Cluster-Based Input Weight Initialization for Echo State Networks

Peter Steiner, Azarakhsh Jalalvand, *Member, IEEE*, and Peter Birkholz, *Member, IEEE*

***Abstract*— Echo state networks (ESNs) are a special type of recurrent neural networks (RNNs), in which the input and recurrent connections are traditionally generated randomly, and only the output weights are trained. Despite the recent success of ESNs in various tasks of audio, image, and radar recognition, we postulate that a purely random initialization is not the ideal way of initializing ESNs. The aim of this work is to propose an unsupervised initialization of the input connections using the *K*-means algorithm on the training data. We show that for a large variety of datasets, this initialization performs equivalently or superior than a randomly initialized ESN while needing significantly less reservoir neurons. Furthermore, we discuss that this approach provides the opportunity to estimate a suitable size of the reservoir based on prior knowledge about the data.**

***Index Terms*— Clustering, echo state networks (ESNs), reservoir computing, unsupervised pretraining.**

I. Introduction

SINCE the breakthrough of echo state networks (ESNs) [1], a lot of design strategies for ESNs have been proposed. Although randomly initialized ESNs have achieved state-of-the-art results in various directions, several publications, such as [2]–[4], argue that there should exist better approaches that incorporate more prior or biologically plausible knowledge. According to [5], it requires a lot of trial and error methods to initialize an ESN for a task, and the relationship between the different weight matrices is not completely understood.

To better understand the behavior of randomly initialized ESNs for digit and phoneme recognition, Jalalvand *et al.* [6], Jalalvand *et al.* [7], and Bala *et al.* [8] have analyzed an ESN with optimized hyperparameters and determined the impact of the different hyperparameters. For example, it turned out that even very sparse weight matrices are still sufficient for achieving proper results. Similarly, in [9], it was shown that the hyperparameters should be tuned to match the spectral properties of the reservoir states and the target outputs. Another way to design ESNs more efficiently is to concatenate multiple reservoirs. Different architectures, such as layered ESNs [6] or deep and tree ESNs [10]–[13], were shown to improve the performance over a single-layer ESN while reducing the training complexity by utilizing smaller reservoirs in each layer.

Other approaches step even farther away from random initialization. The publications [14]–[16] proposed simple ESN reservoirs in different flavors, e.g., delay lines with optional feedback, cyclic reservoirs, or even a simple chain of neurons. It was shown that the proposed reservoir design strategies outperformed randomly initialized ESNs in various aspects, such as classification or regression accuracy and in terms of memory capacity. A big advantage is a relatively high sparsity, which is memory-efficient and computationally cheap. However, at least in [15], the authors warned that their findings might not hold in practice, when one needs to deal with high-dimensional inputs or more complex tasks. A related approach to simplify the reservoir initialization was proposed in [17], where the reservoir weights are only allowed to have the values 0 and ±1. Starting with the values 1 and 0, which were assigned to the reservoir in a deterministic way, several 1 weights were flipped to -1 and the authors have shown that this strongly influenced the behavior of the reservoir in terms of fitting error and memory capacity. In [18], the aforementioned approaches were further simplified, and only one neuron was used in the reservoir. The output of the neuron was fed back to its input using different delay times. This produced virtual nodes simulated a larger reservoir.

All these pioneering approaches try to initialize the reservoir and/or input weights in a more or less deterministic way that is almost task-independent. Alternative techniques also aim to initialize the ESN in a deterministic way that is, however, more task-dependent or dependent on the input data. For example, [19] adopted recurrent self-organizing maps (SOMs) to initialize the input and recurrent weight matrices using the SOM algorithm. Therefore, a new neuron model was used, and the weight matrices were pretrained using the unsupervised SOM algorithm. In [20], scale-invariant maps (SIMs), an extension of SOMs, were used to initialize the input weights. The same group also used the Hebbian learning in [21]. Lazar *et al.* [22] proposed a biologically inspired self-organizing recurrent neural network (SORN) consisting of spiking neuron models, in which the weights of frequently firing neurons are increased during training. This was adopted for the batch intrinsic plasticity (BIP) for ESNs [23], where the reservoir weights were iteratively pretrained.

Manuscript received March 1, 2021; revised August 17, 2021 and October 25, 2021; accepted January 19, 2022. This work was supported in part by the Europäischer Sozialfonds (ESF), in part by the Free State of Saxony under Grant 100327771, and in part by the Special Research Fund of Ghent University under Grant BOF19/PDO/134. *(Corresponding author: Peter Steiner.)*

Peter Steiner and Peter Birkholz are with the Institute of Acoustics and Speech Communication, Technische Universität Dresden, 01069 Dresden, Germany (e-mail: peter.steiner@tu-dresden.de; peter.birkholz@tu-dresden.de).

Azarakhsh Jalalvand is with IDLab, Ghent University–imec, 9052 Ghent, Belgium, and also with the Mechanical and Aerospace Engineering Department, Princeton University, Princeton, NJ 08544 USA (e-mail: azarakhsh.jalalvand@ugent.be).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TNNLS.2022.3145565.

Digital Object Identifier 10.1109/TNNLS.2022.3145565

Yet another family of unsupervised learning algorithms are clustering techniques. In [24], the inverse weighted K-means (IWK) algorithm [25], [26] was proposed to initialize the weight matrices of an ESN. After randomly initializing the input weights, they applied IWK to the neuron inputs and adapted the input weights. Then, they randomly initialized the recurrent weights and applied IWK again on the reservoir states to adapt the recurrent weights. The authors showed that their method outperformed a randomly initialized ESN and that the performance gets more stable when repeating random initialization.

In this article, we present an alternative strategy to initialize the ESN input weights using the K-means algorithm. Instead of applying the cluster algorithm on the neuron inputs, we used the K-means algorithm to cluster the input features and used the centroid vectors as input weights.

The main advantages of our approach are as follows.

1) The pretrained ESNs perform equally well or better than ESNs without pretraining and need smaller reservoirs. This will be discussed in detail on a large-scale video dataset and evaluated on a broad variety of datasets with different characteristics.
2) We show that the same hyperparameter optimization strategy proposed in [6] and [7] for conventional ESNs can be applied to optimize the hyperparameters of the novel K-means-based initialized ESN (KM-ESN).
3) Applying the clustering techniques is a common data exploration step to study possible correlations within the data. Our approach efficiently benefits from the outcome of this step to also initialize the input weights.
4) Since the clusters are usually associated with the classes to be recognized, e.g., phones in speech or notes in music, the procedure of our approach is interpretable.

The rest of this article is structured as follows. In Section II, we introduce the basic ESN and our proposed unsupervised input weight initialization. In Section III, we introduce, optimize, and evaluate ESNs for door state classification in videos. In Section IV, we present results on a wide variety of multivariate datasets. Finally, we summarize our conclusions and give an outlook to future work in Section V.

II. Methods

Here, we introduce the basic ESN and the K-means algorithm and explain how the input weights of an ESN can be initialized using the K-means algorithm.

A. Basic Echo State Network

The main outline of a basic ESN is depicted in Fig. 1. The model consists of the input weights $\mathbf{W}^{\text{in}}$, the reservoir weights $\mathbf{W}^{\text{res}}$, and the output weights $\mathbf{W}^{\text{out}}$. The input weight matrix $\mathbf{W}^{\text{in}}$ has the dimension of $N^{\text{res}} \times N^{\text{in}}$, where N^{res} and N^{in} are the size of the reservoir and dimension of the input feature vector $\mathbf{u}[n]$ with the time index n, respectively. Typically, the values inside the input weight matrix are initialized randomly from a uniform distribution between ± 1 and are scaled afterward using the input scaling factor α_u, which is a hyperparameter to be tuned. In [6], it was shown that it is sufficient to have only a limited number of connections from the input nodes to the nodes inside the reservoir. We therefore connect each node of the reservoir to only $K^{\text{in}} = 10$ ($\ll N^{\text{in}}$) randomly selected input features. This makes $\mathbf{W}^{\text{in}}$ very sparse and feeding the feature vectors into the reservoir potentially more efficient.

