Efficient Cross-Validation of Echo State Networks

Mantas Lukoševičius(✉) and Arnas Uselis

Kaunas University of Technology, Studentu st. 50—406, 51368 Kaunas, Lithuania
{mantas.lukosevicius,arnas.uselis}@ktu.edu

Abstract. Echo State Networks (ESNs) are known for their fast and precise one-shot learning of time series. But they often need good hyper-parameter tuning for best performance. For this good validation is key, but usually, a single validation split is used. In this rather practical contribution we suggest several schemes for cross-validating ESNs and introduce an efficient algorithm for implementing them. The component that dominates the time complexity of the already quite fast ESN training remains constant (does not scale up with k) in our proposed method of doing k-fold cross-validation. The component that does scale linearly with k starts dominating only in some not very common situations. Thus in many situations k-fold cross-validation of ESNs can be done for virtually the same time complexity as a simple single split validation. Space complexity can also remain the same. We also discuss when the proposed validation schemes for ESNs could be beneficial and empirically investigate them on several different real-world datasets.

Keywords: Echo State Networks · Reservoir computing · Recurrent neural networks · Cross-validation · Time complexity

1 Introduction

Echo State Network (ESN) [1–3] is a recurrent neural network training technique of reservoir computing type [4], known for its fast and precise one-shot learning of time series. But it often needs good hyper-parameter tuning to get the best performance. For this fast and representative validation is very important.

Validation aims to estimate how well the trained model will perform in testing. Typically ESNs are validated on a single data subset. This single training-validation split is just one shot at estimating the test performance. Doing several splits and averaging the results can make a better estimation. This is known as cross-validation ([5] is often cited as one of the earliest descriptions), as the same data can be used for training in one split and for validation in another and vice versa.

In this contribution we suggest several schemes for cross-validating ESNs and introduce an efficient algorithm for implementing them. We also test the validation schemes on five different real-world datasets.

I. V. Tetko et al. (Eds.): ICANN 2019, LNCS 11731, pp. 121–133, 2019.
https://doi.org/10.1007/978-3-030-30493-5_12

The goal of the experiments here is not to obtain the best possible performance on the datasets, but to compare different validation methods of ESNs on equal terms. The best performance here is often sacrificed for the simpler models and procedures. Therefore we use classical ESNs here, but the proposed validation schemes apply to any type of reservoirs with the same time and space complexity savings, as long as they have the same linear readouts.

We introduce our ESN model, training, and notation in Sect. 1.1, discuss different classes of tasks that might be important for validation in Sect. 2.1, discuss cross-validation nuances of time series in Sect. 2.2, suggest several cross-validation schemes for ESNs in Sect. 2.3, suggest several ways of producing the final trained model in Sect. 2.4, and introduce a time- and space-efficient algorithm for cross-validating ESNs in Sect. 2.5. We also report empirical experiments with different types of data in Sect. 3 and conclude with a discussion in Sect. 4.

1.1 Basic ESN Training

Here we introduce our ESN model and notation following [6].

The typical update equations of ESN are

$$\tilde{\mathbf{x}}(n) = \tanh\left(\mathbf{W}^{\text{in}}[1; \mathbf{u}(n)] + \mathbf{W}\mathbf{x}(n-1)\right), \tag{1}$$

$$\mathbf{x}(n) = (1-\alpha)\mathbf{x}(n-1) + \alpha\tilde{\mathbf{x}}(n), \tag{2}$$

where $\mathbf{x}(n) \in \mathbb{R}^{N_\text{x}}$ is a vector of reservoir neuron activations and $\tilde{\mathbf{x}}(n) \in \mathbb{R}^{N_\text{x}}$ is its update, all at time step n, $\tanh(\cdot)$ is applied element-wise, $[\cdot;\cdot]$ stands for a vertical vector (or matrix) concatenation, $\mathbf{W}^{\text{in}} \in \mathbb{R}^{N_\text{x}\times(1+N_\text{u})}$ and $\mathbf{W} \in \mathbb{R}^{N_\text{x}\times N_\text{x}}$ are the input and recurrent weight matrices respectively, and $\alpha \in (0, 1]$ is the leaking rate.

