron can be represented using only $\log_2(2\kappa + 1)$ bits of memory. For example, when $\kappa = 7$, there are 15 unique values of a neuron, which can be stored with just four bits. We have also shown recently that the usage of the clipping might be beneficial when implementing resource-efficient alternatives of self-organizing maps [64].

Summarizing the aforementioned differences, the update of intESN is described as

$$\mathbf{x}(n) = f_\kappa(\mathrm{Sh}(\mathbf{x}(n-1), 1) + \mathbf{u}^{\mathrm{HD}}(n) + \mathbf{y}^{\mathrm{HD}}(n-1)). \quad (5)$$

IV. Performance Evaluation

In this section, the proposed intESN approach is verified and compared to the conventional ESN and the ring-based ESN [46] on a set of typical RC tasks. In particular, three aspects are evaluated: short-term memory, classification of time series, and modeling of dynamic processes. Short-term memories are compared using the trajectory association task [56], introduced in the area of holographic reduced representations [51]. In addition, an approach for storing and decoding analog values using intESN is demonstrated on image patches. Classification of time series is studied using the standard data sets from UCI and UCR. Modeling of dynamic processes is tested on two typical cases. First, the task of learning a simple sinusoidal function is considered. Next, networks are trained to reproduce a complex dynamical system produced by a Mackey–Glass series. Unless otherwise stated, ridge regression (the regularization coefficient is denoted as λ) with the Moore–Penrose pseudoinverse was used to learn the readout matrix $\mathbf{W}^{\mathrm{out}}$. The values of the input neurons $\mathbf{u}(n)$ were not used for training the readout in any of the experiments in the following.

A. Short-Term Memory

1) Sequence Recall Task: The sequence recall task includes two stages: memorization and recall. At the memorization stage, a network continuously stores a sequence of tokens (e.g., letters and phonemes). The number of unique tokens is denoted as D ($D = 27$ in the experiments), and one token is presented as input each timestep. At the recall stage, the network uses the content of its reservoir to retrieve the token stored d steps ago, where d denotes delay. In the experiments, the range of delay varied between 0 and 15.

For the conventional and ring-based ESNs, the dictionary of tokens was represented by a one-hot encoding, i.e., the number of input layer neurons was set to the size of the dictionary $K = D = 27$. The same encoding scheme was adopted for the output layer, $L = 27$. The input vector was projected to the reservoir by the projection matrix $\mathbf{W}^{in}$ where each entry was independently generated from the uniform distribution in the range $[-1, 1]$, and the projection gain was set to $\beta = 0.1$. The reservoir connection matrix $\mathbf{W}$ for the conventional ESN was first generated from the standard normal distribution and then orthogonalized. The reservoir connection matrix $\mathbf{W}$ for the ring-based ESN was generated as a permutation matrix. The feedback strength of both reservoir connection matrices was set to $\rho = 0.94$.

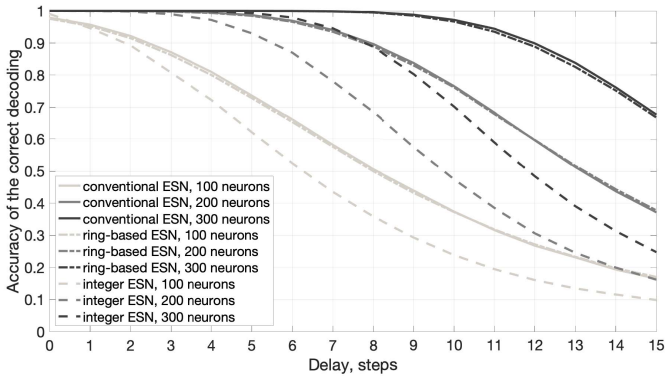


Fig. 3. Accuracy of the correct decoding of tokens for the conventional ESN, ring-based ESN, and intESN for three different values of N.

For intESN, the item memory was populated with D random high-dimensional bipolar vectors. The threshold for the clipping function was set to $\kappa = 3$. The output layer was the same as in ESN with $L = 27$ and one-hot encoding of tokens. It is worth noting that ρ and κ were chosen in such a way that the accuracy curves would resemble each other as close as possible. The diligent readers are kindly referred to the Supplementary Materials (Fig. S.1) where, for the case $N = 200$, the curves for the range of ρ and κ values are presented.

For each value of the delay d, a readout matrix $\mathbf{W}^{\mathrm{out}}$ was trained, producing 16 matrices in total. The training sequence presented 2000 random tokens to the network, and only the last 1500 steps were used to compute the readout matrices. The regularization parameter for ridge regression was set to $\lambda = 0$. The training sequence of tokens delayed by the particular d was used as the ground truth for the activations of the output layer. During the operating phase, both the inclusion of a new token into the reservoir and the recall of the delayed token from the reservoir were simultaneous. Experiments were performed for three different sizes of the reservoir: $N = 100$, $N = 200$, and $N = 300$.

The memory capacity of the network is characterized by the accuracy of the correct decoding of tokens for different values of the delay. Fig. 3 shows the accuracy for all networks conventional ESN (solid lines), ring-based ESN (dashed–dotted line), and intESN (dashed lines). The capacities of all the networks grow with the increased number of neurons in the reservoir. Since the capacities of the conventional ESN and the ring-based ESN are almost identical, which is in line with [42], for the rest of this section, we assume both of these networks when using the term ESN.

The capacities of ESN and intESN are comparable for small d, i.e., for the most recent tokens. For the increased delays, the curves featured slightly different behaviors. With increase in the value of d, the performance of intESN started to decline faster compared to ESN. Eventually, all curves converge to the value of the random guess, which equals $1/D$. Moreover, the information capacity of a network is characterized by the amount of information decoded from the reservoir. This amount is determined using the amount of information per token ($\log_2 D$), the probability of correctly

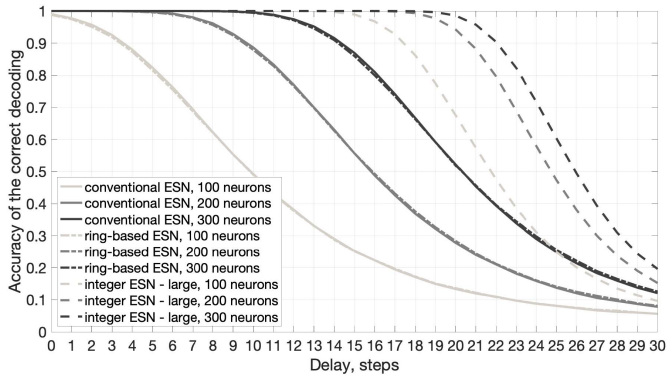


Fig. 4. Accuracy of the correct decoding of tokens for the conventional ESN, ring-based ESN, and intESN for three different values of N. "intESN-large" refers to the fact that the number of neurons in intESN was equivalent to the memory footprint required by ESN for the stated number of neurons.

decoding a token at each delay value, and the concept of mutual information. We calculated the amount of information for all networks in Fig. 3 in the considered delay range. For 100 neurons, intESN preserved 19.3% less information, and for 200 and 300 neurons, intESN preserved 21.7% less.