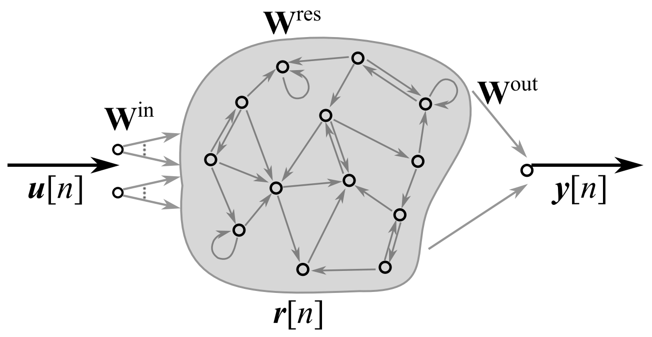


Fig. 1. Main components of a basic ESN. The input features $\mathbf{u}[n]$ are fed into the reservoir using the fixed input weight matrix $\mathbf{W}^{\text{in}}$. The reservoir consists of unordered neurons, sparsely inter-connected via the fixed reservoir matrix $\mathbf{W}^{\text{res}}$. The output $\mathbf{y}[n]$ is a linear combination of the reservoir states $\mathbf{r}[n]$ based on the output weight matrix $\mathbf{W}^{\text{out}}$, which is trained using linear regression.

The reservoir weight matrix $\mathbf{W}^{\text{res}}$ is a square matrix of the size $N^{\text{res}} \times N^{\text{res}}$. Typically, the values inside this matrix are initialized from a standard normal distribution. Similar to the input weight matrix, we connect each node inside the reservoir to a limited number of $K^{\text{rec}} = 10$ ($\ll N^{\text{res}}$) randomly selected other nodes in the reservoir, and set the remaining weights to zero. In order to fulfill the echo state property (ESP) that requires that the states of all reservoir neurons need to decay in a finite time for a finite input pattern, the reservoir weight matrix is normalized by its largest absolute eigenvalue and rescaled by the spectral radius ρ, because it was shown in [1] that the ESP holds as long as $\rho < 1$.

Together, the input scaling factor α_u and the spectral radius ρ determine how strongly the network relies on the memorized past inputs compared with the present input. These hyperparameters need to be optimized during the training process.

Every neuron inside the reservoir receives an additional constant bias input. The bias weight vector $\mathbf{w}^{\text{bi}}$ with N^{res} entries is initialized by fixed random values from a uniform distribution between ± 1 and multiplied by the hyperparameter α_b. With the three weight matrices $\mathbf{W}^{\text{in}}$, $\mathbf{W}^{\text{res}}$, and $\mathbf{w}^{\text{bi}}$, the reservoir state $\mathbf{r}[n]$ can be computed as follows:

$$\mathbf{r}[n] = (1-\lambda)\mathbf{r}[n-1] + \lambda f_{\text{res}}\big(\mathbf{W}^{\text{in}}\mathbf{u}[n] + \mathbf{W}^{\text{res}}\mathbf{r}[n-1] + \mathbf{w}^{\text{bi}}\big). \quad (1)$$

Equation (1) is a leaky integration of the reservoir neurons, which is equivalent to a first-order low-pass filter. Depending on the leakage $\lambda \in (0, 1]$, a specific amount of the past reservoir state is leaked over time. Together with the spectral radius ρ, the leakage λ determines the temporal memory of the reservoir.

The reservoir activation function $f_{\text{res}}(\cdot)$ controls the nonlinearity of the system. Conventionally, the sigmoid or tanh functions are used, because their lower and upper boundaries facilitate the reservoir state stability.

The output weight matrix $\mathbf{W}^{\text{out}}$ has the dimensions $N^{\text{out}} \times (N^{\text{res}} + 1)$ and connects the reservoir state $\mathbf{r}[n]$, which is expanded by a constant intercept term of 1 for regression, to the output vector $\mathbf{y}[n]$ using the following equation:

$$\mathbf{y}[n] = \mathbf{W}^{\text{out}}\mathbf{r}[n]. \tag{2}$$

Typically, the output weight matrix is computed using ridge regression. Therefore, all reservoir states calculated for the training data are concatenated into the reservoir state collection matrix $\mathbf{R}$. As linear regression usually contains one intercept term, every reservoir state $\mathbf{r}[n]$ is expanded by a constant of 1. All desired outputs $\mathbf{d}[n]$ are collected into the output collection matrix $\mathbf{D}$. Then, $\mathbf{W}^{\text{out}}$ can be computed using the following equation, where ϵ is the regularization parameter that needs to be tuned on a validation set:

$$\mathbf{W}^{\text{out}} = \left(\mathbf{R}\mathbf{R}^{\text{T}} + \epsilon\mathbf{I}\right)^{-1}\left(\mathbf{D}\mathbf{R}^{\text{T}}\right). \tag{3}$$

The size of the output weight matrix determines the total number of free parameters to be trained in ESNs. Because linear regression can be obtained in the closed form, ESNs are quite efficient and fast to train compared with typical deep-learning approaches.

B. *K-Means Clustering*

In this work, we studied the frequently used K-means algorithm [27] to improve the input weight initialization of ESNs. The K-means algorithm groups N feature vectors (observations) $\mathbf{u}[n]$ with N^{in} features into K clusters. Each observation is assigned to the cluster with the closest centroid, the prototype of the cluster. The basic K-means algorithm aims to partition all N observations into K sets $S_1, S_2, \ldots, S_K$ and thereby minimizes the within-cluster sum of squares (SSE)

$$\text{SSE} = \sum_{k=1}^{K} \sum_{\mathbf{u}[n]\in S_k} \|\mathbf{u}[n] - \mu_k\|^2. \tag{4}$$

Here, μ_k is the centroid of the kth set S_k, which is usually the mean of all points belonging to S_k.

In this article, we utilized relatively large datasets. Thus, we used the fast mini-batch K-means algorithm proposed by Sculley [28] to determine the μ_k and initialized it based on "K-Means++" [29].

C. *Novel Input Weight Initialization*

In this article, we propose to initialize the input weight matrix $\mathbf{W}^{\text{in}}$ using the cluster centers μ_k determined by the K-means algorithm. To understand how a feature vector is passed to the reservoir in general, we reconsider (1), which describes the computation of a new reservoir state based on the current feature vector and the previous reservoir state. For the sake of simplicity, we briefly assume a reservoir without leakage ($\lambda = 1$), without any recurrent connections ($K^{\text{rec}} = 0$), and with a linear activation function f_{res}. Thus, we can simplify (1) to (5) and (6) for the kth reservoir neuron

$$\mathbf{r}[n] = \mathbf{W}^{\text{in}}\mathbf{u}[n] \tag{5}$$

$$r_k[n] = \sum_{m=1}^{N^{\text{in}}} w_{k,m}^{\text{in}} u_m[n] = \mathbf{w}_k^{\text{in}} \cdot \mathbf{u}[n] \tag{6}$$

where m is the feature index inside $\mathbf{u}[n]$, and $\mathbf{w}_k^{\text{in}}$ is the kth row of $\mathbf{W}^{\text{in}}$ with the presynaptic input weights for the kth neuron in the reservoir.

This dot product is in fact closely related to the cosine similarity S in (7). The only difference between (6) and (7) is the normalization

$$S = \frac{1}{\|\mathbf{w}_k^{\text{in}}\| \, \|\mathbf{u}[n]\|} \, \mathbf{w}_k^{\text{in}} \cdot \mathbf{u}[n]. \tag{7}$$

The input weights of an ESN are responsible for passing feature vectors to the reservoir that consists of nonlinear neurons. Due to the—typically—random initialization of the input weights, several linear combinations of the input features for different neurons in the reservoir can be computed. In this article, we do not neglect this assumption, but we hypothesize that the main task of the input weights is to structure features according to their similarity. We also stick to the conventional linear regression-based training of ESNs, because this is a key advantage of such networks. However, we would like to incorporate prior knowledge about the feature vectors, in an unsupervised fashion, so that it is "easier" for the ESN to solve a specific task.

Thus, we propose to replace the randomly initialized input weights by the cluster centroids obtained from the K-means algorithm, i.e., $\mathbf{w}_k^{\text{in}} = \mu_k$. The K-means algorithm detects prior structure in the feature vectors, such as phones or phone transitions in speech datasets, common segments of images [30]. In this way, passing feature vectors to the ESN basically consists of computing the cosine similarity between the centroids and the feature vectors.

Typically, the reservoir size N^{res} is increased after tuning the hyperparameters using small ESNs. We have been shown that this final step significantly improves the classification results [6], [7], [31]–[34]. However, if we would simply increase the reservoir size in our novel ESN model, we needed also to increase K, as each reservoir neuron represents one cluster so far. If we increase K too much, we might end up with less meaningful clusters. Thus, in this article, we propose that K does not need to be equal to N_{res}.

