Expert Systems With Applications 143 (2020) 113069

Contents lists available at ScienceDirect

Expert Systems With Applications

journal homepage: www.elsevier.com/locate/eswa

SOINN+, a Self-Organizing Incremental Neural Network for Unsupervised Learning from Noisy Data Streams

Chayut Wiwatcharakoses, Daniel Berrar*

Data Science Laboratory, Tokyo Institute of Technology, 2-12-1-S3-70 Ookayama, Meguro-ku, Tokyo 152-8550, Japan

ARTICLE INFO

Article history:
Received 15 July 2019
Revised 11 October 2019
Accepted 31 October 2019
Available online 2 November 2019

Keywords:
Continuous learning
Catastrophic forgetting
Self-organizing incremental neural network
Non-stationary data
Data stream
Concept drift

ABSTRACT

The goal of continuous learning is to acquire and fine-tune knowledge incrementally without erasing already existing knowledge. How to mitigate this erasure, known as *catastrophic forgetting*, is a grand challenge for machine learning, specifically when systems are trained on evolving data streams. Self-organizing incremental neural networks (SOINN) are a class of neural networks that are designed for continuous learning from non-stationary data. Here, we propose a novel method, SOINN+, which is fundamentally different from the previous generations of SOINN with respect to how "forgetting" is modeled. To achieve a more graceful forgetting, we developed three new concepts: idle time, trustworthiness, and unutility of a node. SOINN+ learns a topology-preserving mapping of the input data to a network structure, which reveals clusters of arbitrary shapes in streams of noisy data. Our experiments with synthetic and real-world data sets showed that SOINN+ can maintain discovered structures even under sudden and recurring concept drifts.

1. Introduction

A data stream is a sequence of potentially infinite, non-stationary data that become progressively available over time (Ghesmoune, Lebbah, & Azzag, 2016). Data streams often exhibit concept drift, which means that the data distribution suddenly changes over time in unforeseeable ways (Ghomeshi, Gaber, & Kovalchuk, 2019). *Continuous learning* considers adaptive algorithms capable of learning from such data streams, for example, in order to find clusters in the data (Ghesmoune et al., 2016; Parisi, Kemker, Part, Kanan, & Wermter, 2019; Parisi, Tani, Weber, & Wermter, 2017).

A key characteristic of a continuous learning system is the ability to assimilate new data without erasing previously acquired knowledge (Grossberg, 1987), i.e., without *catastrophic forgetting* or *catastrophic interference* (McCloseky & Cohen, 1989; Richardson & Thomas, 2008; Robins, 1995). This is a major challenge for machine learning, including state-of-the-art deep neural networks, because of the *stability-plasticity dilemma*: on the one hand, neural networks should be plastic (or adaptive), via changing weights, to new input data from non-stationary distributions. On the other hand, too many or too large weight changes can imply a loss of previously acquired knowledge. Thus, there is a trade-off between stability and plasticity.

Predictive models are often tailored to domain-specific, stationary environments (Bashivan et al., 2018), i.e., domains in which the process that generated the data is assumed to be stable over time. In many real-world scenarios, however, the data are generated by processes that evolve over time (Ditzler, Roveri, Alippi, & Polikar, 2015). This non-stationarity presents considerable challenges, which may also affect the learning task. For example, a model that has learned to discriminate between only two classes (e.g., images of dogs and cats) would not be able to recognize a new, third class (e.g., images of birds) without a retraining from scratch, which is extremely inefficient for learning in real time. A complete rehearsal might even be impossible when old data are no longer accessible, or when the data are simply too big (Li & Hoiem, 2016; Lomonaco & Maltoni, 2017). Ideally, a continuous learner would adapt to new classes without the need for a complete retraining by processing the data once, extracting meaningful concepts, and internally representing these concepts efficiently for future use.

