Cluster-based Input Weight Initialization for Echo State Networks

Peter Steiner, Azarakhsh Jalalvand *Member, IEEE,* 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 whilst 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*—Echo State Networks, Reservoir Computing, Clustering, Unsupervised pretraining .**

I. Introduction

Since the breakthrough of 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 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]–[8] have analyzed an ESN with optimized hyper-parameters and determined the impact of the different hyper-parameters. 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 hyper-parameters 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 whilst reducing the training complexity by utilizing smaller reservoirs in each layer.

This research was financed by Europäischer Sozialfonds (ESF), the Free State of Saxony (Application number: 100327771), and Special Research Fund of Ghent University (BOF19/PDO/134). *(Corresponding author: Peter Steiner)*

P. Steiner and P. Birkholz are with the Institute for 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, Belgium, as well as Mechanical and Aerospace Engineering department, Princeton University, USA (email: azarakhsh.jalalvand@ugent.be)

Manuscript received MONTH XX, 2021; revised MONTH XX, 2021.

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 behaviour 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 that 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 pre-trained 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 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 pre-trained.

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 paper, 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:

- The pre-trained ESNs perform equally well or better than ESNs without pre-training 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.
- We show that the same hyper-parameter optimization strategy proposed in [6], [7] for conventional ESNs can be applied to optimize the hyper-parameters of the novel K-Means-based initialized ESN (KM-ESN).
- Applying 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.
- 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 remainder of this paper 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

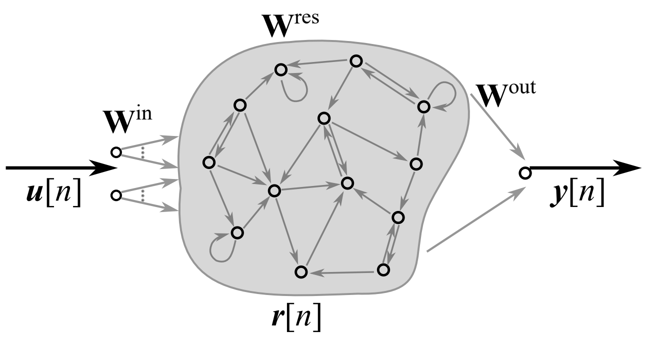


Fig. 1. Main components of a basic ESN: The input features $\mathbf{u}[n]$ are fed into the reservoir using the fixed input weight matrix $\mathbf{W}^{\text{in}}$. The reservoir consists of unordered neurons, sparsely inter-connected via the fixed reservoir matrix $\mathbf{W}^{\text{res}}$. The output $\mathbf{y}[n]$ 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 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 afterwards using the input scaling factor α_{u}, which is a hyper-parameter 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.

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 fulfil 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 to the present input. These hyper-parameters 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 hyper-parameter 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:

$$\begin{aligned}\mathbf{r}[n] =& (1-\lambda)\mathbf{r}[n-1] + \\ & \lambda f_{\text{res}}(\mathbf{W}^{\text{in}}\mathbf{u}[n] + \mathbf{W}^{\text{res}}\mathbf{r}[n-1] + \mathbf{w}^{\text{bi}})\end{aligned} \tag{1}$$

Equation (1) is a leaky integration of the reservoir neurons, which is equivalent to a first-order lowpass 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 non-linearity of the system. Conventionally, the sigmoid or $\tanh$ functions are used, because their lower and upper boundaries facilitate the reservoir states 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 Equation (2).

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

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 Equation (3), 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 closed form, ESNs are quite efficient and fast to train compared to 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 k-th set S_k, which is usually the mean of all points belonging to S_k.

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

C. Novel input weight initialization

In this paper, 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 Equation (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 ($K^{rec} = 0$) and with a linear activation function f_{res}. Thus, we can simplify Equation (1) to Equations (5) and (6) for the k^{th} reservoir neuron.