These results highlight a very important tradeoff: the performance versus a complexity of implementation. While the performance of intESN is somewhat poorer in this task, one has to bear in mind its memory footprint. With the clipping threshold $\kappa = 3$, only 3-bit are needed to represent the state of a neuron compared to 32-bit per neuron (the size of type float) in ESN. In other words, intESN allowed lowering the memory footprint of the reservoir by an order of magnitude by sacrificing only a fraction of the performance with respect to the information capacity. Thus, we conjecture that some reduction in the performance for ten folds memory footprint reduction is an acceptable price in applications on resource-constrained computing devices. On the other hand, we can check the performance of the networks with equal memory footprints. For this, we increased the number of neurons in intESN so that the total memory consumed by the reservoir with the same clipping threshold $\kappa = 3$ would match that of the conventional or ring-based ESN. This network is denoted as "intESN-large." Since $\kappa = 3$ requires only 3 bit, in order to get the memory footprint corresponding to ESN, intESN could use more than ten times more neurons. Thus, the memory footprint of intESN with 1000 neurons corresponds to ESN with 100 neurons, whereas intESN with 2000 and 3000 neurons correspond to ESN with 200 and 300 neurons, respectively. The results for this case are presented in Fig. 4 (the training sequence was prolonged to 9000 random tokens). With such settings, intESN-large has clearly higher information capacity. In particular, for ESN memory footprint with 100 neurons, the decoded amount of information has increased 2.2 times, whereas for 200 and 300 neurons, it increased 1.6 and 1.3 times, respectively. It is important to note, however, that while the memory consumed by the reservoir of intESN-large was comparable to the corresponding ESN, the readout matrix for intESN-large was larger and more computationally demanding than the ESN readout matrix since the size of a readout matrix is proportional to the number of neurons in the reservoir. In order to consider the size of the readout matrix, we have to include in the memory footprint both the reservoir and the readout sizes. In this case, the memory footprint of ESN is

$$r N_{\mathrm{ESN}} + r N_{\mathrm{ESN}} L = r N_{\mathrm{ESN}}(L+1) \tag{6}$$

where r denotes the resolution of a neuron/weight in ESN (e.g., 32-bit) and N_{ESN} is the size of ESN's reservoir. The memory footprint of intESN is

$$N_{\mathrm{intESN}}(\lceil \log_2(2\kappa+1) \rceil + rL) \tag{7}$$

where N_{intESN} is the size of intESN's reservoir. Thus, using the footprints above, we could obtain an equation describing the ratio $N_{\mathrm{intESN}}/N_{\mathrm{ESN}}$ characterizing the size of intESN's reservoir via the size of ESN's reservoir in the case when both networks have the same memory footprint (reservoir plus readout)

$$\frac{N_{\mathrm{intESN}}}{N_{\mathrm{ESN}}} = \frac{r(L+1)}{\lceil \log_2(2\kappa+1) \rceil + rL}. \tag{8}$$

Assuming the standard value of r being 32-bit, the ratio depends on L and κ and has its largest value when L and κ are small. For example, when $\kappa = 3$ and there is only one output neuron ($L = 1$), intESN could have in its reservoir 1.83 more neurons than ESN for the same memory footprint when $L = 10$ this value decreases to 1.09.

2) Storage of Analog Values in intESN: This section presents the feasibility of storing analog values in intESN using image patches as a showcase. It is important to emphasize that this section does not go into detailed comparisons with other methods as the main purpose here is the principal demonstration of the possibilities of storing continuous data in reservoirs consisting of integers in a limited range. In other words, with this showcase, we are aiming at demonstrating the feasibility of using integer approximation of neuron states in intESN to work with analog representations. A value of a pixel (in an RGB channel) can be treated as an analog value in the range between 0 and 1. For each pixel, it is possible to generate a unique bipolar HD vector. The typical approach to encode an analog value is to multiply all elements of the HD vector by that value. The whole sequence is then represented using the bundling operation on all scaled HD vectors. The result of bundling can be used as an input to a reservoir. However, the resultant composite HD vector will not be in the integer range anymore. We address this problem by using sparsity. Instead of scaling elements of an HD vector, we propose to randomly set the fraction of elements of the HD vector to zeros, i.e., the HD vector will become ternary. The proportion of zero elements is determined by the pixel's analog value. Pixels with values close to zero will have very sparse HD vectors, whereas pixels with values close to one will have dense HD vectors, but all entries will always be $[-1, 0, \text{or } +1]$. The result of bundling of such HD vectors (i.e., HD vector for an image) will still have integer values. Such representational scheme allows keeping integer values in the reservoir, but it still can effectively store analog values.

The examples of results are presented in Fig. 5. The top row depicts original images stored in the reservoir. The other rows

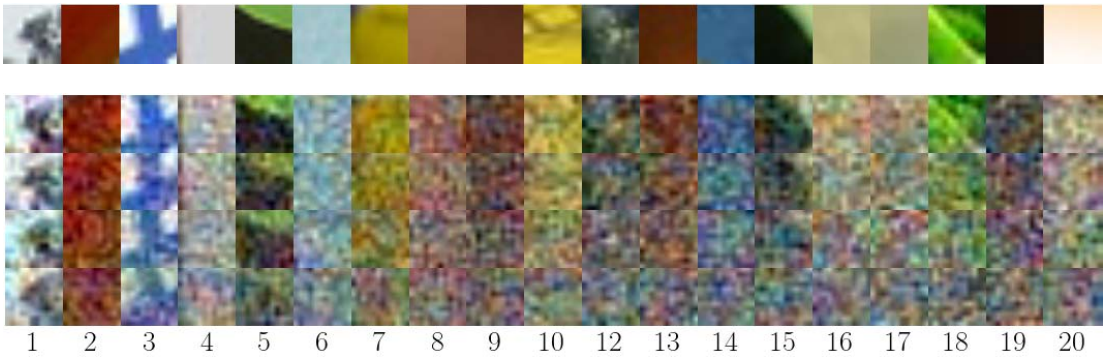


Fig. 5. Example of image patches decoded from an intESN. Top row represents the original images stored in the reservoir. Other rows depict the patches reconstructed from intESN for different reservoir sizes and clipping thresholds.

depict images reconstructed from the reservoir. The following parameters of intESN were used (top to bottom): $N = 64000$, $\kappa = 11$; $N = 32000$, $\kappa = 8$; $N = 16000$, $\kappa = 6$; and $N = 8000$, $\kappa = 4$. The values of κ were optimized for a particular N. Columns correspond to the delay values (i.e., how many steps ago an image was stored in the reservoir) as in the previous experiment. As one would anticipate, the quality of the reconstructed images is improving for larger reservoir sizes. At the same time, the quality of the reconstructed images is deteriorating for larger delay values, i.e., the worst quality of the reconstructed image could be observed in the bottom-right corner, while the best reconstruction is located in the top-left corner. Nevertheless, the main observation for this experiment is that it is possible to project analog values into the reservoir with integer values using the mapping via varying sparsity and then retrieve the values from the reservoir. Moreover, we have shown recently [65] that the mapping via varying sparsity could even be helpful when solving classification problems with a feedforward variant of the ESN.

B. Classification of Time Series

In this section, ESN (conventional and ring-based) and intESN networks are compared in terms of classification accuracy obtained on standard time-series data sets. Following [25], we used several (four) univariate data sets from UCR[3] [66] and several (three) multivariate data sets from UCI[4] [67]. Details of data sets are presented in Table I. For each data set, the table includes the name, number of variables (#V), number of classes (#C), and the number of examples in training and testing data sets.

Configurations of the networks were kept fixed for all data sets. In fact, the configuration of the conventional and ring-based ESNs was set in accordance to [25]: reservoir size was set to $N = 800$, projection gain was set to $\beta = 0.25$, and the feedback strength was set to $\rho = 0.99$. The regularization parameter for the ridge regression was set to $\lambda = 1.0$. The intESN was also trained with the same λ. The clipping threshold for the intESN was set to $\kappa = 7$. Also, for intESN,

TABLE I
DETAILS OF DATA SETS FOR TIME-SERIES CLASSIFICATION

Univariate datasets from UCR				
Name	***#V***	**Train**	**Test**	***#C***
Swedish Leaf	1	500	625	15
Distal Phalanx	1	139	400	3
ECG	1	100	100	2
Wafer	1	1000	6164	2
Multivariate datasets from UCI				
Character Trajectories	3	300	2558	20
Spoken Arabic Digit	13	6600	2200	10
Japanese Vowels	12	270	370	9

the quantized values of time series were mapped to bipolar vectors using scatter codes [63], [68]. The input signal $\mathbf{u}$(n) was quantized as

$$\mathbf{u}(n)_q = \lfloor 200\mathbf{u}(n) \rceil / 200 \tag{9}$$

where $\lfloor * \rceil$ denotes rounding to the closest integer. Two sizes of intESN's reservoir were used. The first size corresponded to the size of the conventional and ring-based ESNs, i.e., $N = 800$. The second size ("intESN-large") corresponded to the same memory footprint[5] required for ESN reservoir assuming that one ESN neuron requires 32 bit, while one intESN neuron requires 4 bit (when $\kappa = 7$). Thus, "intESN-large" had $N = 6400$ neurons.