Continuous learning therefore represents a long-standing goal for machine learning. This learning paradigm is also referred to as *incremental learning* and specifically *lifelong learning* (Sodhani, Chandar, & Bengio, 2018), which is reminiscent of how humans and other animals acquire and fine-tune their knowledge (Flesch, Balaguer, Dekker, Nili, & Summerfield, 2018; Parisi et al., 2019). In fact, a hallmark of human intelligence is the ability to apply knowledge from one domain to solve problems in another

* Corresponding author.
E-mail addresses: wiwatcharakoses.c.aa@m.titech.ac.jp (C. Wiwatcharakoses), daniel.berrar@ict.e.titech.ac.jp (D. Berrar).

https://doi.org/10.1016/j.eswa.2019.113069
0957-4174/

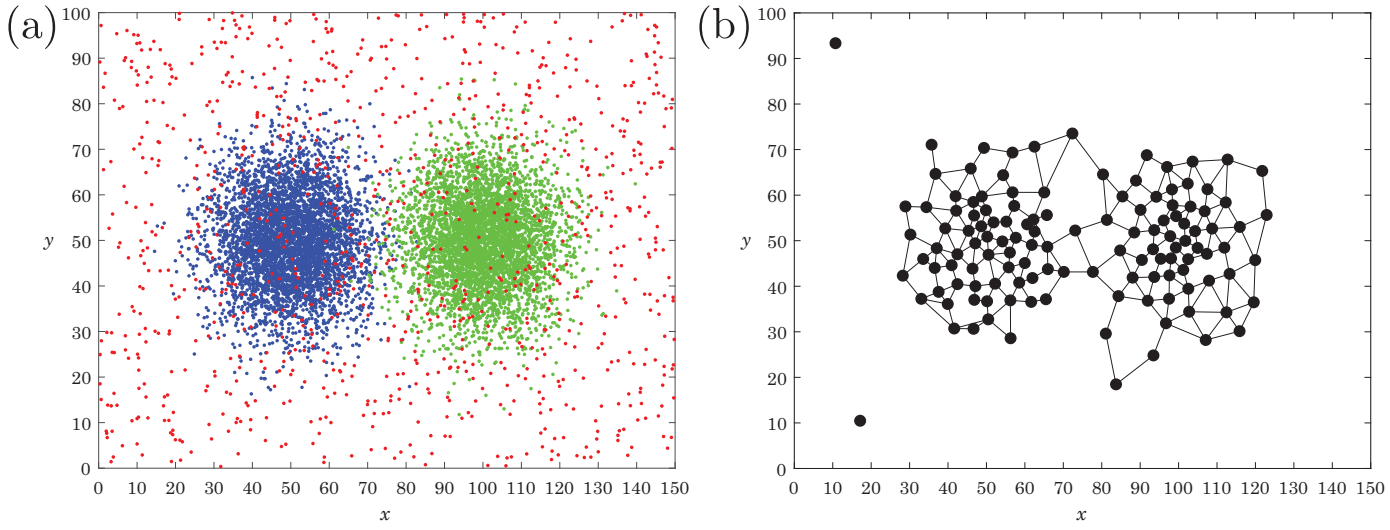

Fig. 1. (a) Feature space of the training set of 9000 normally distributed bivariate cases from two different classes (blue and green dots) and 1000 instances of noise (red dots). (b) Network after processing the training set. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

domain. Such *transfer learning* is also often regarded as a desirable property of a continuous learning system.

Self-organizing incremental neural networks (SOINN) encompass a family of neural networks that have in common that they find a topological mapping of the input data to a network structure by competitive learning (Shen & Hasegawa, 2006). In short, a SOINN maps a p-dimensional input $\mathbf{x} = (x_1, x_2, \ldots, x_p)$, where x_i is the i^{th} feature value, to a single node in an undirected graph. The mapping corresponds to a point in the p-dimensional feature space. Learning in SOINN means an adaptation of the topological map: nodes can move, merge with other nodes, remain as singletons, or be deleted, and edges between nodes can be created or deleted. A node can be thought of as a microcluster of input cases that are close to each other. Edges can be thought of as consolidated links between related nodes, for example, nodes belonging to the same (macro-)cluster.

The key characteristic of a SOINN network is its adaptability to changes of the probability distribution of the unknown data generation process. For example, consider Fig. 1a; here, the training set consists of 9000 two-dimensional, normally distributed cases from two different classes (blue and green dots) and 1000 instances of noise (red dots) (Wiwatcharakoses & Berrar, 2019). Fig. 1 shows the resulting network. Suppose that a new data stream brings cases that are affected by concept drift, for example, with an increasing variance in the y-variable. The network in Fig. 1b would then adapt accordingly, with a topology that protrudes farther in the y-dimension.[1]