This has the advantage that we can increase K and N_{res} together, until the improvement of further increasing K gets low. Then, we can keep K constant and add additional "zero-connections" to $\mathbf{W}^{\text{in}}$. In that way, we ensure that the centroids are still representing useful information, and we reduce the computational complexity by using very sparse $\mathbf{W}^{\text{in}}$ in the case of large reservoirs. By padding the new input weights with a lot of "zero-connections," we specifically limit the amount of neurons in the reservoir that receive input features. This can be compared with a cortical column in the brain that also mainly consists of recurrent connections and in which only a part of the neurons directly receives input information [35]. Thus,

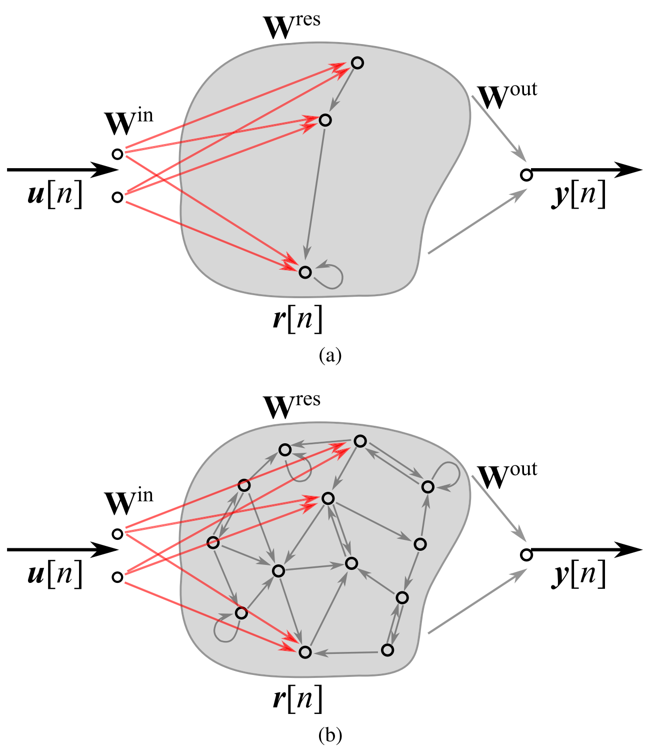


Fig. 2. Example ESN architectures for (a) $K = N^{\text{res}} = 3$ and (b) $K = 3$ and $N^{\text{res}} = 13$. Only the nonzero parts of $\mathbf{W}^{\text{in}}$ are visualized in red. For the sparse KM-ESN, only a few neurons receive input data.

very large K-means-based ESNs are even getting biologically plausible.

Fig. 2 visualizes two very simple examples for the proposed K-means-based ESN architectures. Fig. 2(a) shows the case $K = N^{\text{res}} = 3$, where the number of centroids is equal to the reservoir size. Fig. 2(b) shows the case $K = 3$ and $N^{\text{res}} = 13$, where the number of centroids is much smaller than the reservoir size. As indicated by the red arrows, only a very small amount of the neurons inside the reservoir receive information directly from the input features, while most of the neurons are only connected to other neurons inside the reservoir.

In the rest of this article, we refer to the basic ESN with randomly and sparsely initialized input weights as "basic ESN" and to the ESN with K-means-based initialized input weights as "KM-ESN." If $N^{\text{res}} > K$, we call it "sparse KM-ESN."

III. Experiment 1: Door State Recognition

In the first experiment, we consider a frame-level classification task, namely, event detection in door surveillance systems. The task is to continuously classify the status of a door that can be opened, closed, or half-opened from a low-resolution camera sensor. Using a large-scale dataset, we illustrate the impact of the K-means clustering on the hyperparameters.

A. Dataset

We used the publicly available dataset [36] that contains recordings of a low-resolution camera (Avago Technologies ADNS-3080 mouse sensor) set up in front of a door. The resolution was 30×30 pixels with a frame rate of 90 frames per second. The dataset has more than 830 000 frames in total. For each frame, the label (0 for the closed, 1 for the half-opened, and 2 for the opened door, respectively) was semiautomatically generated using magnetic sensors that were placed in the middle of the door and close to the door hinge. In the dataset, three movies of different lengths are included, where the camera position in each movie was slightly displaced to introduce more variable input features.

The dataset does not have any default split into training, validation, and test sets. We prepared the dataset as follows. Each movie was split into consecutive sequences with a length of ≈1 min. The first half of each movie was used as the training set (N) and the latter one as the test set. To optimize the hyperparameters, we only used the first movie, whose training set was partitioned in fivefold for cross validation. We sequentially optimized the hyperparameters according to [32].

B. Feature Extraction

Following [36], we converted each frame with 30×30 pixels to a vector with 900 elements and did not consider further feature reduction techniques. Since the pixel values were integers between 0 and 255 (grayscale values), we divided each value by 255 to obtain values between 0 and 1. We did not do any further preprocessing steps and directly used the rescaled vectors as input for the ESN models. The training set contained ≈76 min of data leading to $N_{\text{samples}} = 415\,620$ frames in total.

C. Target Preparation and Readout Postprocessing

In this task, we have three binary outputs, one for each door state (close, half-open, and open). We converted the integer label of each frame in a 3-D output with one-hot encoded targets, where only the output indexed by the integer label is 1.

During inference, the output with the highest value in every frame indicated the current state of the door.

D. Measurements

To optimize the hyperparameters, we used the mean squared error (MSE) between the one-hot encoded targets and the computed outputs.

To report the final performance, the frame-level error rate (FER) (portion of frames assigned to the wrong class, the following equation) was used:

$$\text{FER} = \frac{N_{\text{error}}}{N_{\text{frames}}} \tag{8}$$

where N_{error} and N_{frames} were the misclassified frames and the total number of frames, respectively.

E. Number of Centroids

The key parameter of the K-means algorithm is K, the number of clusters to be used. This is strongly task-dependent

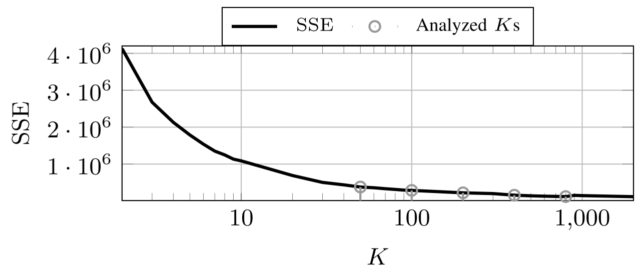


Fig. 3. SSE for different numbers of centroids. A fast decrease was observed until $K \approx 20$. Afterward, SSE still continued to decrease more slowly.

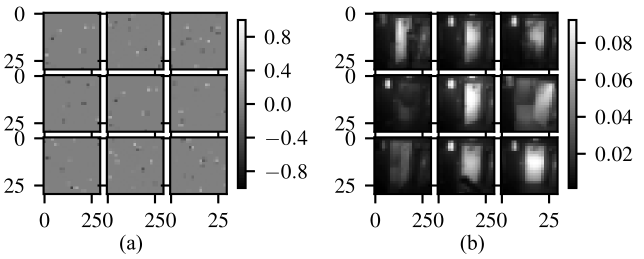


Fig. 4. Input weights for both ESN models. (a) In the case of the basic ESN, every neuron received ≈10 randomly selected input features and thus has ≈890 zero connections. The remaining values are uniformly distributed between ±1. (b) In the case of the KM-ESN, the weights were learned from the training dataset and represent basically mean values of various images.

and there exists no general solution for this optimization problem. One way to determine the number of centroids is to observe the summed squared error (SSE) for different K values and to search for the point when the slope of the SSE gets less steep. From Fig. 3, where the SSE is visualized over K for the training set, we observed that the SSE decreased quite fast until $K \approx 50$. Afterward, SSE still continued to decrease but with a slower pace.

The low effective number of clusters that is able to cluster the dataset to some extent is in line with the nature of this dataset. Given that the camera was fixed in each movie and that it recorded a limited set of interactions between humans and the door, the majority of all pixels were more or less constant over time. Although different persons interacted with the door, the resolution of the camera is rather low so that it mostly captures the overall shape of each person. Since the dataset is rather noisy, it was furthermore difficult for the camera to record objects (e.g., chairs) that the people carried into or out of the room. During the reservoir hyperparameter optimization, we first of all fixed the number of centroids to $K = 50$ and later increased it together with the reservoir size.