The linear readout layer is typically defined as

$$\mathbf{y}(n) = \mathbf{W}^{\text{out}}[1; \mathbf{u}(n); \mathbf{x}(n)], \tag{3}$$

where $\mathbf{y}(n) \in \mathbb{R}^{N_\text{y}}$ is the network output and $\mathbf{W}^{\text{out}} \in \mathbb{R}^{N_\text{y}\times N_\text{r}}$ the output weight matrix. We denote $N_\text{r} = 1 + N_\text{u} + N_\text{x}$ as the size of the "expanded" reservoir $[1; \mathbf{u}(n); \mathbf{x}(n)]$ for brevity.

Equation (3) can be written in a matrix notation as

$$\mathbf{Y} = \mathbf{W}^{\text{out}}\mathbf{X}, \tag{4}$$

where $\mathbf{Y} \in \mathbb{R}^{N_\text{y}\times T}$ are all $\mathbf{y}(n)$ and $\mathbf{X} \in \mathbb{R}^{N_\text{r}\times T}$ are all $[1; \mathbf{u}(n); \mathbf{x}(n)]$ produced by presenting the reservoir with $\mathbf{u}(n)$, both collected into respective matrices by concatenating the column-vectors horizontally over the training period $n = 1, \ldots, T$. We use here a single $\mathbf{X}$ instead of $[\mathbf{1}; \mathbf{U}; \mathbf{X}]$ for notational brevity.

Finding the optimal weights $\mathbf{W}^{\text{out}}$ that minimize the squared error between $\mathbf{y}(n)$ and $\mathbf{y}^{\text{target}}(n)$ amounts to solving a system of linear equations

$$\mathbf{Y}^{\text{target}} = \mathbf{W}^{\text{out}}\mathbf{X}, \tag{5}$$

where $\mathbf{Y}^{\text{target}} \in \mathbb{R}^{N_\text{y} \times T}$ are all $\mathbf{y}^{\text{target}}(n)$, with respect to $\mathbf{W}^{\text{out}}$ in a least-square sense, i.e., a case of linear regression. In this context $\mathbf{X}$ can be called the *design matrix*. The system is typically overdetermined because $T \gg N_\text{r}$.

The most commonly used solution to (5) in this context is ridge regression:

$$\mathbf{W}^{\text{out}} = \mathbf{Y}^{\text{target}} \mathbf{X}^{\text{T}} \left(\mathbf{X}\mathbf{X}^{\text{T}} + \beta \mathbf{I} \right)^{-1}, \tag{6}$$

where β is a regularization coefficient and $\mathbf{I}$ is the identity matrix. It is advisable to set the first element of $\mathbf{I}$ to zero to exclude the bias connection from the regularization.

For more details on generating and training ESNs see [6].

2 Validation in Echo State Networks

In this chapter we discuss the validation options for ESNs, and propose an efficient algorithm for cross-validation.

2.1 Different Tasks

Some details of implementation and computational savings depend on what type of task we are learning. Let us distinguish three types of temporal machine learning tasks:

1. **Generative** tasks, where the computed output $\mathbf{y}(n)$ comes back as (part of) the input $\mathbf{u}(n+k)$. This is often pattern generation or multi-step time series prediction in a generative mode.[1]
2. **Output** tasks, where the computed output time series $\mathbf{y}(n)$ does not come back as part of input. This is often detection or recognition in time series, or deducing a signal from other contemporary signals.
3. **Classification** tasks, of separate (pre-cut) finite sequences, where a class $\mathbf{y}$ is assigned to each sequence $\mathbf{u}(n)$.

For the latter type of tasks we usually store only an averaged or a fixed number of states $\mathbf{x}(n)$ for every sequence in the state matrix $\mathbf{X}$. It is similar to a non-temporal classification task.

The experiments Sect. 3 is structured according to this distinction.

2.2 Cross-Validation in Time Series

k-fold cross-validation, arguably the most popular cross-validation type, is a standard technique in static (non-temporal) machine learning tasks where data points are independent of each other. Here the data are partitioned into k usually

[1] Note that this can alternatively be implemented with feedback connections $\mathbf{W}^{\text{fb}}$ from $\mathbf{y}(n-1)$ to $\tilde{\mathbf{x}}(n)$ in (1).

equal folds, and k different train-validate splits of the data are done, where one (each time different) fold is used for validation and the rest for testing.