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

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

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

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

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

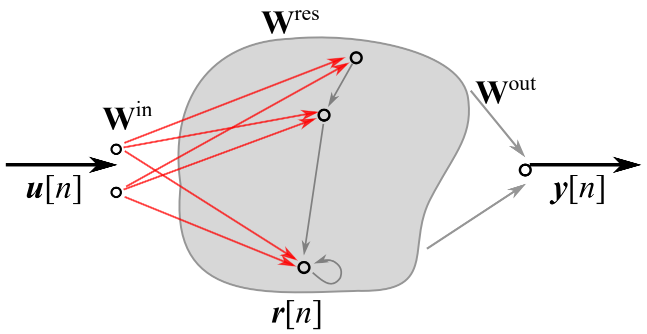
The input weights of an ESN are responsible for passing feature vectors to the reservoir that consists of non-linear neurons. Due to the – typically – random initialization of the input weights, several linear combinations of the input features for different neurons in the reservoir can be computed. In this paper, 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 tasks.

Thus, we propose to replace the randomly initialized input weights by the cluster centroids obtained from the K-means algorithm, i.e., $\mathbf{w}^{\text{in}}_k = \mu_k$. 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]. This way, passing feature vectors to the ESN basically consists of computing the cosine similarity between the centroids and the feature vectors.

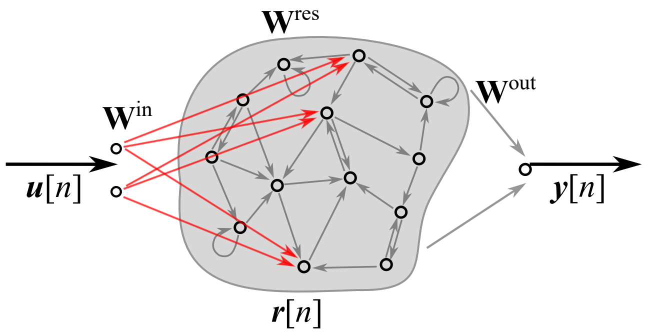
Typically, the reservoir size N^{res} is increased after tuning the hyper-parameters using small ESNs. We have been shown that this final step significantly improves the classification results [6], [7], [31]–[34]. However, if we would simply increase the reservoir size in our novel ESN model, we needed 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 paper, 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 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, very large K-Means-based ESNs are even getting biologically plausible.

Figure 2 visualizes two very simple examples for the proposed K-Means-based ESN architectures. Fig. 2a shows the case $K = N^{\text{res}} = 3$, where the number of centroids is equal to the reservoir size. Figure 2b 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, whilst most of the neurons are only connected to other neurons inside the reservoir.



(a) K-Means-based ESN when $K = N^{\text{res}} = 3$.



(b) K-Means-based ESN when $K < N^{\text{res}}$.

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

In the remainder of this paper, 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 open, closed or half-open from a low resolution camera sensor. Using a large-scale dataset, we illustrate the impact of the K-Means clustering on the hyper-parameters.

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 semi-automatically 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 minute. The first half of each movie was used as training set (N) and the latter one as the test set. To optimize the hyper-parameters, we only used the first movie, whose training set was partitioned in five folds for cross validation. We sequentially optimized the hyper-parameters 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 pre-processing steps and directly used the rescaled vectors as input for the ESN models. The training set contained ≈ 76 minutes of data leading to $N_{\text{samples}} = 415\,620$ frames in total.

C. Target preparation and readout post-processing

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 three-dimensional 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 hyper-parameters, 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, Eq. 8) was used.

$$\text{FER} = \frac{\text{N}_{\text{error}}}{\text{N}_{\text{frames}}} \, , \tag{8}$$

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 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 and to search for the point when the slope of the SSE gets less steep. From Figure 3, where the summed squared error SSE is visualized over K for the training set, we observed that the SSE decreased quite fast until $K \approx 50$. Afterwards, 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 hyper-parameter optimization, we first of all fixed the number of centroids to $K = 50$ and later increased it together with the reservoir size.

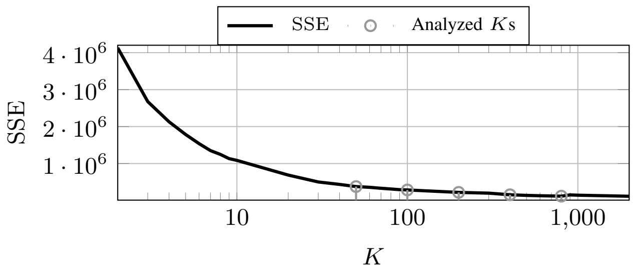


Fig. 3. Summed Squared Error (SSE) for different numbers of centroids. A fast decrease was observed until $K \approx 20$. Afterwards, SSE still continued to decrease more slowly.