The output layers of the networks were representing one-hot encodings of classes in a data set, i.e., for the particular data set L =#C of that data set. The readout layers of all networks were trained using time series from a training data set in the so-called endpoints mode [24] when only final temporal reservoir states for each time series are used for training a single readout matrix.

The experimental accuracies obtained from the networks for the considered data sets are presented in Figs. 6 and 7. Fig. 6 shows the results for univariate data sets, whereas

[3]UCR. Time Series Classification Archive [online], 2020.—Available online: https://www.cs.ucr.edu/%7Eeamonn/time_series_data_2018/.

[4]UCI. Machine Learning Repository [online], 2020.—Available online: https://archive.ics.uci.edu/ml/index.php.

[5]Except for the Japanese Vowels data set where such reservoir size seemed to significantly overfit the training data. In that case, the number of neurons was increased twice.

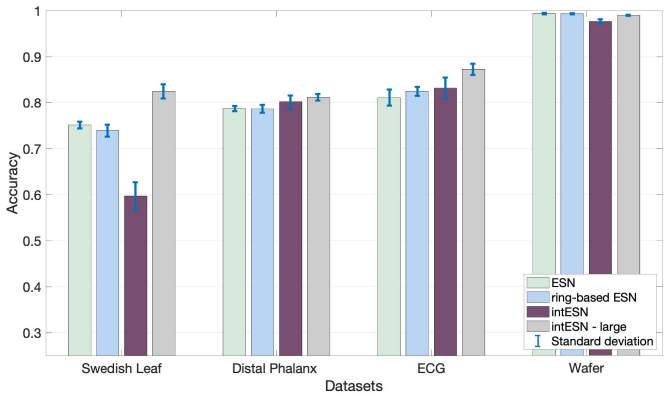


Fig. 6. Classification accuracy for univariate data sets from UCR. Bars depict mean values, and lines depict standard deviations. Bars denoted as "ESN" and "intESN" had the same number of neurons in their reservoirs, whereas for "intESN-large," the number of neurons corresponded to ESN's memory footprint.

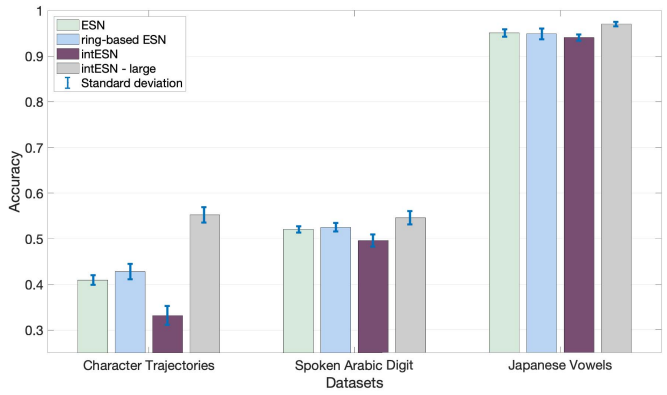


Fig. 7. Classification accuracy for multivariate data sets from UCI. Bars depict mean values, and lines depict standard deviations. Bars denoted as "ESN" and "intESN" had the same number of neurons in their reservoirs, whereas for "intESN-large," the number of neurons corresponded to ESN's memory footprint.

Fig. 7 shows the results for multivariate data sets. The figures depict mean and standard deviation values across ten independent random initializations of the networks. Similar to Section IV-A1, the accuracy of the conventional ESN and the ring-based ESN is almost identical, and thus, for the rest of this section, we assume both of these networks when using the term ESN.

The obtained results strongly depend on the characteristics of the data. However, it was generally observed that intESN with the memory footprint equivalent to ESN demonstrated higher classification accuracy. On the other hand, the classification accuracy of intESN with the same number of neurons as in ESN was similar to ESN's performance for all considered data sets but two ("Swedish Leaf" and "Character Trajectories") for which the accuracy degradation was sensible. We, therefore, conjecture that in a general case, one cannot guarantee the same classification accuracy as for ESN. The empirical evidence, however, shows that it is not infeasible. Since placing the reported results into the general context of time-series classifications is outside the scope of this article, we do not further elaborate on fine-tuning of hyperparameters of intESN for the best classification performance. However,

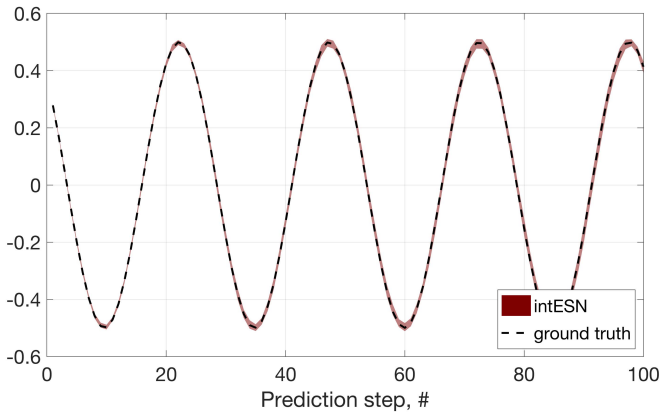


Fig. 8. Generation of a sinusoidal signal.

the interested readers are kindly referred to the Supplementary Materials (Fig. S.2 and S.3) where several different values of N, κ, and ρ were examined for each data set.

C. Modeling of Dynamic Processes

1) Learning Sinusoidal Function: The task of learning a sinusoidal function [69] is an example of a learning simple dynamic system with the constant cyclic behavior. The ground truth signal was generated as follows:

$$y(n) = 0.5 \sin(n/4). \tag{10}$$

In this task, the input layer was not used, i.e., $K = 0$, but the network projected the activations of the output layer back to the reservoir using $\mathbf{W}^{\text{back}}$. The output layer had only one neuron ($L = 1$). The reservoir size was fixed to $N = 1000$ neurons. The length of the training sequence was 3000 (first 1000 steps were discarded from the calculation). For ESN, the feedback strength for the reservoir connection matrix was set to $\rho = 0.8$, and for both networks, λ was set to 0. A continuous value of the ground-truth signal was fed-in to ESN during the training.

For intESN, in order to map the input signal to a bipolar vector, the quantization was used. The signal was quantized as

$$y(n)_q = \lfloor 100 y(n) \rceil / 100. \tag{11}$$

The item memory for the projection of the output layer was populated with bipolar vectors preserving linear (in terms of dot product) similarity between quantization levels [62]. The threshold for the clipping function was set to $\kappa = 3$.