Various modifications of the original SOINN have been developed for a variety of tasks, including unsupervised learning (clustering and non-parametric density estimation) (Nakamura & Hasegawa, 2017; Shen & Hasegawa, 2006; Shen, Ogura, & Hasegawa, 2007), supervised learning (Shen & Hasegawa, 2008), semi-supervised learning (Kamiya, Ishii, Shen, & Hasegawa, 2017), active learning (Shen, Yu, Sakurai, & Hasegawa, 2011), associative memory (Sudo, Sato, & Hasegawa, 2009), and various applications in robotics, including gesture recognition in humanoid robots (Okada, Kobayashi, Ishibashi, & Nishida, 2010), navigation (Kawewong, Honda, Tsuboyama, & Hasegawa, 2010), and basic language acquisition in humanoid robots through interaction with humans (He, Kojima, & Hasegawa, 2007).

Here, we present a novel self-organizing incremental neural network, called SOINN+, which inherits some ideas from its predecessors but is fundamentally different with respect to how *forgetting* is modelled. In previous generations of SOINN, batches of nodes and edges are purged at fixed, user-defined intervals, which leads to sudden changes in the network structure, as large regions of the network are deleted periodically before training resumes. This "batch forgetting" at fixed intervals conflicts with the more gradual forgetting that can usually be observed in natural cognitive systems (French, 1999). In previous SOINN, the deletion of nodes and edges is determined by two parameters, which have to be optimized for each application at hand via cross-validation or similar resampling approaches. SOINN+ addresses this shortcoming by considering the deletion of nodes and edges as an intrinsic part of the learning process. The network deletes nodes and edges if they fail to be relevant for the learning task at hand, which results in a more graceful forgetting.

The novel contributions of this paper are as follows:

1. We formulate three new concepts that are key factors for modeling "forgetting" in a self-organizing incremental neural network: idle time, trustworthiness, and (un-)utility of a node. Using these concepts, we developed a novel method, called SOINN+.
2. SOINN+ can be used to detect clusters in noisy data streams with sudden and recurring concept drifts. Using both synthetic and real-world benchmark data sets, we compared the performance of SOINN+ with that of five stream clustering methods. SOINN+ identified the real clusters according to the known ground truth, was more resilient to noise, and, importantly, could maintain the clusters even under concept drifts.

The remainder of this article is organized as follows. First, we give a brief overview of related works on continuous learning, with a focus on the various types of SOINN and their main differences. In Section 3, we describe our proposed method in detail. Section 4 reports our experimental results on both stationary and non-stationary learning. We end the paper with a discussion and outlook at future work. All source code and supplementary materials necessary to reproduce our study, including videos

[1] We provide videos showing the evolution of SOINN+ networks at the project website at https://osf.io/6dqu9/.

showing the evolution of SOINN+ networks, are available at https://osf.io/6dqu9/.

2. Related work

Among the oldest approaches to preventing catastrophic forgetting are memory-based systems that store old data for rehearsal (i.e., the continuous retraining on old data) (Robins, 1995). In the age of big data, however, the explicit storage of old data (or data streams) becomes infeasible (Lomonaco & Maltoni, 2017), and alternative methods are required. Conceptually, these methods can be categorized into (*i*) regularization methods; (*ii*) complementary learning systems; and (*iii*) dynamic architectures (Parisi et al., 2019).

Regularization methods mitigate catastrophic forgetting by penalizing weight updates and thereby try to achieve a tradeoff between the performance on old and new tasks. Kirkpatrick et al. (2017) proposed *elastic weight consolidation* (EWC) to prevent drastic weight changes in neural networks. EWC can be used for both supervised learning and reinforcement learning. EWC is similar to synaptic consolidation and enables a network to solve multiple tasks sequentially. The weights that are crucial for solving a given task are changed as little as possible when a new learning task is presented to the network.

Complementary learning systems model two different learning and memory functions of the brain: the retention of episodic memories and the generalization across different memories (McClelland, McNaughton, & O'Reilly, 1995; O'Reilly, Bhattacharyya, Howard, & Ketz, 2014). Parisi, Tani, Weber, and Wermter (2018) developed a dual-memory system consisting of two recurrent networks for realizing these two complementary tasks.

Dynamic architectures adapt to new data or learning tasks by adding additional neurons, layers, or subnetworks to the existing model. Rusu et al. (2016) developed *progressive networks*, a novel network architecture that grows new subnetworks sequentially for new tasks and creates lateral connections between these subnetworks for transfer learning. Progressive networks showed considerable performance on a wide variety of reinforcement learning tasks (Rusu et al., 2016). As a downside of their architecture, Rusu et al. (2016) mention the increasing network complexity with the number of tasks.