F. Optimization of the hyper-parameters of the ESNs

Before optimizing the hyper-parameters 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 Figure 4 as follows: Each neuron has a 900-dimensional input weight vector. We have reshaped them into a 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:

- The input weights of the basic ESN (Fig. 4a) are very sparse ($0.06\,\%$ non-zero values) and have uniformly distributed non-zero values between ± 1.
- Figure 4a shows that images are randomly fed in the basic ESN.
- The KM-ESN (Fig. 4b) is dense and the values are approximately distributed between 0 and 0.25. This sparseness is caused by the dark regions in the frames.
- Fig. 4b 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.

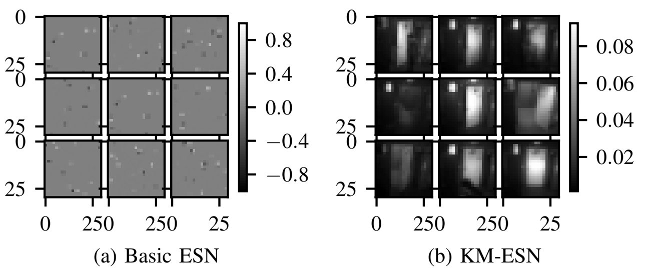


Fig. 4. Input weights for both ESN models. In case of the basic ESN (Fig. 4a), every neuron received ≈ 10 randomly selected input features and thus has ≈ 890 zero connections. The remaining values are uniformly distributed between ± 1. In case of the KM-ESN (Fig. 4b), the weights were learned from the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

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. 4b.

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 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 non-negative. For the following hyper-parameter optimization, the different distributions of the input weights will lead to significant differences between the hyper-parameters of the basic ESN and of the KM-ESN, especially for the input scaling and spectral radius.

In order to optimize the hyper-parameters of both basic and KM-ESN, we followed the sequential optimization approach introduced in [32], [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 hyper-parameter 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 $\alpha_{bi} = 0$.

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 5-fold cross validation is depicted in Fig. 5. To better present the results, we have visualized them using the expression $\min(0.15, \text{MSE})$. We achieved the lowest validation MSE with $(\alpha_{\text{u}}, \rho) = (0.85, 0.05)$

for the basic ESN and $(\alpha_{\mathrm{u}}, \rho) = (0.05, 0.08)$ for the KM-ESN. Comparing the optimization results in Fig. 5, the general behavior of these basic and KM-ESNs is different. In particular, there are two marginal differences: The input scaling is lower in case of the KM-ESN, whereas the spectral radii are similar. The area of the best hyper-parameters in case of the KM-ESN is much smaller than the area in case of a basic ESN.

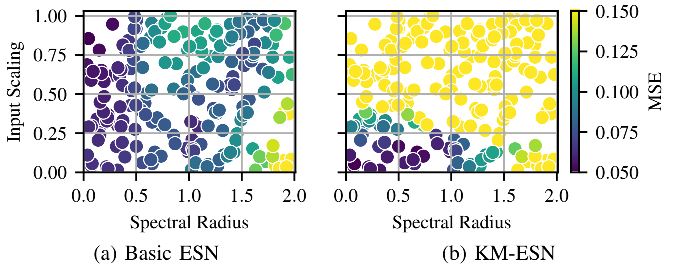


Fig. 5. MSE after the joint optimization of the input scaling α_{u} and the spectral radius ρ. The overall behaviour of the basic ESN and the KM-ESN is similar. The input scaling of the basic ESN (0.85) is very large compared to the one of the KM-ESN (0.05).

The differences in the input scaling values are caused by the different of 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 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 hyper-parameter to be optimized was the leakage λ. Again, we used a random search to optimize this hyper-parameter for the basic ESN and for the KM-ESN. This time, we used a logarithmic uniform distribution between 1×10^{-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 behaviour 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 lowpass filter [38] and smooths the reservoir states.

The last hyper-parameter 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.

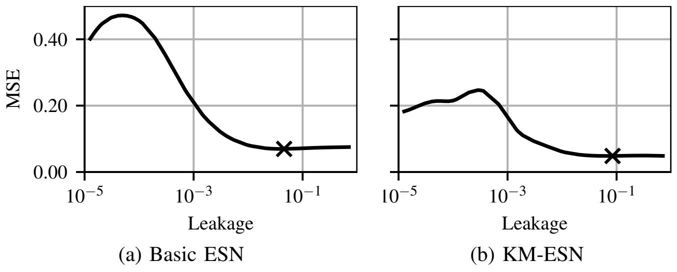


Fig. 6. MSE after the optimization of the leakage λ. The global behaviour of the basic ESN and the KM-ESN is similar. As indicated by the crosses, the optimized leakage values of the basic ESN and of the KM-ESN are 0.05 and 0.08, respectively.