F. Optimization of the Hyperparameters of the ESNs

Before optimizing the hyperparameters of the ESN models, we compared the input weights of the basic ESN and of the KM-ESN. Therefore, the input weights to nine randomly selected reservoir neurons are visualized in Fig. 4 as follows. Each neuron has a 900-dimensional input weight vector. We have reshaped them into 30 × 30 images, since the input weights connect exactly one input image of the same size to the particular neuron. The following differences can be observed.

1) The input weights of the basic ESN [see Fig. 4(a)] are very sparse (0.06% nonzero values) and have uniformly distributed nonzero values between ±1.
2) Fig. 4(a) shows that images are randomly fed in the basic ESN.
3) The KM-ESN [see Fig. 4(b)] is dense and the values are approximately distributed between 0 and 0.25. This sparseness is caused by the dark regions in the frames.
4) Fig. 4(b) shows that the images are fed in the KM-ESN by the correlation with prototype images learned by the K-means algorithm.

These observations can be explained by the different ways of initializing the basic ESN and the KM-ESN. For the first one, we partially followed [36], where each neuron received only $K^{\text{in}} = 5$ randomly chosen inputs. In contrast to [36], we increased K^{in} to 10, as we already know that the level of sparseness does not have a significant impact on the performance [36]. The other observations show that the K-means algorithm learned exactly what we expected it to learn—average images for each cluster. Since the weights of the basic ESN and the KM-ESN are different, we expect the two models to behave differently in the remaining experiment.

The last observation is the most interesting one, since it perfectly visualizes that the K-means algorithm has learned different average images from the training dataset. Thus, different opening phases of the door or silhouettes of people in the room or even the displacement of the camera due to different positions can be observed in Fig. 4(b).

Since the input weights of the KM-ESN are initialized using a purely data-driven approach, the value range strongly depends on the value range of the features. In the case of the video dataset, the values are bounded between 0 and 1; hence, the values of the input weights of the KM-ESN are also nonnegative. For the following hyperparameter optimization, the different distributions of the input weights will lead to significant differences between the hyperparameters of the basic ESN and of the KM-ESN, especially for the input scaling and spectral radius.

In order to optimize the hyperparameters of both basic and KM-ESN, we followed the sequential optimization approach introduced in [32] and [33]. A similar outline was recently published in [37]. In our preliminary experiments, we compared this sequential method with a fully randomized optimization by jointly exploring the entire hyperparameter space. We found that the sequential optimization required fewer search steps and led to a lower loss function. Therefore, in the following, we use the sequential optimization process.

We began with a memory-less ESN ($\rho = 0$) with 50 reservoir neurons. Particularly, we fixed the leakage $\lambda = 1$ and removed the constant bias term by setting $\alpha_{\text{bi}} = 0$.

Then, we jointly optimized α_u and ρ using a random search with 200 iterations. The values for α_u were drawn from a uniform distribution between 0.01 and 1.0, and the values for ρ from a uniform distribution between 0 and 2.

The entire search space after fivefold cross validation is depicted in Fig. 5. To better present the results, we have

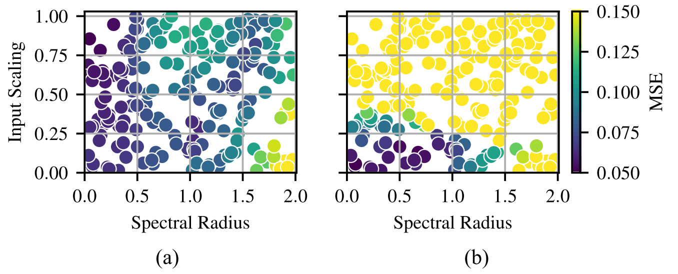


Fig. 5. MSE after the joint optimization of the input scaling α_u and the spectral radius ρ. The overall behavior of (a) basic ESN and (b) KM-ESN is similar. The input scaling of the basic ESN (0.85) is very large compared with the one of the KM-ESN (0.05).

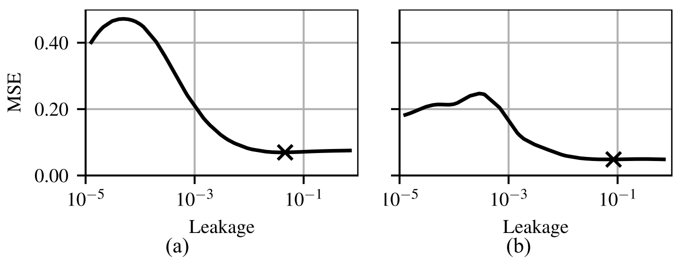


Fig. 6. MSE after the optimization of the leakage λ. The global behavior of (a) basic ESN and (b) KM-ESN is similar. As indicated by the crosses, the optimized leakage values of the basic ESN and of the KM-ESN are 0.05 and 0.08, respectively.

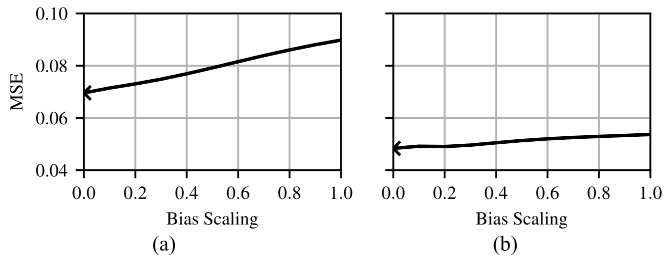


Fig. 7. MSE after the optimization of the bias scaling α_{bi}. The impact of bias scaling is rather small compared with the impact of the previous hyperparameters. As indicated by the crosses, the best performance was achieved with $\alpha_{\mathrm{bi}} = 0$ in both the cases. (a) Basic ESN. (b) KM-ESN.

visualized them using the expression $\min(0.15, \mathrm{MSE})$. We achieved the lowest validation MSE with $(\alpha_u, \rho) = (0.85, 0.05)$ for the basic ESN and $(\alpha_u, \rho) = (0.05, 0.08)$ for the KM-ESN. Comparing the optimization results in Fig. 5, the general behavior of these basic ESNs and KM-ESNs is different. In particular, there are two marginal differences. The input scaling is lower in the case of the KM-ESN, whereas the spectral radii are similar. The area of the best hyperparameters in the case of the KM-ESN is much smaller than the area in the case of a basic ESN.

The differences in the input scaling values are caused by different input weights of the basic ESN and of the KM-ESN. While most of the values in the input weights of the basic ESN are zero, the remaining values need to strongly activate the reservoir neurons, which is achieved with a large input scaling. The KM-ESN in contrast has learned prototype images. If the input image is very similar to a prototype, the cross correlation between the input image and the prototype is high, leading to a strong activation of the associated neuron. Thus, despite the smaller absolute values of the input weights in the KM-ESN, a significantly smaller input scaling value is required. Since the ratio between input scaling and spectral radius is almost 1 in the case of the KM-ESN, it relies more on a combination of current and past information than the basic ESN, which mostly benefited from the current input.

The next hyperparameter to be optimized was the leakage λ. Again, we used a random search to optimize this hyperparameter for the basic ESN and for the KM-ESN. This time, we used a logarithmic uniform distribution between $1e-5$ and 1, because we expected that a large leakage, i.e., very small λ is required for the ESN to make the decision based on a wide range of memory. The results in Fig. 6 show that the leakage has a strong impact on the final performance. Again, the global behavior of the basic ESN and the KM-ESN was similar. The final values were 0.05 and 0.08 for the basic ESN and for the KM-ESN, respectively.

The low values for λ are reasonable for this task, since the dataset is noisy and the outputs need to be constant for a longer time, especially for long phases with an opened or closed door. Thus, a small λ is desired that acts as a first-order low-pass filter [38] and smooths the reservoir states.

The last hyperparameter to be optimized was the bias scaling factor α_{bi} that controls the influence of a constant bias input to each reservoir neuron. In general, the bias scaling has a minor impact on the final ESN performance. Thus, we simplified the optimization scheme here and evaluated values from 0 to 1 with a step of 0.1 to optimize this parameter. Fig. 7 shows that the impact of the bias term is indeed small and that large bias inputs even decreased the performance of the ESN models. The basic ESN as well as the KM-ESN did not need any bias at all.