Temporal data, on the other hand, are time series or signals, often a single continuous one. They are position-dependent. Cross-validation in them is a bit less intuitive and popular.

A classical option for ESNs is the static split of the data into initialization, training, validation, and testing, in that order in time. The short initialization (also called transient) phase is used to get the state of the reservoir $\mathbf{x}(n)$ "in tune" with the input $\mathbf{u}(n)$ [6]. This sequence is finite, and often quite short, because ESNs possess the echo state property [7]. Initialization is only necessary before the first (training) phase, because the subsequent phases can take the last $\mathbf{x}(n)$ from the previous phase if data continue without gaps. In generative tasks the real (future) outputs $\mathbf{y}(n)$ are substituted with targets $\mathbf{y}^{\text{target}}(n)$ in inputs, known as "teacher forcing", to break the cyclic dependency.

Because the memory of ESN is preserved in its collected state, and the classical output that is learned is memory-less, we can do the instant switches between phases. Exploiting this same Markovian property, cross-validation with ESNs is rather straightforward. In other temporal models this can be more involved [8].

We see the following intuitive cases when using cross-validation on time series could be beneficial:

- When the data are scarce, cross-validation efficiently uses all the available data for both training and validation.
- Combining the models trained on different folds could be a form of (additional) regularization, improving stability.
- When the process generating the data are slightly non-stationary and it "wanders" around, cross-validation increases the chances that the testing interval is adequately covered by the model. However, if it "drifts" in one direction validating and tuning the hyper-parameters on the data interval adjacent to the testing one (i.e., the classic validation) might be the best option.

In particular, we do not expect cross-validation to be beneficial on stationary synthetic long time series, like chaotic attractors, as it does not matter which (and to some extent how much if the data are ample) sections are taken for training or validation.

2.3 Validation Schemes

Here we suggest several validation schemes for ESNs.

We firstly split the testing part off the end which is independent of the validation scheme and is left for testing it as illustrated in Fig. 1(b). The classical ESN validation scheme explained above is presented in (c). We will refer to it as **static validation** (SV). Here we also investigate alternative validation schemes where the data left from initialization and testing are used for training and validation differently and iteratively.

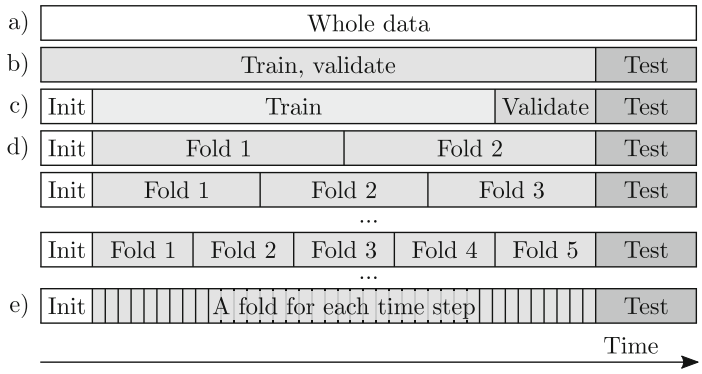


Fig. 1. Splitting the data: (a) all the available data; (b) splitting-off the testing set; (c) a static classical (SV) initialization, training, and validation split for ESNs; (d) splitting data into folds 2 and up for n-fold cross-validation; (e) the maximum amount of folds for leave-one-out cross-validation.

To investigate the k-fold cross-validation of ESNs we split the data into k-folds. The number of folds k can be varied from 2, as in Fig. 1(d), up to available data/time points ending up with leave-one-out cross-validation (e).

In addition to the classical SV split we investigate these validation schemes of using the data between the initialization and testing parts:

1. **k-fold cross-validation** (CV). The data are split into k equal folds. Training and validation are performed k times, each time taking a different single fold for validation and all the rest for training.
2. **k-fold accumulative validation** (AV). First, we split the "minimum" required amount for training only off the beginning, then we divide the rest of the data into k equal folds. Training and validation are performed k times, each time validating on a different fold, similarly to CV, but only training on all the data preceding the validation fold.
3. **k-fold walk forward validation** (FV) is similar to AV: the splitting is identical and validation is done on the same folds, but the training is each time done only on the same fixed "minimal" amount of data directly preceding the validation fold.