In the operating phase, the network acted as the generator of the signal feeding its previous prediction (at time $n - 1$) back to the reservoir. Fig. 8 shows the behavior of intESN during the first 100 prediction steps. The ground truth is depicted by the dashed line, while the prediction of intESN is illustrated by the shaded area between 10% and 90% percentiles (100 simulations were performed). The figure does not show the performance of the conventional ESN as it just followed the ground truth without visible deviations. intESN clearly follows the values of the ground truth, but the deviation from the ground truth is increasing with the number of prediction

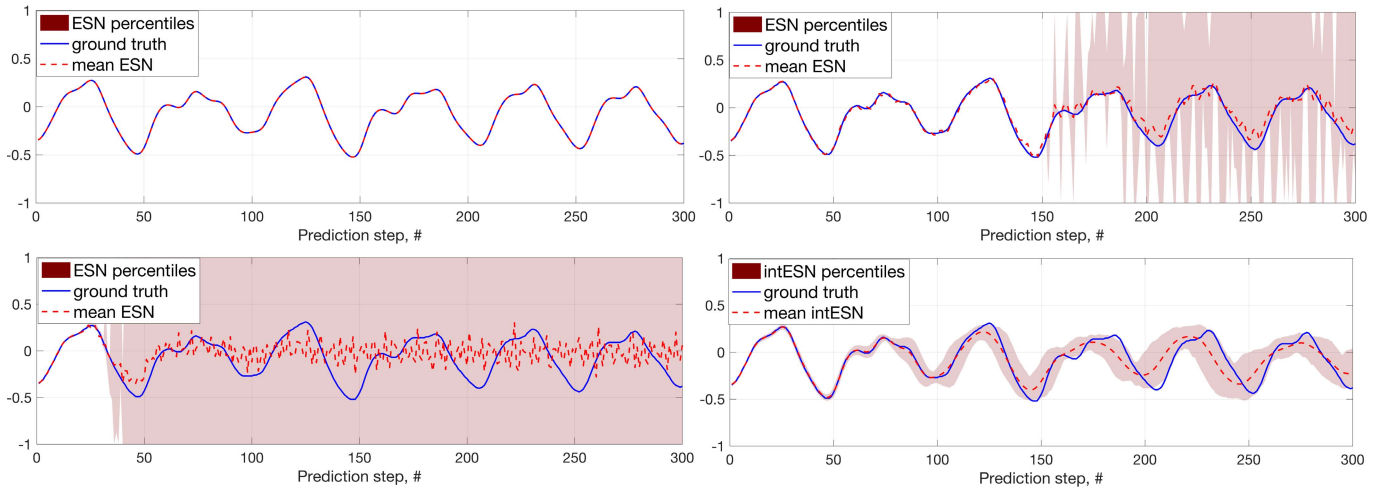


Fig. 9. Prediction of the Mackey–Glass series.

steps. It is unavoidable for the increasing prediction horizon, but, in this scenario, it is additionally accelerated due to the presence of the quantization error at each prediction step. It is worth noting, however, that the quality of predictions for this task could be improved by increasing the value of $\kappa = 3$, i.e., at the cost of extract memory allocated for each neuron. The diligent readers are kindly referred to the Supplementary Materials (Fig. S.4) where several different values of κ were examined. Section IV-C2 will clearly demonstrate the effects caused by the quantization process. The error is accumulated because, every time when feeding the prediction back to the reservoir of intESN, it should be quantized in order to fetch a vector from the item memory.

2) Mackey–Glass Series Prediction: A Mackey–Glass series is generated by the nonlinear time-delay differential equation. It is commonly used to assess the predictive power of an RC approach. In this scenario, we followed the preprocessing of data and the parameters of ESN described in [4]. The parameters of intESN (including quantization scheme) were the same as in Section IV-C1. The interested readers are kindly referred to the Supplementary Materials (Fig. S.5) where several different values of N and κ were examined. The length of the training sequence was 3000 (the first 1000 steps were discarded from the calculation). Fig. 9 shows the results for the first 300 prediction steps. The results were calculated from 100 simulation runs. The figure includes four panels. Each panel depicts the ground truth, the mean value of predictions, as well as areas marking percentiles between 10% and 90%. The lower right corner corresponds to intESN, whereas three other panels show performance of ESN in three different cases related to the quantization of the data.

In these scenarios, ESN was trained to learn the model from the quantized data in order to see to which extent it affects the network. The upper left corner corresponds to ESN without data quantization. In this case, the predictions precisely follow the ground truth. The upper right corner corresponds to ESN trained on the quantized data but with no quantization during the operational phase. In such settings, the network closely follows the ground truth for the first 150 steps, but then it often explodes. The lower left corner corresponds to ESN where the data were quantized during both training and prediction. In this scenario, the network was able produce to produce good prediction just for the first few dozens of steps and then entered the chaotic mode where even the mean value does not reflect the ground truth. These cases demonstrate how the quantization error could affect the predictions, especially when it is added at each prediction step. Note that intESN operated in the same mode as the third ESN. Despite this fact, its performance rather resembles that of the second ESN where the speed deviation of the ground truth is faster. At the same time, the deviation of intESN grows smoothly without a sudden explosion in contrast to ESN.

V. Digital Hardware Experiments

In order to demonstrate the amenability of intESN for digital hardware, we used an FPGA and implemented three different architectures: software ESN as well as hardware accelerated ESN and intESN. An alternative FPGA implementation has been recently proposed in [70]. All architectures were implemented on a ZedBoard FPGA,[6] which contains a dual core ARM Cortex A9 CPU interfacing with a programmable logic fabric. The Xilinx Vivado Design Suite[7] and Vivado SDK[8] were used to design the hardware architectures and program the FPGA board, respectively. The efficiency evaluation (e.g., energy consumption) of the architectures was based on the recall stage of the sequence recall task as described in Section IV-A1.

A. Architectures Design

The software ESN was implemented using only CPU on the FPGA board. For the hardware acceleration experiments,

[6]ZedBoard. Hardware User's Guide [online], 2014.—Available online: http://zedboard.org/sites/default/files/documentations/ZedBoard_HW_UG_v2_2.pdf.

[7]Xilinx. Vivado Design Suite User Guide [online], 2018.—Available online: https://www.xilinx.com/support/documentation/sw_manuals/xilinx2018_1/ug910-vivado-getting-started.pdf.

[8]Xilinx. Generating Basic Software Platforms [online], 2018.—Available online: https://www.xilinx.com/support/documentation/sw_manuals/xilinx2018_3/ug1138-generating-basic-software-platforms.pdf.

we programmed the CPU to generate the inputs and feed them into a hardware architecture (either for ESN or intESN) and then retrieve the outputs once the computation is over. The hardware ESN and intESN architectures were designed using Vivado High Level Synthesis (HLS) using the C programming language and later on synthesized into hardware intellectual property components to be integrated in a larger hardware system comprised of the Zynq Processing System, AXI Direct Memory Access, AXI Interconnect, ESN/intESN architecture, and various other peripherals used for clocking and resetting mechanisms. It is important to note that for resource limitations on the ZedBoard, no pipelining or hardware optimization directives have been used on the HLS designs in order to cope with the resource usage for growing reservoir sizes and remain within the board's capacity. However, we still provide a comparison of a speedup expected from the pipelining for the two hardware architectures.

B. Evaluation Methodology

For each architecture, the reservoir size was varied between 100, 200, and 300 neurons. In the following, we report the number of clock cycles required to accomplish the sequence recall task, the power consumption, and the area utilization (i.e., resources) required for the hardware designs. The number of clock cycles was measured using a hardware timer incorporated into the designs. The area utilization is reported from the hardware synthesis reports provided by Vivado. All three architectures use the same clock frequency of 100 MHz. The ZedBoard contains a 10-mΩ shunt resistor in series with the input supply of the whole board, which could be used to obtain the overall power consumption by measuring the voltage across it. However, at the desired scale of changes, voltage fluctuation and measurement sensitivity made it almost impossible to properly perform a precise comparison. Therefore, instead, we used the Xilinx Power Estimator (XPE) tool[9] provided by the Vivado Design Suite to estimate the power consumption of each architecture, which is a standard option [71]–[73].