Yoon, Yang, Lee, and Ju Hwang (2017) proposed the *dynamically expanding network* (DEN) to learn new tasks. DEN retrains only a part of the previously trained network by selectively adapting the weights of the relevant nodes. If the selective retraining fails to achieve the desired performance, then additional nodes are added. This idea was implemented for both feedforward and convolutional neural networks, which outperformed several state-of-the-art models on three standard benchmark data sets for supervised learning.

Self-organizing incremental neural networks can be categorized as dynamic architectures. The original SOINN consists of a two-layer network (Shen & Hasegawa, 2006), where the first layer is used to learn a topological mapping of the input data onto a network of nodes. The network processes training cases sequentially, either by adding them as new nodes to the existing network or by merging them with existing nodes (and updating their location) to reflect the underlying probability density of the input data. The nodes from the first layer are used as input for the second layer, which separates clusters and eliminates nodes in low-density regions. After training, SOINN outputs prototypical nodes for each cluster.

Learning in SOINN bears similarities with *growing cell structures* (GCS) (Fritzke, 1994a), *dynamic cell structures* (Bruske & Sommer, 1995), and *growing neural gas* (GNG) (Fritzke, 1994b). In GNG, neurons are added at fixed intervals, whereas in SOINN, neurons are added only if the new signal (i.e., training case) is substantially different from existing neurons. SOINN are conceptually similar to the *growing when required* (GWR) network (Marsland, Shapiro, & Nehmzow, 2002) and its extension, the Gamma-GWR (Parisi et al., 2017). In a GWR network, new neurons are added depending on the current network activation: at time t, the activation of a GWR network is an exponential function of the negative Euclidean distance between the current training case and its nearest neighbor (i.e., the best-matching neuron in the network at time t) (Marsland et al., 2002). Like SOINN, GWR is an unsupervised learning algorithm. Parisi et al. (2017) developed the Gamma-GWR, which adds a gamma filter to the original GWR. The effect of this filter is that for the purpose of node adding, the network considers not only the current input but also its past activations. In both the original GWR and Gamma-GWR, edges that exceed a user-defined maximum age are deleted, and unconnected nodes are removed.

One of the drawbacks of GNG, GWR, and Gamma-GWR is that the maximum number of nodes has to be predefined (Hamker, 2001), which is not optimal because this number can usually not be known a priori; specifically, in continual learning tasks, this number should not be limited. By contrast, the maximum number of nodes is not fixed in SOINN because new training cases may be added as new nodes or they may be merged with existing nodes, striving for a balance between stability and plasticity. As a result, the number of nodes is typically substantially smaller than the number of training cases.

Several modifications to the original SOINN have been proposed over the years, beginning with the *enhanced SOINN* (ESOINN) (Shen et al., 2007). ESOINN uses only one layer and fewer network parameters than the original SOINN. Experimental results have shown that ESOINN is more stable. However, the final network structure of ESOINN depends on the input order of the training data. This shortcoming is addressed by the *load-balancing SOINN* (LB-SOINN), which outperformed ESOINN on both synthetic and real-world benchmark data sets (Zhang, Xiao, & Hasegawa, 2014). The next generation of SOINN is the *adjusted SOINN* (ASOINN), which is also a one-layer network (Shen & Hasegawa, 2008). Based on ASOINN, Nakamura and Hasegawa (2017) developed the *kernel density estimation SOINN* (KDESOINN), a combination of nonparametric kernel density estimation and SOINN, with the goal to learn probability density functions. KDESOINN uses the local structure of the SOINN network to determine the shape and size of each kernel at each node and then estimates the probability density function by summing up the kernels. A further extension of SOINN, called S-SOINN, was developed for semi-supervised active learning (Shen et al., 2011). During topology learning, S-SOINN attaches a class label to some nodes that are referred to as teacher nodes, for which the true class is known. A teacher node then assigns the same label to (unlabeled) nodes that belong to the same subcluster as the teacher node. S-SOINN can be used for automatic labeling of partially annotated data sets, for example.

SOINN have also been combined with deep convolutional neural networks (CNN) for image recognition. In Part and Lemon (2016), images were fed sequentially into a deep CNN to compute feature vectors, which were then input to LB-SOINN for clustering.