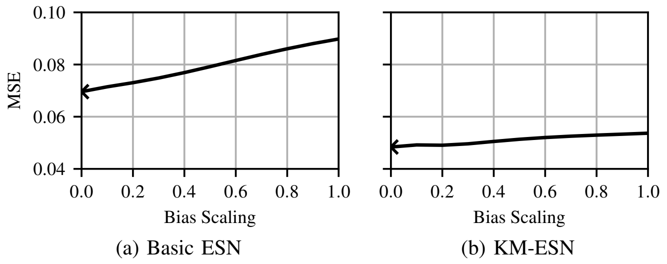


Fig. 7. MSE after the optimization of the bias scaling α_{bi}. The impact of bias scaling is rather small compared to the impact of the previous hyper-parameters. As indicated by the crosses, the best performance was achieved with $\alpha_{\mathrm{bi}} = 0$ in both cases.

G. Impact of the reservoir size

As mentioned in the introduction, ESNs typically benefit from increasing the reservoir size after fixing the other hyper-parameters. 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 hyper-parameter optimization for these sparsely connected KM-ESNs.

The optimized hyper-parameters 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 behaviour, the optimal hyper-parameters have changed. In particular, it is interesting that the spectral radius strongly increased by a factor of ten while input scaling increased only by a factor of three. This means that especially the sparse KM-ESN not only uses the current input to compute the output but also needs the memory provided by the recurrent connections. Less smoothing by the leaky integration is required now.

TABLE I
OPTIMIZED HYPER-PARAMETERS FOR THE BASIC ESN, SMALL KM-ESN ($N_{res} < 200$ AND DENSE INPUT LAYER) AND LARGE KM-ESN ($N_{res} > 200$ AND SPARSE INPUT LAYER) AFTER 5-FOLD 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}

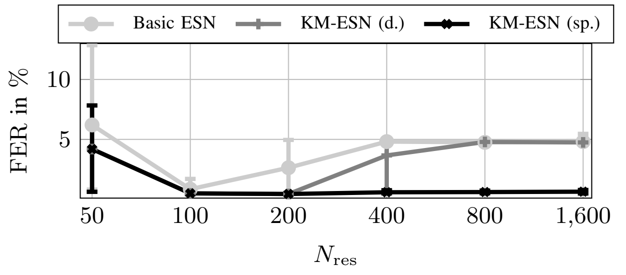


Fig. 8. Mean, minimum and maximum Frame Error Rate (FER) for different reservoir sizes after ten random initializations. The KM-ESN always performed equally well or better than the basic ESN for all reservoir sizes. The vertical bars indicate the minimum and maximum FER of the respective models.

In Figure 8, the final FER computed on the test set for different ESN architectures are visualized. Overall, for both basic ESN and 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 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 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 case of large basic ESNs, it is likely that many static pixels are selected by the input weights, and in 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 case of large sparse KM-ESNs and further improved to FER $= 0.44\,\%$ for $K = 200$ and $N^{\mathrm{res}} = 400$. Reconsidering Table I, we can say that large reservoirs benefit from the memory incorporated by means of the high spectral radius. For $N^{\mathrm{res}} = 1600$, the FER slightly increased towards $0.64\,\%$. Overall, the KM-ESN (regardless dense or sparse) was always more robust against ten different random initalizations 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 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 show that the K-Means algorithm adds to the complexity, but needs to be performed only once before the hyper-parameter optimization.

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 HYPER-PARAMETER OPTIMIZATION.

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

IV. EXPERIMENT 2: MULTI-DATASET 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 multi-class tasks.

The datasets are by default split in training and test subsets. As before, for hyper-parameter optimization, we partitioned the training set in five folds for cross validation and then again sequentially optimized the hyper-parameters as in [32] and in the previous experiment. Since the datasets "CMU subject 16", "Kick vs. Punch" and "Walk vs. Run" contained only very few training sequences, we used 3-fold cross validation to optimize the hyper-parameters for these tasks.