The three validation schemes are illustrated in the left column of Fig. 2.

Each validation scheme has some rationale behind it.

In CV all the data are used for either training or validation in each split. Also, all the data have been used exactly $k-1$ times for training and 1 time for validation. This has its benefits explained in Sect. 2.4. What is a bit unusual for temporal data here is that training data also come later in the sequence than the validation data. But this should not necessarily be considered a problem. In fact, for time series output and classification tasks the columns of $\mathbf{X}$ and $\mathbf{Y}^{\text{target}}$ could in principle be randomly shuffled, before applying k-fold cross-validation,

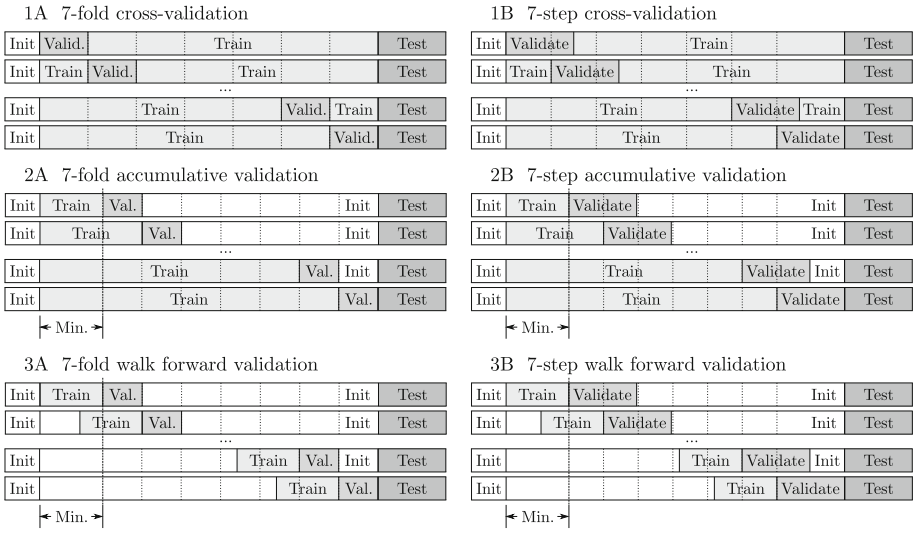


Fig. 2. Different validation schemes used.

as is common in non-temporal tasks. CV is geared towards training the model once on a fixed well-representative dataset.

AV emulates the classical static training and validation SV k times by each time training on all the available data that come before the validation fold and then validating. This scenario would also happen when the model is repeatedly updated with newly available data. Here we do not allow our model to "peek into the future". The downsides are that models are not trained on the same amount of data and more models are trained on the beginning of the data than the ending.

FV is similar to AV but keeps the training data length constant like CV, which is more consistent when selecting good hyper-parameters. Training length can be set to match several folds, which is quite common when doing walk forward validation. Here the assumption of stationarity of generating process is weaker, models are trained and validated "locally" in time.

CV, AV, and FV all use progressively less data for training thus, in general, are progressively less advisable when data are scarce.

We have also investigated counterparts for the three validation schemes with validation "folds" set longer, independent of k. They are illustrated in the right column of Fig. 2. We call them "k-step" as opposed to "k-fold". We are inventing terminology here when we could not find (a consistent) one in the literature.

These "k-step" validation schemes are mostly relevant to realistically validate generative tasks, where errors tend to escalate over time when running in the generative mode. For other types of tasks, validation length is usually not important when we compute time-averaged errors. The validation length here can be set to match a testing length in a realistic use scenario. It can also be

set to match the length of several steps, which would use data for training and validation more consistently.

2.4 Final Trained Model

Validation results are usually averaged over splits for every hyper-parameter set and the best set is selected. We investigate several ways of producing the final trained model with the best hyper-parameters for testing:

- **Retrain** ESN on all the training and validation data;
- **Average** $\mathbf{W}^{\text{out}}$s of ESNs that have been trained on the k splits;
- Select the ESN that validated **best** among the k.