All previous generations of SOINN have in common that "forgetting"—the deletion of nodes and edges—occurs at user-defined intervals. Our proposed method addresses this problem by considering "forgetting" an intrinsic part of the learning process.

3. Proposed method

SOINN+ inherits the idea of adding and merging nodes from ASOINN, but it is fundamentally different with respect to how edges are created and deleted, and how nodes are deleted. To achieve a more graceful forgetting, we developed three new ideas:

idle time, trustworthiness, and (un-)utility of a node. First, we describe the notation to be used in this section. A fundamental concept is the *iteration, t*, which denotes an update cycle of the network. Each time a new training case is presented to the network, the iteration is incremented from t to $t+1$. Bold-face lower-case Roman characters denote network nodes, which are p-dimensional vectors.

t	iteration counter of network update. $t \in \mathbb{N}$
$\mathbf{a}$	node $\mathbf{a}$
$\mathbf{n}_1$	first nearest neighbor (= first winner)
$\mathbf{n}_2$	second nearest neighbor (= second winner)
$\mathbf{n}_{i,t}$	coordinates of node $\mathbf{n}_i$ at iteration t
$\mathbf{x}$	new training case
$e_{\mathbf{ab}}$	edge between nodes $\mathbf{a}$ and $\mathbf{b}$
$\parallel \mathbf{a}-\mathbf{b} \parallel$	distance between nodes $\mathbf{a}$ and $\mathbf{b}$
$\tau(\mathbf{n}_i)$	similarity threshold of node $\mathbf{n}_i$
$\bar{\tau}_1$	arithmetic mean of the similarity thresholds of the first winners that were linked to the second winners
$\bar{\tau}_2$	arithmetic mean of the similarity thresholds of the second winners that were linked to the first winners
σ_1	standard deviation of the similarity thresholds of the first winners that were linked to the second winners
σ_2	standard deviation of the similarity thresholds of the second winners that were linked to the first winners
$\mathcal{S}$	set of all nodes
$\lvert\mathcal{S}\rvert$	number of elements in $\mathcal{S}$ (i.e., cardinality of $\mathcal{S}$)
$\mathcal{E}$	set of all edges
$\mathcal{N}_1$	set of nodes with an edge to first winner $\mathbf{n}_1$
$\mathcal{A}$	set of lifetimes of edges through which the first winner node can be reached
$\mathcal{A}_{\text{del}}$	set of lifetimes of deleted edges
$\mathcal{U}$	set of unutilities of all nodes with edges
u_i	i^{th} element in $\mathcal{U}$
$\mathcal{I}$	set of all nodes without edges
$\mathcal{B}_{\text{del}}$	set of deleted nodes
$WT(\mathbf{a})$	winning time of node $\mathbf{a}$, i.e., how often $\mathbf{a}$ was selected as the first nearest neighbor. $WT(\mathbf{a}) \in \mathbb{N}$
$IT(\mathbf{a})$	idle time of $\mathbf{a}$ (i.e., counter of how many iterations $\mathbf{a}$ was *not* selected as the first winner). $IT \in \mathbb{N}$
$LT(e_{\mathbf{ab}})$	lifetime of the edge between nodes $\mathbf{a}$ and $\mathbf{b}$. $LT \in \mathbb{N}$
R_{noise}	ratio of unconnected nodes to all nodes. $R_{\text{noise}} \in \mathbb{R}_{>0}$
$T(\mathbf{a})$	trustworthiness of node $\mathbf{a}$ (= ratio of winning time of the node and max. winning time). $T \in [0,1]$
$U(\mathbf{a})$	unutility of node $\mathbf{a}$ (= ratio of idle time and winning time). $U \in \mathbb{R}_{>0}$
η	pull factor for node merging
λ_{edge}	threshold for deleting edges. $\lambda_{\text{edge}} \in \mathbb{R}_{>0}$
λ_{node}	threshold for deleting nodes. $\lambda_{\text{node}} \in \mathbb{R}_{>0}$
ω_{edge}	measure of outlierness of a lifetime. $\omega_{\text{edge}} \in \mathbb{R}_{>0}$
ω_{node}	measure of outlierness of an unutility. $\omega_{\text{node}} \in \mathbb{R}_{>0}$

3.1. Overview of SOINN+

Algorithm 1 is the pseudocode of the SOINN+ algorithm, which consists of seven subalgorithms. These subalgorithms are explained in detail in the next sections. To summarize, the network is first initialized with randomly sampled training cases (Algorithm 2). Then, for a new training case, a similarity threshold is calculated (Algorithm 3). If the condition for node adding is true, then the corresponding subalgorithm is executed (Algorithm 4) and the training case is added as a new node. Otherwise, the training case is merged with its first winner node (Algorithm 5), and the subroutines for node linking (Algorithm 6) and edge deletion (Algorithm 7) are called. Finally, irrespective of whether the new training case was added as a node or merged with an existing node, the network checks if there are any nodes that should be deleted (Algorithm 8) .