G. Impact of the Reservoir Size

As mentioned in the introduction, ESNs typically benefit from increasing the reservoir size after fixing the other hyperparameters. At the same time, we were reluctant to add to the complexity of the K-means model by increasing K as much as N_{res}. Therefore, we only increased K and N_{res} up to 200, and from that point, we only increased the reservoir size while $K = 200$. This means that the input layers of the ESNs were fully connected until $N_{\mathrm{res}} = 200$, and for all the larger models, the input features were connected to only 200 reservoir nodes (i.e., sparse input connections). We also repeated the hyperparameter optimization for these sparsely connected KM-ESNs.

The optimized hyperparameters for the new initialized ESNs are summarized in Table I. Since introducing neurons in the sparse reservoir that did not receive any input information strongly changes the overall system behavior, the optimal hyperparameters have changed. In particular, it is interesting that the spectral radius strongly increased by a factor of 10 while input scaling increased only by a factor of 3. This means that especially the sparse KM-ESN not only uses the current input to compute the output but also needs the memory

TABLE I

OPTIMIZED HYPERPARAMETERS FOR THE BASIC ESN, SMALL KM-ESN ($N_{\text{res}} < 200$ AND DENSE INPUT LAYER), AND LARGE KM-ESN ($N_{\text{res}} > 200$ AND SPARSE INPUT LAYER) AFTER FIVEFOLD CROSS VALIDATION

	Basic ESN	KM-ESN	
Parameter		dense	sparse
Input scaling α_{u}	0.85	0.06	0.13
Spectral radius ρ	0.05	0.08	0.99
Leakage λ	0.04	0.08	0.35
Bias scaling α_{bi}	0	0	0
Regularization ϵ	13×10^{-4}	6×10^{-4}	3×10^{-4}

TABLE II

COMPUTATIONAL COMPLEXITY TO TRAIN THE BASIC ESN AND THE KM-ESN. THE K-MEANS ALGORITHM INCREASES THE COMPLEXITY BUT NEEDS TO BE PERFORMED ONLY ONCE BEFORE THE HYPERPARAMETER OPTIMIZATION

Action	Complexity
K-Means [39], [40]	$\mathcal{O}(N^{\text{in}} + N_{\text{samples}} K N^{\text{in}})$
Compute $\mathbf{R}$	$\mathcal{O}(N_{\text{samples}} N^{\text{res}})$
Update $\mathbf{R}\mathbf{R}^{\text{T}}$	$\mathcal{O}(N_{\text{samples}} (N^{\text{res}})^2)$
Update $\mathbf{D}\mathbf{R}^{\text{T}}$	$\mathcal{O}(N_{\text{samples}} N^{\text{res}} N^{\text{out}})$
Invert $\left(\mathbf{R}\mathbf{R}^{\text{T}} + \epsilon\mathbf{I}\right)$	$\mathcal{O}((N^{\text{res}})^3)$
Compute $\mathbf{W}^{\text{out}}$	$\mathcal{O}((N^{\text{res}})^2 N^{\text{out}})$

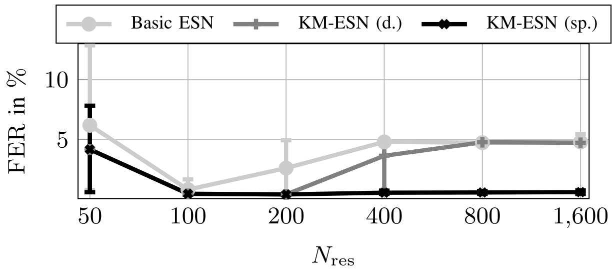


Fig. 8. Mean, minimum, and maximum FERs for different reservoir sizes after ten random initializations. The KM-ESN always performed equally well or better than the basic ESN for all reservoir sizes. The vertical bars indicate the minimum and maximum FER of the respective models.

provided by the recurrent connections. Less smoothing by the leaky integration is required now.

In Fig. 8, the final FER computed on the test set for different ESN architectures are visualized. Overall, for both the basic ESN and the KM-ESN, only a small reservoir with 100 neurons was required to achieve a reasonable performance that was clearly below 1% FER. In particular, the best performing basic ESN with 100 neurons achieved an average FER of 0.85% and the same KM-ESN an average FER of 0.50%. In the case of reservoirs with more than 100 neurons, we noticed that the performance of the basic ESN strongly decreased with an FER of about 5%. The same effect occurred in the case of the dense KM-ESN with more than 200 neurons. Reconsidering the SSE in Fig. 3, we noticed that it almost stopped decreasing with more than 200 neurons. In the case of large basic ESNs, it is likely that many static pixels are selected by the input weights, and in the case of large dense KM-ESNs, it is likely that a lot of centroids are close to each other. This, in turn, means that large KM-ESNs receive a lot of input information in an almost equivalent way, which is counterproductive, since ESNs in general benefit from diverse input. By switching to the sparse KM-ESN when increasing the reservoir size beyond 200 neurons, we restricted the number of centroids and thus did not split up meaningful clusters in too many small subclusters. In addition, we boosted the impact of the recurrent connections, since the number of parameters in W^{res} increases quadratically with N^{res}. From Fig. 8, we can conclude that the performance does not decrease in the case of large sparse KM-ESNs and further improved to FER $= 0.44\%$ for $K = 200$ and $N^{\text{res}} = 400$. Reconsidering Table I, we can say that large reservoirs benefit from the memory incorporated by means of the high spectral radius. For $N^{\text{res}} = 1600$, the FER slightly increased toward 0.64%. Overall, the KM-ESN (regardless dense or sparse) was always more robust against ten different random initializations with only one exception: the dense KM-ESN with 400 neurons, when the performance got more similar to the basic ESN.

H. Computational Complexity

According to [6], passing data through the ESN can be decomposed in a series of actions that include computing the reservoir states, updating the correlation matrices, matrix inversion, and output weight computation. In the case of the KM-ESN, the K-means training is an additional initial step.

The complexity of all steps are summarized in Table II and shows that the K-means algorithm adds to the complexity but needs to be performed only once before the hyperparameter optimization.

IV. EXPERIMENT 2: MULTIDATASET EVALUATION

In this section, we focus on evaluating the KM-ESN on a large variety of datasets with different characteristics, such as dataset size, feature vector size, sequence length, and data type.

A. Datasets

We used exactly the same datasets as in [41], which were provided in the accompanying Github repository.[1] Statistics about the datasets are summarized in Table III and show the diversity of the tasks such as single- and multi-input time series as well as binary classification and multiclass tasks.

The datasets are by default split into training and test subsets. As before, for hyperparameter optimization, we partitioned the training set in fivefold for cross validation and then again sequentially optimized the hyperparameters as in [32] and in the previous experiment. Since the datasets "CMU

[1]https://github.com/FilippoMB/Time-series-classification-and-clustering-with-Reservoir-Computing

TABLE III

DETAILS ABOUT THE DATASETS FOR THE MULTIDATASET EVALUATION WITH THE NUMBER OF INPUT FEATURES (N^{in}), NUMBER OF OUTPUTS (N^{out}), MEAN, MINIMUM, AND MAXIMUM SEQUENCE DURATION (T_{mean}, T_{min} AND T_{max}), THE ENTIRE NUMBER OF FEATURE VECTORS (OBSERVATIONS) IN THE TRAINING SET AND TEST SET ($N_{\mathrm{samples,train}}$, $N_{\mathrm{samples,test}}$), AND THE NUMBER OF SEQUENCES IN THE TRAINING AND TEST SET (#TRAIN, #TEST)

Dataset	Abbreviation	N^{in}	#Train	#Test	N^{out}	T_{mean}	T_{min}	T_{max}	$N_{\mathrm{samples,train}}$	$N_{\mathrm{samples,test}}$
Swedish Leaf	SWE	1	500	625	15	128	128	128	64 000	80 000
Chlorine Concentration	CHLO	1	367	3840	3	166	166	166	77 522	637 440
DistalPhalanxTW	PHAL	1	400	139	3	80	80	80	32 000	11 120
ECG	ECG	2	100	100	2	89.5	39	152	9014	8884
Libras	LIB	2	180	180	15	45	45	45	8100	8100
Character Trajectories	CHAR	3	300	2558	20	120	109	205	36 070	306 869
uWave	UWAV	3	200	427	8	315	315	315	63 000	134 820
NetFlow	NET	4	803	534	13	230.7	50	994	182 881	125 506
Wafer	WAF	6	298	896	2	136.8	104	198	40 833	122 450
Robot Failures	ROBOT	6	100	64	4	14.8	12	15	1476	950
Japanese Vowels	JPVOW	12	270	370	9	15.6	7	29	4274	5687
Arabian Digits	ARAB	13	6600	2200	10	39.8	4	93	263 256	87 063
Auslan	AUS	22	1140	1425	95	57.3	45	136	63 371	83 578
CMU subject 16	CMU	62	16	10	2	305.0	127	580	8462	9229
Kick vs. Punch	KICK	62	16	10	2	426.7	274	841	6413	4682
Walk vs. Run	WALK	62	28	16	2	367.9	128	1918	10 926	5261
PEMS	PEMS	963	267	173	7	144	144	144	38 448	24 912

subject 16," "Kick versus Punch," and "Walk versus Run" contained only very few training sequences, we used threefold cross validation to optimize the hyperparameters for these tasks.