C. Efficiency Evaluation Results

Table II presents the area utilization of the hardware architectures for different sizes of reservoir. It is clear that the resource utilization of the hardware ESN is always larger than that of hardware intESN. This is an empirical manifestation of the facts that intESN: 1) requires a lower memory footprint ($\kappa = 3$) and 2) its machinery uses simpler operations, e.g., the clipping instead of tanh() and the cyclic shift instead of vector-matrix multiplication. It is important to note that the drastic increase of LUT utilization when the reservoir size of ESN was set to 300 is due to resource constraints on the FPGA board since the total number of usable BRAM units is 140. Thus, the increased number of LUTs was used as an alternative way of increasing the memory capacity of the board.

[9]Xilinx. Power Estimator User Guide [online], 2018.—Available online: https://www.xilinx.com/support/documentation/sw_manuals/xilinx2018_3/ug440-xilinx-power-estimator.pdf.

TABLE II
AREA UTILIZATION OF THE HARDWARE ARCHITECTURES

N	**LUT**	**FF**	**BRAM**	**DSP48**
ESN				
100	8317	8724	31.5	47
200	8368	8770	87.5	47
300	15950	8920	135.5	47
intESN				
100	4577	5488	7.5	5
200	4602	5492	11.5	6
300	4671	5497	12.5	6

TABLE III
NUMBER OF CLOCK CYCLES (TIME) INVOLVED IN THE SEQUENCE RECALL TASK

N	**software ESN**	**hardware ESN**	**hardware intESN**
100	3.0×10^8 (3.04 s)	5.8×10^8 (5.77 s)	1.4×10^8 (1.42 s)
200	1.0×10^9 (9.99 s)	1.9×10^9 (18.74 s)	2.8×10^8 (2.82 s)
300	2.1×10^9 (21.16 s)	3.9×10^9 (38.90 s)	4.5×10^8 (4.46 s)

Table II presents the number of clock cycles (time in seconds) necessary to perform the operating phase of the sequence recall task for each architecture. The number of clock cycles was measured with the help of the hardware timer. The time was calculated using the known frequency of the clock (100 MHz) As expected, for each architecture, the operation time increases with the increased reservoir size. However, for any reservoir size, intESN is several times faster than both implementations of ESN (at least 2.1 against the software ESN when $N = 100$). The gain is increasing with the increased reservoir size so that when $N = 300$, the hardware intESN is 8.7 times faster than the hardware ESN.

An interesting remark in Table III is that the hardware ESN seems slower than the software ESN. It is counterintuitive since the hardware architecture is expected to accelerate the computations. In the considered case, this fact is explained by the absence of pipelining (see Section V-A) what prevents further hardware optimizations. The pipelining would certainly increase the speed at the price of the increased area utilization, which would make the hardware designs being inconceivable on the target board. In order to evaluate the effect of the pipelining on the speed of the hardware architectures, we have used the Vivado HLS synthesis report to estimate the number of clock cycles per iteration when $N = 300$. The results are presented in Table IV. It shows that the pipelining significantly decreases the number of clock cycles, which makes the hardware ESN much faster than the software one. At the same time, the speedup obtained for the hardware intESN is still higher than that of the hardware ESN, which makes intESN even more efficient than ESN.

Table V presents the power consumption of each architecture. The first observation is that the consumption of the software ESN remains constant for different N. This is because the hardware itself remains fixed, and the software computations that are performed on the Cortex CPU are the

TABLE IV
SPEEDUP COMPARISON WITH THE USE OF PIPELINING

	ESN		intESN	
Pipelining	✓	✗	✓	✗
Clock cycles	1,297,072	51,948	148,241	2,856
Speed-up	24.97		51.91	

TABLE V
XPE OVERALL POWER CONSUMPTION (WATTS)

N	software ESN	hardware ESN	hardware intESN
100	1.53	1.73	1.59
200	1.53	1.78	1.60
300	1.53	1.95	1.60

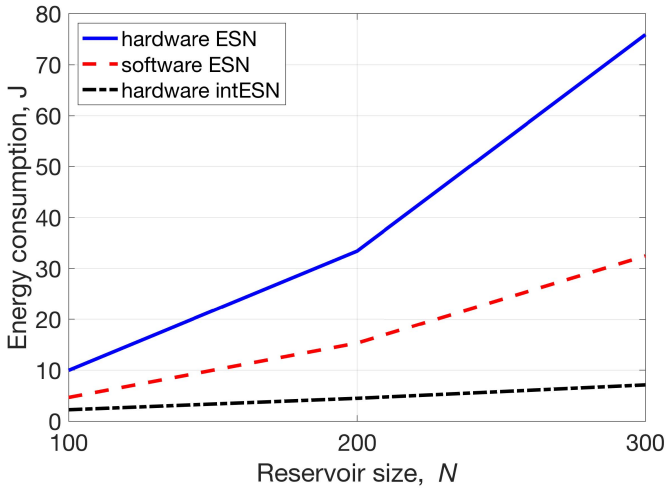


Fig. 10. Overall energy consumption of all three architectures against the reservoir size.

same for each configuration. Moreover, the software ESN has the lowest power consumption because it is only comprised of the CPU and the hardware timer, whereas the other two include other hardware components in addition to those. However, for the hardware implementation of ESN, one could see a noticeable increase for a larger reservoir size. There is also an increase for the hardware intESN, but it is slower than the hardware ESN. Moreover, for the fixed reservoir size, intESN's power consumption remains lower than that of ESN.

Fig. 10 shows the overall energy consumption of each architecture for different sizes of reservoir. The energy consumption was calculated as a product of the overall power consumption reported in Table V and the operating time reported in Table III. Fig. 10 clearly shows that the energy efficiency of the hardware intESN is much better than that of both the hardware and software ESNs. Scaling the reservoir size leads to an increase in the energy consumption of all architectures. However, the slope of the curve for intESN is lower than for ESN's architectures. Therefore, the energy saved by the use of intESN drastically increases with increasing size of reservoir (e.g., for $N = 300$, it needs 10.6 times lower energy than for the hardware ESN), which is a strong argument in favor of intESN.

VI. DISCUSSION

A. Efficiency of intESN

In principle, evaluation of the computational efficiency of the proposed intESN could be done even in a simplistic manner using only a computer. For example, we have performed the initial assessment by simplified execution time measurements with MATLAB. We used our MATLAB implementations of both networks for the trajectory association task (see Section IV-A1) with $N = 300$ neurons to compare the times of projecting data and executing a reservoir. ESN was implemented using 32-bit float type (type single in MATLAB), whereas intESN was implemented using 8-bits integer type (type *int8* in MATLAB). On average, the time required by intESN was 3.9 times less than that of ESN. Thus, even the training time needed for convergence with intESN is shorter than that of ESN because intESN is much faster when it comes to projecting data and executing a reservoir, while the time for estimating the readout matrix would be the same for both networks.

Section V presented the proper evaluation of the computational efficiency of the proposed intESN approach and the conventional ESN using the FPGA board. The hardware implementation of intESN was compared to two reference implementations: software ESN and hardware ESN. The detailed benchmarking tests have supported our claims about the efficiency of intESN. It is also worth noting that the efficiency gains of the intESN would be less relative to the ring-based ESN as essentially both networks use the same efficient mechanism for reservoir's connectivity.

B. Hyperparameters

In comparison to ESN, intESN has fewer hyperparameters. The common hyperparameter is the reservoir size N. Two specific intESN hyperparameters are the clipping threshold κ and the mapping to the reservoir, whereas for ESN, we have to choose ρ, β, and α. When N is fixed, then κ in intESN has an effect on the network's memory similar to ρ and β in ESN (see Section VI-C for details). However, the difference is that κ takes only positive integer values. This has its pros and cons. It could be much easier to optimize a single hyperparameter in the integer range. On the other hand, having real-valued hyperparameters, one can get a configuration providing much finer tuning of network's memory for a given task.