3.2. Network initialization

The network is initialized with three randomly sampled training cases (Fig. 2). The winning time of each node is set to 1.

Algorithm 1: Pseudocode of SOINN+.

Data: $\mathbf{x}$: feature vector of a new training case.

```
Network initialization (Algorithm 2)
x ← new training case
Calculation of similarity threshold (Algorithm 3)
if condition for adding node is true then
    Node adding (Algorithm 4)
else
    Node merging (Algorithm 5)
    Node linking (Algorithm 6)
    Edge deletion (Algorithm 7)
Node deletion (Algorithm 8)
```

Algorithm 2: Network initialization.

Data: $\mathbf{x}$: feature vector of a new training case (represented as new node/query case in Figure 2) and Figure 3).

```
if Initialization then
    t ← 0                                   // Iteration counter.
    S ← Set of 3 randomly selected training cases
    E ← ∅                                   // Set of all edges is empty.
    U_del ← ∅  // Set of unutilities of all deleted nodes is empty.
    B_del ← ∅          // Set of all deleted nodes is empty.
    A_del ← ∅          // Set of all deleted edges is empty.
    for all nodes in S do
        a_i ← i^th node in S
        WT(a_i) ← 1    // The winning time of each node is set to 1.
```

Algorithm 3: Calculation of similarity threshold.

```
x ← new training case
n_1 ← argmin_{b∈S} ‖ x − b ‖          // Find the node that is closest to x.
n_2 ← argmin_{b∈S\{n_1}} ‖ x − b ‖    // Find the second nearest neighbor of x.
N_i ← Set of nodes connected to the i^th winner
for i = 1, 2 do    // Calculate the similarity threshold of n_i.
    if N_i ≠ ∅ then
        τ(n_i) ← max_{a_j∈N_i} ‖ n_i − a_j ‖                    // Eq. 1
    else
        m ← second nearest neighbor of n_i
        τ(n_i) ← ‖ n_i − m ‖                                    // Eq. 1
```

Algorithm 4: Node adding.

```
condition 1 ← ‖ x − n_1 ‖ ≥ τ(n_1)                              // Eq. 2
condition 2 ← ‖ x − n_2 ‖ ≥ τ(n_2)                              // Eq. 3
if condition 1 OR condition 2 then
    S ← S ∪ {x}                          // Add x as a new node.
    WT(x) ← 1    // Initialize the winning time of x to 1.
```

Algorithm 5: Node merging.

$WT(\mathbf{n}_1) \leftarrow WT(\mathbf{n}_1) + 1$
$\mathbf{n}_1 \leftarrow \mathbf{n}_1 + \frac{\mathbf{x}-\mathbf{n}_1}{WT(\mathbf{n}_1)}$ // Eq. 4
; $\mathcal{N}_1 \leftarrow$ Set of nodes with an edge to $\mathbf{n}_1$
$\eta \leftarrow 100$ // Pull factor.
for *all nodes in* $\mathcal{N}_1$ **do**
  $\mathbf{a}_i \leftarrow i^{th}$ node in $\mathcal{N}_1$
  $\mathbf{a}_i \leftarrow \mathbf{a}_i + \frac{\mathbf{x}-\mathbf{a}_i}{\eta \cdot WT(\mathbf{n}_1)}$ // Eq. 5
  ;
$IT(\mathbf{n}_1) \leftarrow 0$ // Set the idle time of the first nearest neighbor to 0.