B. Feature Extraction

Most of the datasets were already preprocessed to some extent. For all datasets except for NetFlow, we subtracted the mean value from each feature and normalized it to unitary variance.

The NetFlow dataset was almost binary and the proposed normalization was not applicable. Instead, we simply rescaled each feature to the range between 0 and 1.

C. Target Preparation and Readout Postprocessing

For all datasets, we have binary outputs, one for each class in the particular dataset. For each dataset, the target outputs (classes) were one-hot encoded across the entire sequence (0 for the inactive classes and 1 for the active class).

During inference, the class scores were obtained by accumulating the class readouts over time. The recognized class is determined as the class with the highest accumulated score over time.

D. Measurements

To measure the overall classification results and to optimize the hyperparameters, the classification error rate (CER), as shown in the following equation, was used:

$$\mathrm{CER} = \frac{N_{\mathrm{error}}}{N_{\mathrm{seq}}} \tag{9}$$

where N_{error} and N_{seq} were the number of incorrect classified sequences and the overall number of sequences, respectively.

E. Number of Centroids

As in the previous experiment, we optimized the number of clusters K of the K-means algorithm. For each dataset, we computed the SSE for different K values and evaluated the values 50, 100, 200, 400, 800, and 1600. In some datasets, e.g., the Auslan (AUS) and the CMU subject 16 (CMU) datasets, we observed that the performance decreased or stagnated when increasing K too much. In that case, we stopped increasing K as soon as the performance dropped and instead switched to a sparse KM-ESN when further enlarging the reservoir size. Since the ROBOT dataset contained less than 1600 samples, we did not evaluate $K = 1600$.

F. Optimization of the Hyperparameters of the ESNs

The optimization of the ESN models followed the same strategy as in Section III-F with one exception. Instead of starting with a default leakage of 1.0, we chose 0.1, because the target outputs were constant across the entire sequence. Thus, we expected a lower leakage. The subsequent optimization procedure was then the sequential optimization of the hyperparameters, during which we minimized the cross-validation MSE.

The hyperparameters with the lowest cross-validation MSE were then used as the final hyperparameters. During the optimization, we fixed the reservoir size to 50 neurons.

As discussed for the previous experiment, the hyperparameters significantly influence the performance of the ESN

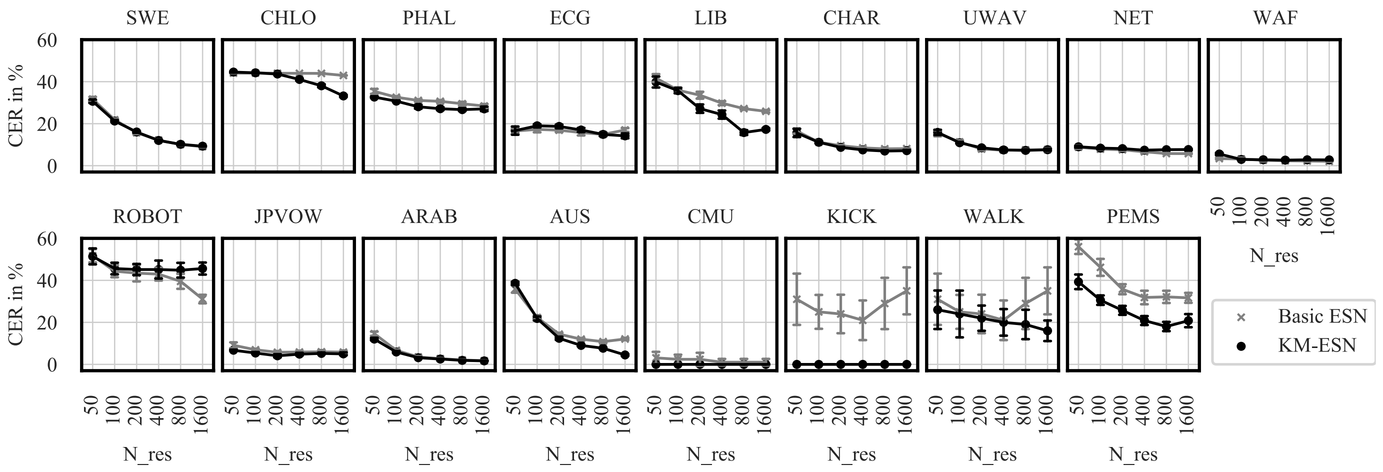


Fig. 9. Multidataset evaluation. The mean and standard deviation of the CER of the basic ESN and of the KM-ESN. For each dataset, the models were ten times randomly initialized, trained, and evaluated. The KM-ESN outperformed the basic ESN in many dataset, both the basic ESN and the KM-ESN performed equivalently in various datasets, and only in a few cases, the basic ESN performed better than the KM-ESN.

and they need to be tuned task-dependently. Thus, in contrast to [41], we used models with optimized hyperparameters to report our final results for each dataset. As can be expected and without reporting all the hyperparameters for every task, we observed that these parameters were significantly different across datasets.

G. Results

Fig. 9 shows the mean value and standard deviation of CER for the different datasets for different reservoir sizes and ten instances of each model. In order to investigate the impact of random initializations on the performance, we repeated the training procedure ten times. In each run, all weights of the basic ESN and the reservoir weights of the KM-ESN were randomly initialized, trained using the training datasets, and finally evaluated on the test sets. Fig. 9 shows that the KM-ESN outperformed the basic ESN in many datasets that both the basic ESN and the KM-ESN performed equivalently in various datasets, and only in a few cases, the basic ESN performed better than the KM-ESN. Since for some datasets and models, the standard deviation of the loss was so low and the error bars are not always visible. However, overall, the standard deviation of the loss of the KM-ESN is often lower than that of the basic ESN. Thus, as in the previous experiment, the KM-ESN is more robust against random initializations.

The KM-ESN outperformed the basic ESN in particular for very high-dimensional datasets, such as PEMS, KICK, and WALK, which all have more than 60 features. Another case, in which the KM-ESN performed remarkably better than the basic ESN, is the 2-D LIB dataset, where the features are uniformly distributed in a first glance. However, the features can still be well clustered, and thus, clustering the features still improves the performance of the KM-ESN compared with the basic ESN in the case of larger reservoirs. In the ARAB dataset, which consists of extracted MFCC features from Arabian spoken digits, the smaller KM-ESNs slightly outperformed the basic ESNs, whereas for larger reservoirs, the performance became more and more similar. Since this is a dataset, in which features can be linearly separated, we postulate that the KM-ESN works particularly well for linearly separable datasets. Interestingly, in the case of the datasets JPVOW and AUS, we can see that the KM-ESN not only outperformed the basic ESN, but it also allows to further increase the reservoir size without the risk of overfitting.

In the case of the CHLO and the PHAL datasets, the KM-ESN slightly outperformed the basic ESN. Since these datasets have 1-D input features, this means that, given linearly separable features, the K-means algorithm can cluster them. In fact, both CHLO and PHAL show patterns that make it possible to determine at least a few clusters, whereas the 1-D features of the SWE dataset are more normally distributed. Thus, the resulting centroids of the K-means algorithm are concentrated at the maximum of the distribution.

In the case of the ROBOT dataset, the performance of the KM-ESN was relatively low compared with the basic ESN. According to Table III, ROBOT was the smallest dataset to be considered in this study. It consists of less than 1500 samples. That means that, for the largest reservoir with 1600 neurons, we were forced to introduce sparsity to the KM-ESN. Furthermore, K-means algorithms are known to be sensitive against outliers. Applying the principal component analysis (PCA) on the ROBOT dataset showed a large amount of outliers in this dataset. This, in turn, means that it is likely that the K-means algorithm found a lot of centroids in a very small part of the feature space and thus many neurons in the KM-ESN received a lot of input information in an almost equivalent way. These could be the reasons why the KM-ESN performed worse than the basic ESN on the ROBOT dataset.