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

TABLE III
DETAILS ABOUT THE DATASETS FOR THE MULTI-DATASET EVALUATION WITH THE NUMBER OF INPUT FEATURES (N^{in}), NUMBER OF OUTPUTS ($\mathrm{N}^{\mathrm{out}}$), 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

B. Feature extraction

Most of the datasets were already pre-processed 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 post-processing

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 *recognised* 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 hyper-parameters, the Classification Error Rate CER (9) was used.

$$\mathrm{CER} = \frac{\mathrm{N}_{\mathrm{error}}}{\mathrm{N}_{\mathrm{seq}}} \ , \tag{9}$$

where $\mathrm{N}_{\mathrm{error}}$ and $\mathrm{N}_{\mathrm{seq}}$ 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 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 hyper-parameters of the ESNs

The optimization of the ESN models followed the same strategy as in Sec. 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 hyper-parameters, during which we minimized the cross validation MSE.

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

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

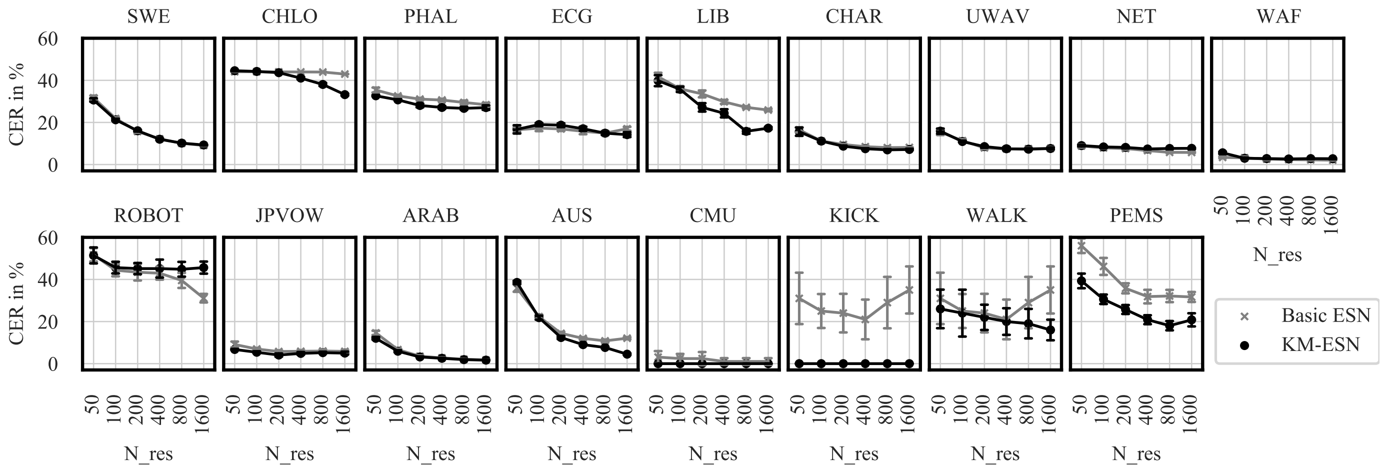


Fig. 9. Multi-dataset evaluation: Mean and standard deviation of the CER of the basic ESN and of the KM-ESN. For each dataset, the models were ten times randomly initialized, trained and evaluated. The KM-ESN outperformed the basic ESN in many datasets, both the basic ESN and the KM-ESN performed equivalently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

G. Results

Fig. 9 shows the mean value and standard deviation of CER for the different datasets for different reservoir sizes and 10 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, 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 initalizations.

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 two-dimensional 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 to the basic ESN in 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 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 risk of overfitting.

In case of the CHLO and the PHAL datasets, the KM-ESN slightly outperformed the basic ESN. Since these datasets have one-dimensional 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 one-dimensional features of the SWE dataset are more normal distributed. Thus, the resulting centroids of the K-Means algorithm are concentrated at the maximum of the distribution.

In case of the ROBOT dataset, the performance of the KM-ESN was relatively low compared to 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 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. Conclusions and Outlook

We presented an effective way to initialize the input weights of Echo State Networks 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 to 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 case of the multi-dataset 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 towards 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 time-series. Another follow-up work would be to investigate ways for pre-training 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].