With respect to the mapping (projection) to the reservoir, it is probably the most nontrivial part on intESN, especially, when data to be projected are real numbers. In this article, we mention three different strategies for mapping real numbers to bipolar or ternary vectors: puncturing of nonzero elements (see Section IV-A2), mapping preserving linear similarity between vectors [62], and nonlinear approximate mapping using scatter codes [63], [68]. We suggest that it is a useful heuristic to try each of the approaches and choose the one performing best.

C. On Equivalence of ESN and intESN in Terms of Forgetting Time Constants

Section IV-A1 presented the experimental comparison of storage capabilities of intESN and ESN. An analytical

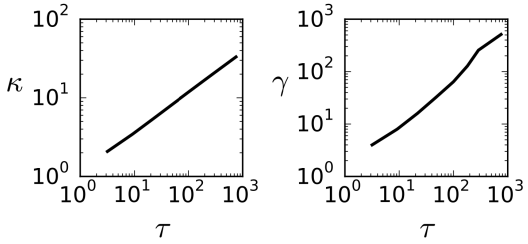


Fig. 11. Correspondence between time constants of intESN and special cases of ESN.

approach to the treatment of memory capacity of reservoir was presented in [10, Sec. 2.4.4]. The work introduced an analytical tool called the forgetting time constant (denoted as τ). The forgetting time constant is a scalar characterizing the memory decay in a network. In the case of intESN, it can be calculated analytically using the clipping threshold κ. For ESN, currently, only special cases can be analyzed. For example, when reservoir neurons are linear, the feedback strength ρ will determine τ. The other example is when reservoir neurons use tanh, the feedback strength is fixed to one, and the reservoir update rule is modified to $f(\mathbf{x}) = \gamma \tanh(\mathbf{x}/\gamma)$, where γ denotes a gain parameter. This parameter affects τ, which could be calculated numerically. Fig. 11 shows the implicit comparison of this special case of ESN and intESN via their forgetting time constants that are determined by γ and κ, respectively. Both parameters similarly (not far from linear in logarithmic coordinates) affect τ. It allows arguing that the networks are close to being functionally identical in terms of storage capabilities. The development of an approach for estimating the forgetting time constant for ESN in a general case is a part of our agenda for the future work.

D. Training the Readout in a Generator Mode

In the experiments generating time series, we used the teacher forcing approach for training a readout matrix. This, however, does not have to be the compulsory choice for intESN. We do not foresee any complications for applying other approaches allowing to modify the network's behavior for producing complex target functions. In particular, the FORCE method [2], which uses error-based modification of readout weights during the training process, can be used as it has a mode in which only weights of a readout matrix are changed while leaving the rest of the network fixed.

VII. Conclusion

In this article, we proposed an architecture for integer approximation of the RC, which is based on the mathematics of hyperdimensional computing. The neurons in reservoir are described by integers in the limited range, and the update operations include only addition, permutation (cyclic shift), and clipping. Therefore, the intESN has substantially smaller memory footprint and higher computational efficiency compared to the conventional echo state network with the same number of neurons in the reservoir. The actual number of bits for representing a neuron depends on the clipping threshold κ but can be significantly lower than 32-bit floats in ESN. For example, in our experiments, the results were obtained with $\kappa = 3$ and $\kappa = 7$, which effectively makes it sufficient to represent a neuron with only three or four bits, respectively. We demonstrated that the performance of the intESN is comparable to the conventional ESN in terms of memory capacity, potential capabilities for classification of time series, and modeling dynamic systems. The better performance was observed when the memory footprint of reservoir of the intESN was set to that of the conventional ESN. The experiment on the digital hardware has validated the amenability of the intESN for significantly improving the energy efficiency of computations. Further improvements can be made by optimization of the parameters and better quantization schemes for handling continuous values. Naturally, due to the peculiarity of input data projection into the intESN, the performance of the network in tasks for modeling dynamic systems is to a certain degree lower than that of the conventional ESN. This, however, does not undermine the importance of intESNs, which are extremely attractive for memory and power savings, and in the general area of approximate computing, where errors and approximations are becoming acceptable as long as the outcomes have a well-defined statistical behavior.

References

[1] M. Lukoševičius and H. Jaeger, "Reservoir computing approaches to recurrent neural network training," *Comput. Sci. Rev.*, vol. 3, no. 3, pp. 127–149, Aug. 2009.

[2] D. Sussillo and L. F. Abbott, "Generating coherent patterns of activity from chaotic neural networks," *Neuron*, vol. 63, no. 4, pp. 544–557, Aug. 2009.

[3] F. Triefenbach, A. Jalalvand, B. Schrauwen, and J.-P. Martens, "Phoneme recognition with large hierarchical reservoirs," in *Proc. Adv. Neural Inf. Process. Syst. (NIPS)*, 2010, pp. 2307–2315.

[4] H. Jaeger, "Harnessing nonlinearity: Predicting chaotic systems and saving energy in wireless communication," *Science*, vol. 304, no. 5667, pp. 78–80, Apr. 2004.

[5] L. Appeltant *et al.*, "Information processing using a single dynamical node as complex system," *Nature Commun.*, vol. 2, no. 1, p. 468, Sep. 2011.

[6] J. Pathak, B. Hunt, M. Girvan, Z. Lu, and E. Ott, "Model-free prediction of large spatiotemporally chaotic systems from data: A reservoir computing approach," *Phys. Rev. Lett.*, vol. 120, no. 2, Jan. 2018, Art. no. 024102.

[7] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "XNOR-Net: Imagenet classification using binary convolutional neural networks," in *Proc. Eur. Conf. Comput. Vis.* (Lecture Notes in Computer Science), vol. 9908. Cham, Switzerland: Springer, 2016, pp. 525–542.

[8] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized neural networks," in *Proc. Adv. Neural Inf. Process. Syst. (NIPS)*, 2016, pp. 1–9.

[9] I. Hubara, M. Courbariaux, and D. Soudry, "Quantized neural networks: Training neural networks with low precision weights and activations," *J. Mach. Learn. Res.*, vol. 18, no. 1, pp. 6869–6898, 2018.

[10] E. P. Frady, D. Kleyko, and F. T. Sommer, "A theory of sequence indexing and working memory in recurrent neural networks," *Neural Comput.*, vol. 30, no. 6, pp. 1449–1513, Jun. 2018.

[11] P. Kanerva, "Hyperdimensional computing: An introduction to computing in distributed representation with high-dimensional random vectors," *Cognit. Comput.*, vol. 1, no. 2, pp. 139–159, Jun. 2009.

[12] R. Gayler, "Vector symbolic architectures answer Jackendoff's challenges for cognitive neuroscience," in *Proc. Joint Int. Conf. Cognit. Sci. (ICCS/ASCS)*, 2003, pp. 133–138.

[13] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," *IEEE Trans. Neural Netw.*, vol. 5, no. 2, pp. 157–166, Mar. 1994.

[14] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, 1997.

[15] W. Maass, T. Natschläger, and H. Markram, "Real-time computing without stable states: A new framework for neural computation based on perturbations," *Neural Comput.*, vol. 14, no. 11, pp. 2531–2560, Nov. 2002.

[16] H. Jaeger, "Adaptive nonlinear system identification with echo state networks," in *Proc. Adv. Neural Inf. Process. Syst. (NIPS)*, 2003, pp. 593–600.

[17] B. Igelnik and Y.-H. Pao, "Stochastic choice of basis functions in adaptive function approximation and the functional-link net," *IEEE Trans. Neural Netw.*, vol. 6, no. 6, pp. 1320–1329, Nov. 1995.