V. Conclusion and Outlook

We presented an effective way to initialize the input weights of ESNs using the unsupervised K-means algorithm. Motivated by the fact that passing feature vectors to a reservoir neuron is closely related to the cosine similarity, we used the centroids of the K-means algorithm as input weights. This controls the neuron activation such that they were high only if the feature vector and a centroid were similar. We showed that the input weight distribution of the basic ESN and the KM-ESN is significantly different, because the weights of the latter one were trained using the K-means algorithm and are thus basically the average value of subsets of the training samples. This supported the hypothesis that the K-means algorithm indeed supplied the KM-ESN with beneficial information about the dataset.

We demonstrated that the KM-ESN model outperformed basic ESNs on various datasets. First of all, we studied the impact of replacing randomly initialized input weights with centroids obtained by the K-means algorithm. Based on our experiments, the input weights are no more uniformly distributed between ± 1 and the initialization is driven by the data. Also the K-means-based ESN was more robust compared with random initialization, and the performance of the KM-ESN was higher than the performance of a basic ESN in particular for small numbers of neurons. In the case of the multidataset evaluation, we have presented different use cases together with suggestions when to prefer the KM-ESN or a basic ESN. It turned out that the KM-ESN is in particular useful for very high-dimensional datasets with a sufficient number of samples to train the K-means algorithm. However, if the features have outliers, such as in the ROBOT task, the KM-ESN was less successful than the basic ESN, which by itself did not perform well.

As the K-means-based input weight initialization is both data-driven and unsupervised, we obtain a set of input weights that is optimized for a given dataset. In general, if we would use a basic ESN for two different datasets with different features (e.g., audio features and sensor data) but with the same N^{in}, we would simply use the same random set of input weights. However, since the features have completely different characteristics, it is likely that task-dependently adapted input weights boost the performance of ESNs. Such an adaptation is the main reason to support KM-ESN. Furthermore, we can easily extend a given dataset with more data, e.g., video recordings with different camera positions or with additional objects and kinds of noise without needing labels to help the KM-ESN to generalize toward unknown situations.

In the future, one would analyze the capability of the proposed technique to solve more complex tasks, such as phoneme recognition in spoken language or multipitch tracking in music signals. It would also be interesting to determine the capability of predicting the time series. Another follow-up work would be to investigate ways for pretraining the reservoir weights as well. Furthermore, it would be interesting to study whether the proposed KM-ESN is a universal approximator for dynamic systems according to [42].

TABLE IV

Search Spaces for the Sequential Optimization Based on the Steps Proposed in [32], and for the Joint Randomized Search

Parameter	Search space		Iterations
α_{u}	10^{-3} to 1	uniform	200
ρ	0 to 2	uniform	
λ	10^{-5} to 1	loguniform	50
α_{bi}	0 to 2	uniform	21
ϵ	10^{-5} to 10	loguniform	50

TABLE V

Optimized Hyperparameters and Final Loss Values for the Basic ESN and for the Dense KM-ESN ($N_{\text{res}} < 200$) After the Joint Randomized Search and the Sequential Optimization

	Basic ESN		KM-ESN	
Parameter	random	sequential	random	sequential
Input scaling α_{u}	0.95	0.85	0.03	0.06
Spectral radius ρ	0.14	0.05	1.05	0.08
Leakage λ	0.03	0.04	0.14	0.08
Bias scaling α_{bi}	0.21	0	0.13	0
Regularization ϵ	15×10^{-4}	13×10^{-4}	11×10^{-4}	6×10^{-4}
Final loss	7.2×10^{-2}	7.0×10^{-2}	6.8×10^{-2}	4.8×10^{-2}

Code examples for the two experiments are publicly available in our Github repository (https://github.com/TUD-STKS/PyRCN/).

Appendix
Hyperparameter Optimization

Since the hyperparameter optimization explained in Section III-F and proposed in [32] was originally introduced for the basic ESN and successfully applied to various speech, music, and image recognition tasks, we checked whether it can be used for the KM-ESN as well.

To do so, we compared two optimization strategies on the video classification task (see Section III).

1) Sequential optimization as described in Section III-F using the parameters in Table IV. Only α_{bi} was optimized with a 1-D grid search. All other steps were randomized searches. We needed 321 optimization steps in total.
2) Fully randomized optimization by jointly exploring the search space consisting of all parameters in Table IV. We evaluated 2000 parameter combinations in total.

Since we used fivefold cross validation, each parameter combination (regardless sequential or joint optimization) was evaluated five times.

From the results in Table V, it can be seen that the sequential optimization and the joint randomized search in the case of the basic ESN led to similar hyperparameters. There are strong differences between the spectral radii and the bias scalings. The final loss is comparable as well.

In the case of the KM-ESN, we can see that the spectral radii are particularly different. In the case of the sequential optimization, the spectral radius is close to 0, whereas the spectral radius in the case of the joint randomized search is close to 1. However, the losses differ more than in the case of the basic ESN. Table V shows that it decreased much more in the case of the sequential search.

REFERENCES

[1] H. Jaeger, "The 'echo state' approach to analysing and training recurrent neural networks-with an erratum note," German Nat. Res. Center Inf. Technol., Schloss Birlinghoven, Sankt Augustin, Germany, Tech. Rep. 148, 2001. [Online]. Available: http://www.faculty.iu-bremen.de/hjaeger/pubs/EchoStatesTechRep.pdf and https://www.izb.fraunhofer.de/en/contact.html

[2] 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, doi: 10.1162/neco.2007.19.1.111.

[3] M. Lukoševičius, H. Jaeger, and B. Schrauwen, "Reservoir computing trends," *KI-Künstliche Intell.*, vol. 26, no. 4, pp. 365–371, Nov. 2012.

[4] S. Scardapane and D. Wang, "Randomness in neural networks: An overview," *Wiley Interdiscipl. Rev.: Data Mining Knowl. Discovery*, vol. 7, no. 2, p. e1200, Mar. 2017. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/widm.1200

[5] Y. Xue, L. Yang, and S. Haykin, "Decoupled echo state networks with lateral inhibition," *Neural Netw.*, vol. 20, no. 3, pp. 365–376, Apr. 2007. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0893608007000378

[6] A. Jalalvand, F. Triefenbach, K. Demuynck, and J.-P. Martens, "Robust continuous digit recognition using reservoir computing," *Comput. Speech Lang.*, vol. 30, no. 1, pp. 135–158, Mar. 2015.

[7] A. Jalalvand, K. Demuynck, W. De Neve, and J.-P. Martens, "On the application of reservoir computing networks for noisy image recognition," *Neurocomputing*, vol. 277, pp. 237–248, Feb. 2018.

[8] A. Bala, I. Ismail, R. Ibrahim, and S. M. Sait, "Applications of metaheuristics in reservoir computing techniques: A review," *IEEE Access*, vol. 6, pp. 58012–58029, 2018.

[9] P. V. Aceituno, G. Yan, and Y.-Y. Liu, "Tailoring echo state networks for optimal learning," *iScience*, vol. 23, no. 9, Sep. 2020, Art. no. 101440.

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

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

[12] C. Gallicchio and S. Scardapane, *Deep Randomized Neural Networks*. Cham, Switzerland: Springer, 2020, pp. 43–68.

[13] 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, doi: 10.1103/PhysRevLett.120.024102.

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

[15] 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. [Online]. Available: https://doi.org/10.1162/NECO_a_00374

[16] A. Griffith, A. Pomerance, and D. J. Gauthier, "Forecasting chaotic systems with very low connectivity reservoir computers," *Chaos: Interdiscipl. J. Nonlinear Sci.*, vol. 29, no. 12, Dec. 2019, Art. no. 123108, doi: 10.1063/1.5120710.

[17] T. L. Carroll and L. M. Pecora, "Network structure effects in reservoir computers," *Chaos: Interdiscipl. J. Nonlinear Sci.*, vol. 29, no. 8, Aug. 2019, Art. no. 083130, doi: 10.1063/1.5097686.

[18] L. Appeltant *et al.*, "Information processing using a single dynamical node as complex system," *Nature Commun.*, vol. 2, no. 468, pp. 1–6, Sep. 2011.

[19] M. Lukoševičius, "On self-organizing reservoirs and their hierarchies," Jacobs Univ., Bremen, Germany, Tech. Rep. 25, 2010.