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

References

[1] H. Jaeger, "The "echo state" approach to analysing and training recurrent neural networks," German National Research Center for Information Technology, Tech. Rep. GMD Report 148, 2001. [Online]. Available: http://www.faculty.iu-bremen.de/hjaeger/pubs/EchoStatesTechRep.pdf

[2] M. C. Ozturk, D. Xu, and J. C. Príncipe, "Analysis and Design of Echo State Networks," *Neural Computation*, vol. 19, no. 1, pp. 111–138, 2007. [Online]. Available: https://doi.org/10.1162/neco.2007.19.1.111

[3] M. Lukoševičius, H. Jaeger, and B. Schrauwen, "Reservoir Computing Trends," *KI – Künstliche Intelligenz*, vol. 26, no. 4, pp. 365–371, 2012.

[4] S. Scardapane and D. Wang, "Randomness in neural networks: an overview," *WIREs Data Mining and Knowledge Discovery*, vol. 7, no. 2, p. e1200, 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 Networks*, vol. 20, no. 3, pp. 365–376, 2007, echo State Networks and Liquid State Machines. [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," *Computer Speech & Language*, vol. 30, no. 1, pp. 135–158, 2015.

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

[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. 58 012–58 029, 10 2018.

[9] P. V. Aceituno, G. Yan, and Y.-Y. Liu, "Tailoring Echo State Networks for Optimal Learning," *iScience*, vol. 23, no. 9, p. 101440, 2020.

[10] C. Gallicchio, A. Micheli, and L. Pedrelli, "Deep reservoir computing: A critical experimental analysis," *Neurocomputing*, vol. 268, pp. 87–99, 2017, advances in artificial neural networks, machine learning and computational intelligence.

[11] ——, "Design of Deep Echo State Networks," *Neural Networks*, vol. 108, pp. 33–47, 2018.

[12] C. Gallicchio and S. Scardapane, *Deep Randomized Neural Networks*. Cham: Springer International Publishing, 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," *Physical Review Letters*, vol. 120, p. 024102, 01 2018. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevLett.120.024102

[14] A. Rodan and P. Tino, "Minimum Complexity Echo State Network," *IEEE Transactions on Neural Networks*, vol. 22, no. 1, pp. 131–144, 01 2011.

[15] T. Strauss, W. Wustlich, and R. Labahn, "Design Strategies for Weight Matrices of Echo State Networks," *Neural Computation*, vol. 24, no. 12, pp. 3246–3276, 2012, pMID: 22970872. [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: An Interdisciplinary Journal of Nonlinear Science*, vol. 29, no. 12, p. 123108, 2019. [Online]. Available: https://doi.org/10.1063/1.5120710

[17] T. L. Carroll and L. M. Pecora, "Network structure effects in reservoir computers," *Chaos: An Interdisciplinary Journal of Nonlinear Science*, vol. 29, no. 8, p. 083130, 2019. [Online]. Available: https://doi.org/10.1063/1.5097686

[18] L. Appeltant, M. C. Soriano, G. V. der Sande, J. Danckaert, S. Massar, J. Dambre, B. Schrauwen, C. R. Mirasso, and I. Fischer, "Information processing using a single dynamical node as complex system," *Nature communications*, vol. 2, no. 1, pp. 1–6, 2011.

[19] M. Lukoševičius, "On self-organizing reservoirs and their hierarchies," 2010.

[20] S. Basterrech, C. Fyfe, and G. Rubino, "Self-Organizing Maps and Scale-Invariant Maps in Echo State Networks," in *2011 11th International Conference on Intelligent Systems Design and Applications*, 11 2011, pp. 94–99.

[21] S. Basterrech and V. Snášel, "Initializing Reservoirs with Exhibitory and Inhibitory Signals Using Unsupervised Learning techniques," in *Proceedings of the Fourth Symposium on Information and Communication Technology*, ser. SoICT '13. New York, NY, USA: Association for Computing Machinery, 2013, pp. 53—60. [Online]. Available: https://doi.org/10.1145/2542050.2542087

[22] A. Lazar, G. Pipa, and J. Triesch, "SORN: a self-organizing recurrent neural network," *Frontiers in Computational Neuroscience*, vol. 3, p. 23, 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, no. 7, pp. 1159–1171, 2008, progress in Modeling, Theory, and Application of Computational Intelligence. [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," *International Journal of Signal Processing, Image Processing and Pattern Recognition*, vol. 9, no. 5, pp. 15–24, 2016.