[18] G.-B. Huang, Q.-Y. Zhu, and C.-K. Siew, "Extreme learning machine: Theory and applications," *Neurocomputing*, vol. 70, nos. 1–3, pp. 489–501, Dec. 2006.

[19] G. Huang, "What are extreme learning machines? Filling the gap between Frank Rosenblatt's dream and John von Neumann's puzzle," *Cogn. Comput.*, vol. 7, pp. 263–278, May 2015.

[20] H. Jaeger, "The 'echo state' approach to analysing and training recurrent neural networks-with an erratum note," German Nat. Res. Center Inf. Technol., Sankt Augustin, Germany, Tech. Rep. 148, 2001.

[21] S. J. Weddell and R. Y. Webb, "Reservoir computing for prediction of the spatially-variant point spread function," *IEEE J. Sel. Topics Signal Process.*, vol. 2, no. 5, pp. 624–634, Oct. 2008.

[22] P. Antonik, M. Haelterman, and S. Massar, "Brain-inspired photonic signal processor for generating periodic patterns and emulating chaotic systems," *Phys. Rev. A, Gen. Phys.*, vol. 7, no. 5, pp. 1–16, May 2017.

[23] E. M. Forney, C. W. Anderson, W. J. Gavin, P. L. Davies, M. C. Roll, and B. K. Taylor, "Echo state networks for modeling and classification of EEG signals in mental-task brain-computer interfaces," Colorado State Univ., Fort Collins, CO, USA, Tech. Rep. CS-15-102, Nov. 2015.

[24] A. A. Prater, "Comparison of echo state network output layer classification methods on noisy data," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, May 2017, pp. 2644–2651.

[25] F. M. Bianchi, S. Scardapane, S. Løkse, and R. Jenssen, "Reservoir computing approaches for representation and classification of multivariate time series," *IEEE Trans. Neural Netw. Learn. Syst.*, early access, Jun. 29, 2020, doi: 10.1109/TNNLS.2020.3001377.

[26] O. Yilmaz, "Machine learning using cellular automata based feature expansion and reservoir computing," *J. Cellular Automata*, vol. 10, nos. 5–6, pp. 435–472, 2015.

[27] D. Kleyko, S. Khan, E. Osipov, and S.-P. Yong, "Modality classification of medical images with distributed representations based on cellular automata reservoir computing," in *Proc. IEEE 14th Int. Symp. Biomed. Imag. (ISBI)*, Apr. 2017, pp. 1–4.

[28] N. Karvonen, J. Nilsson, D. Kleyko, and L. L. Jimenez, "Low-power classification using FPGA—An approach based on cellular automata, neural networks, and hyperdimensional computing," in *Proc. 18th IEEE Int. Conf. Mach. Learn. Appl. (ICMLA)*, Dec. 2019, pp. 370–375.

[29] M. Schmuck, L. Benini, and A. Rahimi, "Hardware optimizations of dense binary hyperdimensional computing: Rematerialization of hypervectors, binarized bundling, and combinational associative memory," *ACM J. Emerg. Technol. Comput. Syst.*, vol. 15, no. 4, pp. 1–25, Dec. 2019.

[30] D. Kleyko, E. Paxon Frady, and F. T. Sommer, "Cellular automata can reduce memory requirements of collective-state computing," 2020, *arXiv:2010.03585*. [Online]. Available: http://arxiv.org/abs/2010.03585

[31] O. Yilmaz, "Symbolic computation using cellular automata-based hyperdimensional computing," *Neural Comput.*, vol. 27, no. 12, pp. 2661–2692, Dec. 2015.

[32] S. Nichele, Oslo, A. University College of Applied Sciences, A. Molund, N. University of Science, and Technology, "Deep learning with cellular automaton-based reservoir computing," *Complex Syst.*, vol. 26, no. 4, pp. 319–340, Dec. 2017.

[33] N. McDonald, "Reservoir computing & extreme learning machines using pairs of cellular automata rules," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, May 2017, pp. 2429–2436.

[34] C. Gallicchio, A. Micheli, and L. Pedrelli, "Deep reservoir computing: A critical experimental analysis," *Neurocomputing*, vol. 268, pp. 87–99, Dec. 2017.

[35] C. Gallicchio, A. Micheli, and L. Pedrelli, "Design of deep echo state networks," *Neural Netw.*, vol. 108, pp. 33–47, Dec. 2018.

[36] M. Han and M. Xu, "Laplacian echo state network for multivariate time series prediction," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 29, no. 1, pp. 238–244, Jan. 2018.

[37] J. Qiao, F. Li, H. Han, and W. Li, "Growing echo-state network with multiple subreservoirs," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 28, no. 2, pp. 391–404, Feb. 2017.

[38] F. M. Bianchi, L. Livi, and C. Alippi, "Investigating echo-state networks dynamics by means of recurrence analysis," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 29, no. 2, pp. 427–439, Feb. 2018.

[39] L. Livi, F. M. Bianchi, and C. Alippi, "Determination of the edge of criticality in echo state networks through Fisher information maximization," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 29, no. 3, pp. 706–717, Mar. 2018.

[40] M. C. Ozturk, D. Xu, and J. C. Príncipe, "Analysis and design of echo state networks," *Neural Comput.*, vol. 19, no. 1, pp. 111–138, Jan. 2007.

[41] L. Büsing, B. Schrauwen, and R. Legenstein, "Connectivity, dynamics, and memory in reservoir computing with binary and analog neurons," *Neural Comput.*, vol. 22, no. 5, pp. 1272–1311, May 2010.

[42] T. Strauss, W. Wustlich, and R. Labahn, "Design strategies for weight matrices of echo state networks," *Neural Comput.*, vol. 24, no. 12, pp. 3246–3276, Dec. 2012.

[43] S. Parfitt, P. Tiňo, and G. Dorffner, "Graded grammaticality in prediction fractal machines," in *Proc. Adv. Neural Inf. Process. Syst. (NIPS)*, 2000, pp. 52–58.

[44] P. Tino, M. Cernansky, and L. Benuskova, "Markovian architectural bias of recurrent neural networks," *IEEE Trans. Neural Netw.*, vol. 15, no. 1, pp. 6–15, Jan. 2004.

[45] P. Tiňo, B. Hammer, and M. Bodén, "Markovian bias of neural-based architectures with feedback connections," in *Perspectives of Neural-Symbolic Integration* (Studies in Computational Intelligence), vol. 77. Berlin, Germany: Springer, 2007, pp. 95–133.

[46] A. Rodan and P. Tino, "Minimum complexity echo state network," *IEEE Trans. Neural Netw.*, vol. 22, no. 1, pp. 131–144, Jan. 2011.

[47] M. Chang, A. Terzis, and P. Bonnet, "Mote-based online anomaly detection using echo state networks," in *Distributed Computing in Sensor Systems (DCOSS)* (Lecture Notes in Computer Science), vol. 5516. Berlin, Germany: Springer, 2009, pp. 72–86.

[48] D. Bacciu, S. Chessa, C. Gallicchio, A. Micheli, and P. Barsocchi, "An experimental evaluation of reservoir computation for ambient assisted living," in *Proc. Neural Nets Surroundings: 22nd Italian Workshop Neural Nets* (Smart Innovation, Systems and Technologies), vol. 19. Berlin, Germany: Springer, 2013, pp. 41–50.

[49] D. Bacciu, P. Barsocchi, S. Chessa, C. Gallicchio, and A. Micheli, "An experimental characterization of reservoir computing in ambient assisted living applications," *Neural Comput. Appl.*, vol. 24, no. 6, pp. 1451–1464, May 2014.

[50] M. Lukoševičius, "A practical guide to applying echo state networks," in *Neural Networks: Tricks of the Trade* (Lecture Notes in Computer Science), vol. 7700. Berlin, Germany: Springer, 2012, pp. 659–686.