Each method again has its own rationale. Retraining uses all the available data to an ESN in a straightforward way. However, this requires some additional computation and the hyper-parameters might no longer be optimal for the longer training sequence. Averaging $\mathbf{W}^{\text{out}}$s introduces additional regularization, which adds to the stability of outputs. This method might make less sense on AV, since models are trained on different amounts of data. The best-validated split, on the other hand, was likely trained on the hardest parts of the data (and validated on the easiest). A weighting scheme among splits could also be introduced when combining ESNs, as well as time step weighting as discussed in [6].

Regularization parameter is grid-searched for every split individually, as opposed to other hyper-parameters, that are searched in outside loops (see [6] for why this is efficient). For average and best ESNs with their best regularization for that particular fold can be used, whereas for retraining the regularization that is best on average (over the folds) is utilized. Regularization is most important in generative tasks.

For the classic static split SV we also have two options: to either retrain the model on the whole data or to use the validated model as it is.

2.5 Efficient Implementations

Running the ESN reservoir (1) is dominated by $\mathbf{W}\mathbf{x}$ which takes $\mathcal{O}({N_\text{x}}^2)$ operations per time step with dense $\mathbf{W}$, or $\mathcal{O}({N_\text{x}}^2 T)$ for all the data. This can be pushed down to $\mathcal{O}(N_\text{x} T)$ with sparse $\mathbf{W}$ [6] which is the same $\mathcal{O}(N_\text{r} T)$ required for collecting $\mathbf{X}$. For computing $\mathbf{W}^{\text{out}}$ (6), collecting $\mathbf{X}\mathbf{X}^\text{T}$ takes $\mathcal{O}({N_\text{r}}^2 T)$ and the matrix inversion takes $\mathcal{O}({N_\text{r}}^3)$ operations in practical implementations. Thus the whole training of ESN (dominated by collecting $\mathbf{X}\mathbf{X}^\text{T}$) is back to

$$\mathcal{O}({N_\text{r}}^2 T). \tag{7}$$

The same (7) applies to training and validating ESN, as validation itself has the same complexity as simply running it. And doing a straightforward ESN k-fold cross-validation is

$$\mathcal{O}(k{N_\text{r}}^2 T). \tag{8}$$

Space complexity can be pushed from $\mathcal{O}(N_r T)$ for $\mathbf{X}$ down to $\mathcal{O}({N_r}^2)$ when collecting $\mathbf{X}\mathbf{X}^\mathrm{T}$ and $\mathbf{Y}^\mathrm{target}\mathbf{X}^\mathrm{T}$ on the fly, which also allows ESNs to be one-shot-trained on virtually infinite time sequences [6].

However, we do not need to rerun the ESN for every split. We can collect and store the matrices $\mathbf{X}\mathbf{X}^\mathrm{T}$ and $\mathbf{Y}^\mathrm{target}\mathbf{X}^\mathrm{T}$ for the whole sequence once. Then in every split we only run the reservoir on the validation fold. Validation folds should be arranged consecutively like in Fig. 2.1A, so that after running one validation fold we can save the reservoir state $\mathbf{x}(n)$ for the next validation fold of the next split. We collect $\mathbf{X}\mathbf{X}^\mathrm{T}$ and $\mathbf{Y}^\mathrm{target}\mathbf{X}^\mathrm{T}$ on the validation fold and subtract them from the global ones to compute $\mathbf{W}^\mathrm{out}$ for the particular split. If we are doing output or classification task (see Sect. 2.1) and we can afford to store $\mathbf{X}$ of the validation fold in memory, we can reuse it to compute the validation output $\mathbf{y}(n)$ (3). If not, we need to rerun the validation fold one more time for this. *This way the ESN is rerun through the whole data only two or three times irrespective of* k.

Notice also, that the space complexity of such implementation remains $\mathcal{O}({N_r}^2)$. We could alternatively also store $\mathbf{X}\mathbf{X}^\mathrm{T}$ and $\mathbf{Y}^\mathrm{target}\mathbf{X}^\mathrm{T}$ for every fold and save one running through the data this way, by having space complexity $\mathcal{O}(k{N_r}^2)$.