[20] S. Basterrech, C. Fyfe, and G. Rubino, "Self-organizing maps and scale-invariant maps in echo state networks," in *Proc. 11th Int. Conf. Intell. Syst. Design Appl.*, Nov. 2011, pp. 94–99.

[21] S. Basterrech and V. Snášel, "Initializing reservoirs with exhibitory and inhibitory signals using unsupervised learning techniques," in *Proc. 4th Symp. Inf. Commun. Technol. (SoICT)*, New York, NY, USA, 2013, pp. 53–60, doi: 10.1145/2542050.2542087.

[22] A. Lazar, "SORN: A self-organizing recurrent neural network," *Frontiers Comput. Neurosci.*, vol. 3, p. 23, Mar. 2009. [Online]. Available: https://www.frontiersin.org/article/10.3389/neuro.10.023.2009

[23] B. Schrauwen, M. Wardermann, D. Verstraeten, J. J. Steil, and D. Stroobandt, "Improving reservoirs using intrinsic plasticity," *Neurocomputing*, vol. 71, nos. 7–9, pp. 1159–1171, Mar. 2008. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0925231208000519

[24] W. M. Ashour, A. S. Abu-Issa, and O. Hellwich, "Clustering algorithms in echo state networks," *Int. J. Signal Process., Image Process. Pattern Recognit.*, vol. 9, no. 5, pp. 15–24, May 2016.

[25] W. Barbakh and C. Fyfe, "Online clustering algorithms," *Int. J. Neural Syst.*, vol. 18, no. 3, pp. 185–194, 2008. [Online]. Available: https://doi.org/10.1142/S0129065708001518

[26] W. Barbakh and C. Fyfe, "Local vs global interactions in clustering algorithms: Advances over K-means," *Int. J. Knowledge-Based Intell. Eng. Syst.*, vol. 12, no. 2, pp. 83–99, May 2008.

[27] S. Lloyd, "Least squares quantization in PCM," *IEEE Trans. Inf. Theory*, vol. 28, no. 2, pp. 129–137, Mar. 1982.

[28] D. Sculley, "Web-scale K-means clustering," in *Proc. 19th Int. Conf. World Wide Web (WWW)*, 2010, pp. 1177–1178.

[29] D. Arthur and S. Vassilvitskii, "K-means++: The advantages of careful seeding," in *Proc. 18th Annu. ACM-SIAM Symp. Discrete Algorithms*. Philadelphia, PA, USA: SIAM, 2007, pp. 1027–1035.

[30] A. Coates, A. Ng, and H. Lee, "An analysis of single-layer networks in unsupervised feature learning," in *Proc. 14th Int. Conf. Artif. Intell. Statist.* (Proceedings of Machine Learning Research), vol. 15, G. Gordon, D. Dunson, and M. Dudík, Eds. Fort Lauderdale, FL, USA: PMLR, Apr. 2011, pp. 215–223. [Online]. Available: https://proceedings.mlr.press/v15/coates11a.html

[31] F. Triefenbach, A. Jalalvand, B. Schrauwen, and J. P. Martens, "Phoneme recognition with large hierarchical reservoirs," in *Proc. Adv. Neural Inf. Process. Syst.* Red Hook, NY, USA: Curran Associates, 2010, pp. 2307–2315. [Online]. Available: http://papers.nips.cc/paper/4056-phoneme-recognition-with-large-hierarchical-reservoirs.pdf

[32] P. Steiner, S. Stone, P. Birkholz, and A. Jalalvand, "Multipitch tracking in music signals using echo state networks," in *Proc. 28th Eur. Signal Process. Conf. (EUSIPCO)*, Jan. 2021, pp. 126–130. [Online]. Available: https://www.eurasip.org/Proceedings/Eusipco/Eusipco2020/pdfs/0000126.pdf

[33] P. Steiner, A. Jalalvand, S. Stone, and P. Birkholz, "Feature engineering and stacked echo state networks for musical onset detection," in *Proc. 25th Int. Conf. Pattern Recognit. (ICPR)*, Jan. 2021, pp. 9537–9544.

[34] P. Steiner, S. Stone, and P. Birkholz, "Note onset detection using echo state networks," in *Studientexte zur Sprachkommunikation: Elektronische Sprachsignalverarbeitung*, R. Böck, I. Siegert, and A. Wendemuth, Eds. Dresden, Germany: TUDpress, 2020, pp. 157–164.

[35] W. Maass, "Liquid state machines: Motivation, theory, and applications," in *Computability in Context: Computation and Logic in the Real World*, 2011, pp. 275–296.

[36] A. Jalalvand, G. Van Wallendael, and R. Van De Walle, "Real-time reservoir computing network-based systems for detection tasks on visual contents," in *Proc. 7th Int. Conf. Comput. Intell., Commun. Syst. Netw.*, Jun. 2015, pp. 146–151.

[37] X. Hinaut and N. Trouvain, "Which hype for my new task? Hints and random search for echo state networks hyperparameters," in *Artificial Neural Networks and Machine Learning (ICANN)*, I. Farkaš, P. Masulli, S. Otte, and S. Wermter, Eds. Cham, Switzerland: Springer, 2021, pp. 83–97.

[38] H. Jaeger, M. Lukoševičius, D. Popovici, and U. Siewert, "Optimization and applications of echo state networks with leaky- integrator neurons," *Neural Netw.*, vol. 20, no. 3, pp. 335–352, Apr. 2007.

[39] M. K. Pakhira, "A linear time-complexity K-means algorithm using cluster shifting," in *Proc. Int. Conf. Comput. Intell. Commun. Netw.*, Nov. 2014, pp. 1047–1051.

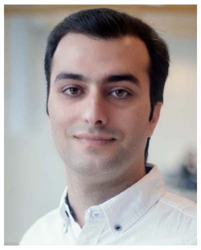
[40] O. Bachem, M. Lucic, S. H. Hassani, and A. Krause, "Approximate K-means++ in sublinear time," in *Proc. AAAI Conf. Artif. Intell.*, Feb. 2016, vol. 30, no. 1, pp. 1459–1467. [Online]. Available: https://ojs.aaai.org/index.php/AAAI/article/view/10259

[41] 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.*, vol. 32, no. 5, pp. 2169–2179, May 2021.

[42] D. Wang and M. Li, "Stochastic configuration networks: Fundamentals and algorithms," *IEEE Trans. Cybern.*, vol. 47, no. 10, pp. 3466–3479, Oct. 2017.

Peter Steiner received the master's (Dipl.Ing.) degree from Technische Universität Dresden (TUD), Dresden, Germany, in 2017, where he is currently pursuing the Ph.D. degree with the Institute of Acoustics and Speech Communication.

His research interests include signal processing, machine learning, and its application to different kinds of audio, visual, and multisensor signals. He is in particular working with reservoir computing and is thereby interested in improving these techniques using unsupervised learning methods.



Azarakhsh Jalalvand (Member, IEEE) is currently a Senior Data Scientist with Ghent University, Ghent, Belgium. His research interests include data-driven discovery research tracks, including audio, visual, and radar data analysis, and multisensor signal processing for variety of applications, such as object recognition, surveillance, predictive maintenance, and anomaly detection.

Mr. Jalalvand received a three-year Special Postdoctoral Fellowship Award (UGent-BOF) in 2020 to investigate data-driven models for condition monitoring and plasma control in the magnetic confinement devices to produce controlled thermonuclear fusion power.

Peter Birkholz (Member, IEEE) was born in Rostock, Germany, in 1978. He received the Diploma degree in computer science and the Ph.D. degree (Hons.) in signal processing from the University of Rostock, Rostock, in 2002 and 2005, respectively.

He worked as a Research Associate with the University of Rostock from 2005 to 2009, and the Department of Phoniatrics, Pedaudiology, and Communication Disorders, RWTH Aachen University, Aachen, Germany, from 2009 to 2014. He became a Junior Professor of cognitive systems at TU Dresden, Dresden, Germany, in 2014, and a Full Professor of speech technology and cognitive systems at TU Dresden in 2020. His research interests include speech production, articulatory speech synthesis, computational neuroscience, silent speech interfaces, and measurement techniques for speech research.

Dr. Birkholz was awarded the Joachim-Jungius Prize in 2006 by the University of Rostock for his dissertation on articulatory speech synthesis, and the Klaus-Tschira Award for Achievements in Public Understanding of Science in 2006.