[51] T. A. Plate, "Holographic reduced representations," *IEEE Trans. Neural Netw.*, vol. 6, no. 3, pp. 623–641, May 1995.

[52] A. Rahimi *et al.*, "High-dimensional computing as a nanoscalable paradigm," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 64, no. 9, pp. 2508–2521, Sep. 2017.

[53] S. I. Gallant and T. W. Okaywe, "Representing objects, relations, and sequences," *Neural Comput.*, vol. 25, no. 8, pp. 2038–2078, Aug. 2013.

[54] R. W. Gayler, "Multiplicative binding, representation operators & analogy," in *Advances in Analogy Research: Integration of Theory and Data From the Cognitive, Computational, and Neural Sciences*, D. Gentner, K. J. Holyoak, and B. N. Kokinov, Eds. Sofia, Bulgaria: New Bulgarian Univ., 1998, pp. 1–4.

[55] S. I. Gallant and P. Culliton, "Positional binding with distributed representations," in *Proc. Int. Conf. Image, Vis. Comput. (ICIVC)*, Aug. 2016, pp. 108–113.

[56] T. A. Plate, *Holographic Reduced Representations: Distributed Representation for Cognitive Structures*. Stanford, CA, USA: Center for the Study of Language and Information (CSLI), 2003.

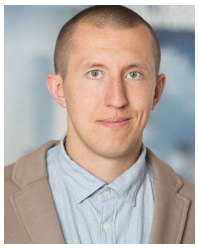
[57] D. A. Rachkovskij, "Representation and processing of structures with binary sparse distributed codes," *IEEE Trans. Knowl. Data Eng.*, vol. 13, no. 2, pp. 261–276, Apr. 2001.

[58] A. Rahimi, S. Benatti, P. Kanerva, L. Benini, and J. M. Rabaey, "Hyperdimensional biosignal processing: A case study for EMG-based hand gesture recognition," in *Proc. IEEE Int. Conf. Rebooting Comput. (ICRC)*, Oct. 2016, pp. 1–8.

[59] P. Kanerva, *Sparse Distributed Memory*. Cambridge, MA, USA: MIT Press, 1988.

[60] D. Kleyko, E. Osipov, A. Senior, A. I. Khan, and Y. A. Sekercioglu, "Holographic graph neuron: A bioinspired architecture for pattern processing," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 28, no. 6, pp. 1250–1262, Jun. 2017.

[61] D. A. Rachkovskij, S. V. Slipchenko, E. M. Kussul, and T. N. Baidyk, "Sparse binary distributed encoding of scalars," *J. Autom. Inf. Sci.*, vol. 37, no. 6, pp. 12–23, 2005.
[62] D. Widdows and T. Cohen, "Reasoning with vectors: A continuous model for fast robust inference," *Log. J. IGPL*, vol. 23, no. 2, pp. 141–173, Apr. 2015.
[63] D. Kleyko, A. Rahimi, D. A. Rachkovskij, E. Osipov, and J. M. Rabaey, "Classification and recall with binary hyperdimensional computing: Tradeoffs in choice of density and mapping characteristics," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 29, no. 12, pp. 5880–5898, Dec. 2018.
[64] D. Kleyko, E. Osipov, D. D. Silva, U. Wiklund, and D. Alahakoon, "Integer self-organizing maps for digital hardware," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, Jul. 2019, pp. 1–8.
[65] D. Kleyko, M. Kheffache, E. P. Frady, U. Wiklund, and E. Osipov, "Density encoding enables resource-efficient randomly connected neural networks," *IEEE Trans. Neural Netw. Learn. Syst.*, early access, Aug. 24, 2020, doi: 10.1109/TNNLS.2020.3015971.
[66] H. A. Dau *et al.*, "The UCR time series archive," *IEEE/CAA J. Automatica Sinica*, vol. 6, no. 6, pp. 1293–1305, Nov. 2019.
[67] D. Dua and C. Graff. (2019). *UCI Machine Learning Repository*. [Online]. Available: http://archive.ics.uci.edu/ml
[68] D. Smith and P. Stanford, "A random walk in Hamming space," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, vol. 2, Jun. 1990, pp. 465–470.
[69] H. Jaeger, "Tutorial on training recurrent neural networks, covering BPTT, RTRL, EKF and the echo state network approach," German Nat. Res. Center Inf. Technol., Sankt Augustin, Germany, Tech. Rep. 159, 2002.
[70] O. Nepomnyashchiy, A. Khantimirov, D. Galayko, and N. Sirotinina, "Method of recurrent neural network hardware implementation," in *Computer Science On-line Conference: Artificial Intelligence and Bioinspired Computational Methods (CSOC)* (Advances in Intelligent Systems and Computing), vol. 1225. Cham, Switzerland: Springer, 2020, pp. 429–437.
[71] C. Lammie, T. J. Hamilton, A. van Schaik, and M. Rahimi Azghadi, "Efficient FPGA implementations of pair and triplet-based STDP for neuromorphic architectures," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 66, no. 4, pp. 1558–1570, Apr. 2019.
[72] S. Moini, B. Alizadeh, M. Emad, and R. Ebrahimpour, "A resource-limited hardware accelerator for convolutional neural networks in embedded vision applications," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 64, no. 10, pp. 1217–1221, Oct. 2017.
[73] Y. Eminaga, A. Coskun, and I. Kale, "Area and power efficient implementation of db4 wavelet filter banks for ECG applications using reconfigurable multiplier blocks," in *Proc. 4th Int. Conf. Frontiers Signal Process. (ICFSP)*, Sep. 2018, pp. 65–68.



Denis Kleyko (Member, IEEE) received the B.S. degree (Hons.) in telecommunication systems and the M.S. degree (Hons.) in information systems from the Siberian State University of Telecommunications and Information Sciences, Novosibirsk, Russia, in 2011 and 2013, respectively, and the Ph.D. degree in computer science from the Luleå University of Technology, Luleå, Sweden, in 2018.

He is currently a Post-Doctoral Researcher on a joint appointment between the Redwood Center for Theoretical Neuroscience at the University of California at Berkeley, Berkeley, CA, USA, and the Intelligent Systems Lab, Research Institutes of Sweden, Kista, Sweden. His current research interests include artificial intelligence, machine learning, collective-state computing, reservoir computing, vector symbolic architectures, and hyperdimensional computing.

Edward Paxon Frady received the B.S. degree in computation and neural systems from California Institute of Technology, Pasadena, CA, USA, in 2008, and the Ph.D. degree in neuroscience from the University of California at San Diego, La Jolla, CA, in 2014.

He is currently a Researcher in Residence with the Neuromorphic Computing Lab, Intel Labs, Santa Clara, CA, USA, and a Visiting Scholar with the Redwood Center for Theoretical Neuroscience, University of California at Berkeley, Berkeley, CA. His research interests include neuromorphic engineering, vector symbolic architectures, hyperdimensional computing, and machine learning.

Mansour Kheffache received the B.S. degree in computer science from The American University in Cairo, New Cairo, Egypt, in 2017, and the M.S. degree in pervasive computing from the Luleå University of Technology, Luleå, Sweden, in 2019.

He is currently a Consultant with Netlight Consulting AB, Stockholm, Sweden. His current research interests include machine learning, reservoir computing, vector symbolic architectures, and hyperdimensional computing.

Evgeny Osipov (Associate Member, IEEE) received the Ph.D. degree in computer science from the University of Basel, Basel, Switzerland, in 2005.

He is currently a Full Professor in dependable communication and computation systems with the Department of Computer Science and Electrical Engineering, Luleå University of Technology, Luleå, Sweden. His research interest is in novel computational models for unconventional computer architectures. His current research focuses on hyperdimensional computing and vector symbolic architectures.