The proposed method pushes down the time complexity of preparation of $\mathbf{X}\mathbf{X}^\mathrm{T}$'s in k-means cross-validation from $\mathcal{O}(k{N_r}^2 T)$ which dominates in (8), to $\mathcal{O}({N_r}^2 T)$. Adding the matrix inversions (6) which are now not necessarily dominated, the proposed more efficient implementation of ESN k-means cross-validation has time complexity

$$\mathcal{O}({N_r}^2 T + k{N_r}^3). \tag{9}$$

Thus we get a k or T/N_r time complexity speedup in a more efficient implementation (9) compared to naive (8), depending on which multiplier is smaller.

When the data sample length T is many times larger than the ESN size N_r (a typical case and a one where optimization is most relevant) and thus k such that $k < T/N_r$, we can say that *the proposed efficient implementation permits doing ESN* k*-folds cross-validation with the same time complexity as a simple one-shot validation. The space complexity can also remain the same.*

We have outlined an efficient method for ESN k-folds cross-validation (CV), but it can easily be adapted to other types of validation schemes described in Sect. 2.3.

3 Experiments

Having established that different validation schemes for ESNs are possible and can be implemented quite efficiently, in this section we test them empirically on several different time series datasets.

As mentioned before, the goal here is not to obtain the best possible performance but to compare different validation methods of simple ESNs on equal terms.

3.1 Generative Mode

We evaluate the proposed validation methods by examining multiple univariate datasets of increasing sizes:

- **Labour:** "Monthly unemployment rate in US from 1948 to 1977" dataset[2];
- **Gasoline:** "US finished motor gasoline product supplied" dataset[3];
- **Sunspots:** "Monthly numbers of sunspots, as from the World Data Center" dataset[4];
- **Electricity:** "Half-hourly electricity demand in England" dataset [9].

Lengths of these datasets and testing, validation split parameters are presented in Table 1. "Min. ratio" here is the percentage of the whole data (excluding testing) used as the minimal training length in AV, or the whole training length in FV ("Min." in Fig. 2).

Table 1. Datasets and validation setup parameters.

Dataset	Samples T	Valid, test samples	Folds, steps k	Min. ratio
Labour	360	10	34	50%
Gasoline	1355	67	18	50%
Sunspots	3177	200	10	50%
Electricity	4033	200	18	50%

For k-fold CV, initial transient length and k are chosen in such a way, that the k folds would have the same size as testing. For k-step validation variants the overlapping validation block used also have the same length as the testing range.

We use a grid search to find the best ESN hyper-parameters. Reservoir size $N_x = 50$ was chosen, and candidates of leaking rate $\alpha \in \{0.1, 0.2, 0.3, ..., 1\}$, spectral radius $\rho \in \{0.1, 0.2, 0.3, ..., 1.5\}$ and regularization degree $\beta \in \{0, 10^{-9}, 10^{-8}, ..., 1\}$ were evaluated following [6].

The experiment results are presented in Table 2. We can see that in all the experiments either FV or AV find the hyper-parameters producing best generalizing models, it is never SV or CV. The relative underperformance of CV can probably be explained by the nature of the non-stationary of the temporal data. The generating processes likely have a one-directional "drift", thus validation

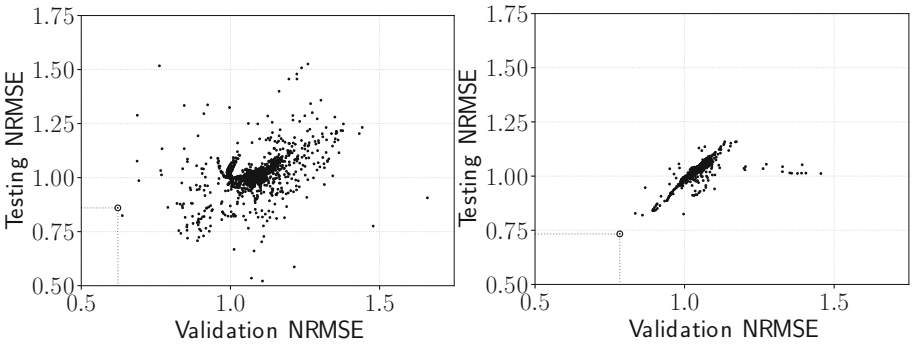
[2] Publicly available at https://data.bls.gov/timeseries/lns14000000.
[3] Publicly available at https://www.eia.gov/dnav/pet/hist/LeafHandler.ashx?n=PET&s=wgfupus2&f=W.
[4] Publicly available at http://www.sidc.be/silso/datafiles.