[25] W. Barbakh and C. Fyfe, "ONLINE CLUSTERING ALGORITHMS," *International Journal of Neural Systems*, vol. 18, no. 03, pp. 185–194,

2008, pMID: 18595148. [Online]. Available: https://doi.org/10.1142/S0129065708001518

[26] ——, “Local vs global interactions in clustering algorithms: Advances over K-means,” *International Journal of Knowledge-based and Intelligent Engineering Systems*, vol. 12, no. 2, pp. 83–99, 2008.

[27] S. Lloyd, “Least squares quantization in PCM,” *IEEE Transactions on Information Theory*, vol. 28, no. 2, pp. 129–137, 03 1982.

[28] D. Sculley, “Web-scale k-means clustering,” in *Proceedings of the 19th international conference on World wide web*, 2010, pp. 1177–1178.

[29] D. Arthur and S. Vassilvitskii, “K-means++: the advantages of careful seeding,” in *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete Algorithms*. Society for Industrial and Applied Mathematics Philadelphia, PA, USA, 2007, pp. 1027—-1035.

[30] A. Coates, A. Ng, and H. Lee, “An Analysis of Single-Layer Networks in Unsupervised Feature Learning,” in *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, ser. Proceedings of Machine Learning Research, G. Gordon, D. Dunson, and M. Dudík, Eds., vol. 15. Fort Lauderdale, FL, USA: PMLR, 04 2011, pp. 215–223. [Online]. Available: https://proceedings.mlr.press/v15/coates11a.html

[31] F. Triefenbach, A. Jalalvand, B. Schrauwen, and J. pierre Martens, “Phoneme Recognition with Large Hierarchical Reservoirs,” in *Advances in Neural Information Processing Systems 23*. Curran Associates, Inc., 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 *2020 28th European Signal Processing Conference (EUSIPCO)*, 2020, 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 *2020 25th International Conference on Pattern Recognition (ICPR)*, 2020, pp. 9537–9544.

[34] P. Steiner, S. Stone, and P. Birkholz, “Note Onset Detection using Echo State Networks,” in *Studientexte zur Sprachkommunikation: Elektronische Sprachsignalverarbeitung 2020*, R. Böck, I. Siegert, and A. Wendemuth, Eds. TUDpress, Dresden, 2020, pp. 157–164.

[35] W. Maass, “Liquid state machines: motivation, theory, and applications,” *Computability in context: computation and logic in the real world*, pp. 275–296, 2011.

[36] A. Jalalvand, G. V. Wallendael, and R. V. D. Walle, “Real-Time Reservoir Computing Network-Based Systems for Detection Tasks on Visual Contents,” in *2015 7th International Conference on Computational Intelligence, Communication Systems and Networks*, 06 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 2021*, I. Farkaš, P. Masulli, S. Otte, and S. Wermter, Eds. Cham: Springer International Publishing, 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 Networks*, vol. 20, no. 3, pp. 335–352, 2007, echo State Networks and Liquid State Machines.

[39] M. K. Pakhira, “A Linear Time-Complexity K-Means Algorithm Using Cluster Shifting,” in *2014 International Conference on Computational Intelligence and Communication Networks*, 11 2014, pp. 1047–1051.

[40] O. Bachem, M. Lucic, S. H. Hassani, and A. Krause, “Approximate K-Means++ in Sublinear Time,” *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 30, no. 1, 02 2016. [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 Transactions on Neural Networks and Learning Systems*, vol. 32, no. 5, pp. 2169–2179, 05 2021.

[42] D. Wang and M. Li, “Stochastic Configuration Networks: Fundamentals and Algorithms,” *IEEE Transactions on Cybernetics*, vol. 47, no. 10, pp. 3466–3479, 10 2017.

Appendix A
Hyperparameter optimization

Since the hyperparameter optimization explained in Sec. 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 compared whether it can be used for the KM-ESN as well.

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

1) Sequential optimization as described in Sec. III-F using the parameters in Table IV. Only α_{bi} was optimized with a 1D 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 5-fold cross validation, each parameter combination (regardless sequential or joint optimization) was evaluated five times.

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

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

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

TABLE V
OPTIMIZED HYPER-PARAMETERS AND FINAL LOSS VALUES FOR THE BASIC ESN AND FOR THE DENSE KM-ESN ($N_{res} < 200$) 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}