Table 2. Validation and testing NRMSEs on generative datasets

Method		Labour		Gasoline		Sunspots		Electricity	
Validation	Final	Valid	Test	Valid	Test	Valid	Test	Valid	Test
SV	As is	1.034	1.927	0.891	0.881	0.749	0.784	0.623	0.860
	Retrained		1.957		1.132		0.755		0.835
k-fold CV	Averaged	2.009	1.835	1.000	0.914	1.060	0.924	0.834	0.990
	Retrained		1.833		0.913		0.970		1.006
	Best		1.838		0.901		1.008		0.995
k-step AV	Averaged	1.927	4.469	1.040	0.867	0.703	0.835	0.812	0.829
	Retrained		1.171		0.962		0.742		1.006
	Best		4.546		**0.829**		0.855		0.820
k-step FV	Averaged	2.188	3.413	1.065	0.925	0.726	0.640	0.783	**0.733**
	Retrained		**0.681**		0.949		**0.612**		1.006
	Best		2.799		0.894		0.649		0.769

schemes FV and AV, that select models capable of predicting sequences directly following the training ones, win. They also win over SV, as this selection is validated over k splits instead of just one.

The main bottleneck with the Labour dataset is its scarcity: only 360 samples in total. We see that SV overfits the hyper-parameters on the single split; CV gets a better estimate; and AV and FV fail in averaged and best modes, as these use the scarce data inefficiently, but produce the very best overall results when retrained. When having more data the benefits of retraining are less evident.



(a) SV Retrained (b) k-step FV Averaged

Fig. 3. Results of grid search on Electricity dataset. Every point corresponds to one combination of hyper-parameters.

Validation vs. testing errors of two validation schemes on the Electricity dataset are illustrated in Fig. 3. We can see that while there exist outliers with very good testing performance in Fig. 3(a), they would not be picked up by the validation. In fact, k-step FV validation errors for these hyper-parameter sets were so bad, that they went off-scale in Fig. 3(b). This indicates that the lucky outliers were particular to the testing data and most likely would not do well on other. On the other hand, k-step FV gives a much better overall correlation between validation and testing errors and a much better solution based on validation is picked in Fig. 3(b) (the small circles).

3.2 Time Series Classification

To evaluate the validation methods on classification tasks we use a classical Japanese Vowels dataset[5]. This benchmark comes as 270 training samples and 370 testing samples, where each sample consists of varying length 12 LPC cepstrum coefficients taken from nine male speakers. The goal of this task is to classify a speaker based on his pronunciation of vowel /ae/. In the training set, there are 30 samples for each user, while in the testing set, this number varies from 29 to 88.

We note that 0 test set misclassifications have been previously achieved with ESNs as reported in [10]. Therefore we shift our efforts to achieve better results on models that reportedly have been performing sub-optimally. It has been reported that models that only store the last state vector $\mathbf{x}(n)$ for every speaker have at best been able to achieve 8 test misclassifications. We replicate the model used in [10], and set up a grid search as described in Sect. 3.1. We run an 18-fold cross-validation, and use a validation length of 15 instances. As the dataset consists of a permutable set of sequences (the order does not matter), we only test the classical SV and the standard k-fold CV. We run the experiments 5 times having different random initializations of the ESN weights. For each of the 5 experiments we run grid search separately and report aggregated results.

We also do individual regularization for each validation split. When doing cross-validation, each split on each regularization degree candidate is evaluated. $\mathbf{W}^{\text{out}}$s of each split with their individual best regularization degrees are averaged. We refer to this variation as "IReg Averaged". In another option, the model was retrained on the whole data using the average of the best regularization degrees. We refer to this variation as "IReg Retrained".

The NRMSE errors, misclassifications and their standard deviations are presented in Table 3. We see that all the CV variations outperform all the SV variations. We also see that the individual regularization "IReg" further slightly improves both validation and testing errors, as well as misclassification, which is not surprising. In all validation schemes, except the "CV IReg Averaged", there was at least one model produced that was able to achieve 2 test misclassifications.

[5] Publicly available at https://archive.ics.uci.edu/ml/datasets/Japanese+Vowels.

Table 3. Average results on Japanese Wovels task

Method	Final	Validation error	Test error	Misclassifications
SV	As is	0.504 ± 0.017	0.491 ± 0.005	5.0 ± 1.5
	Retrained		0.486 ± 0.003	4.8 ± 1.6
CV	Averaged	0.493 ± 0.004	0.472 ± 0.008	4.2 ± 1.8
	Retrained		0.468 ± 0.006	4.4 ± 2.1
CV IReg	Averaged	0.489 ± 0.004	**0.468** ± 0.009	4.4 ± 1.2
	Retrained		0.470 ± 0.008	**3.8** ± 1.8

4 Discussion

In this contribution we have proposed and motivated different cross-validation schemes for ESNs, have introduced a space- and time-efficient algorithm for doing this, and empirically investigated their effects on several real-world datasets.

The component that dominates the time complexity of the already quite fast ESN training remains constant (does not scale up with k) in our proposed method of doing k-fold cross-validation. The component that does scale linearly with k starts dominating only in some not very common situations, in particular when k is very large. Thus in typical situations k-fold cross-validation of ESNs can be done for virtually the same time complexity as a simple single validation. The time savings are also less evident when the data are short, but then they are also less pertinent. The methods can also have the same space complexity as a simple single validation.

This further sets apart the speed of ESN training from error backpropagation based recurrent neural network training methods, where cross-validation could also be used in principle.

We have demonstrated the proposed validation schemes for classical ESNs, but they apply to other reservoir types as well with the same time and space complexity savings. Namely, the time and space complexity of running the reservoirs is added to the ones of training instead of multiplying them by k in a k-fold/step cross-validation. The time complexity savings on reservoir running could also hold for other readout types.

We have also empirically investigated the benefits of the proposed validation schemes for ESNs on several different datasets. There is no single winner among the validation schemes. The results highly depend on the nature of the data. Overall our experiments show that typically cross-validation predicts testing errors more accurately and produces more robust results. It also can use scarce data more sparingly. AV and FV validation schemes can apparently select for good "forward-predicting" models when the generating process is "evolving" and not quite stationary. SV can also have its benefits, as it can be seen as the last fold of AV. For stationary and ample, well-representative data cross-validation might not be crucial.

How the final trained model is produced from the cross-validation results is also very important.

Acknowledgments. This research was supported by the Research, Development and Innovation Fund of Kaunas University of Technology (grant No. PP-91K/19).

References

1. Jaeger, H.: The "echo state" approach to analysing and training recurrent neural networks. Technical report GMD Report 148, German National Research Center for Information Technology (2001)
2. Jaeger, H., Haas, H.: Harnessing nonlinearity: predicting chaotic systems and saving energy in wireless communication. Science **304**(5667), 78–80 (2004)
3. Jaeger, H.: Echo state network. Scholarpedia **2**(9), 2330 (2007)
4. Lukoševičius, M., Jaeger, H.: Reservoir computing approaches to recurrent neural network training. Comput. Sci. Rev. **3**(3), 127–149 (2009)
5. Stone, M.: Cross-validatory choice and assessment of statistical predictions. J. Roy. Stat. Soc.: Ser. B (Methodol.) **36**(2), 111–133 (1974)
6. Lukoševičius, M.: A practical guide to applying echo state networks. In: Montavon, G., Orr, G.B., Müller, K.-R. (eds.) Neural Networks: Tricks of the Trade. LNCS, vol. 7700, pp. 659–686. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-35289-8_36
7. Yildiz, I.B., Jaeger, H., Kiebel, S.J.: Re-visiting the echo state property. Neural Netw. **35**, 1–9 (2012)
8. Bergmeir, C., Hyndman, R.J., Koo, B.: A note on the validity of cross-validation for evaluating autoregressive time series prediction. Comput. Stat. Data Anal. **120**, 70–83 (2018)
9. Taylor, J.W.: Short-term electricity demand forecasting using double seasonal exponential smoothing. J. Oper. Res. Soc. **54**(8), 799–805 (2003)
10. Jaeger, H., Lukoševičius, M., Popovici, D., Siewert, U.: Optimization and applications of echo state networks with leaky-integrator neurons. Neural Netw. **20**(3), 335–352 (2007)