Algorithm 6: Node linking.

for *all nodes in* $\mathcal{S}$ **do** // Update trustworthiness.
  $\mathbf{a}_i \leftarrow i^{th}$ node in $\mathcal{S}$
  $\max(WT) \leftarrow$ maximum winning time of all nodes in $\mathcal{S}$
  $T(\mathbf{a}_i) \leftarrow \frac{WT(\mathbf{a}_i)-1}{\max(WT)-1}$ // Eq. 6
$\bar{\tau}_1 \leftarrow$ arithmetic mean of the similarity thresholds of the first winners that were linked to the second winners
$\bar{\tau}_2 \leftarrow$ arithmetic mean of the similarity thresholds of the second winners that were linked to the first winners
$\sigma_1 \leftarrow$ standard deviation of the similarity thresholds of the first winners that were linked to the second winners
$\sigma_2 \leftarrow$ standard deviation of the similarity thresholds of the second winners that were linked to the first winners
condition 1 $\leftarrow$ number of edges in network is less than 3
condition 2 $\leftarrow \tau(\mathbf{n}_1) \cdot (1 - T(\mathbf{n}_1)) < \bar{\tau}_1 + 2 \cdot \sigma_1$ // Eq. 7
condition 3 $\leftarrow \tau(\mathbf{n}_2) \cdot (1 - T(\mathbf{n}_2)) < \bar{\tau}_2 + 2 \cdot \sigma_2$ // Eq. 8
if *condition 1 OR condition 2 OR condition 3* **then** // Link nodes.
  if *no edge between first and second winner exists* **then**
    Create edge $e_{\mathbf{n}_1\mathbf{n}_2}$ between first and second winner
    Update $\overline{\tau}_1, \overline{\tau}_2, \sigma_1, \sigma_2$
if *edge between first and second winner exists* **then**
  $LT(e_{\mathbf{n}_1\mathbf{n}_2}) \leftarrow 0$ // Set the lifetime of the edge between the first and second nearest neighbor to 0.
$\mathcal{N}_1 \leftarrow$ Set of nodes with an edge to $\mathbf{n}_1$
for *all nodes in* $\mathcal{N}_1$ **do**
  $\mathbf{a}_i \leftarrow i^{th}$ node in $\mathcal{N}_1$
  $LT(e_{\mathbf{n}_1\mathbf{a}_i}) \leftarrow LT(e_{\mathbf{n}_1\mathbf{a}_i}) + 1$ // Increment the lifetime of each edge that connects to the first winner.

Algorithm 7: Edge deletion.

$\mathcal{A} \leftarrow$ Set of lifetimes of edges through which the first winner node can be reached
$\mathcal{A}_{\text{del}} \leftarrow$ Set of lifetimes of all deleted edges
$\mathcal{A}_{0.75} \leftarrow$ 75th percentile of elements in $\mathcal{A}$
$\omega_{\text{edge}} \leftarrow \mathcal{A}_{0.75} + 2 \cdot \text{IQR}(\mathcal{A})$ // Eq. 9
$\lambda_{\text{edge}} \leftarrow \bar{\mathcal{A}}_{\text{del}} \cdot \frac{|\mathcal{A}_{\text{del}}|}{|\mathcal{A}_{\text{del}}|+|\mathcal{A}|} + \omega_{\text{edge}} \cdot \left(1 - \frac{|\mathcal{A}_{\text{del}}|}{|\mathcal{A}_{\text{del}}|+|\mathcal{A}|}\right)$ // Eq. 10
$\mathcal{N}_1 \leftarrow$ Set of nodes with an edge to $\mathbf{n}_1$
for *all nodes in* $\mathcal{N}_1$ **do**
  $\mathbf{a}_i \leftarrow i^{th}$ node in $\mathcal{N}_1$
  condition 1 $\leftarrow LT(e_{\mathbf{n}_1\mathbf{a}_i}) > \lambda_{\text{edge}}$
  if *condition 1* **then**
    $\mathcal{A}_{\text{del}} \leftarrow \mathcal{A}_{\text{del}} \cup \{e_{\mathbf{n}_1\mathbf{a}_i}\}$
    delete $e_{\mathbf{n}_1\mathbf{a}_i}$

Algorithm 8: Node deletion.

for *all nodes in* $\mathcal{S}$ **do** // Update unutility.
  $\mathbf{a}_i \leftarrow i^{th}$ node in $\mathcal{S}$
  $U(\mathbf{a}_i) \leftarrow \frac{IT(\mathbf{a}_i)}{WT(\mathbf{a}_i)}$ // Eq. 11.
  condition 1 $\leftarrow \mathbf{a}_i$ has at least one edge
  if *condition 1* **then**
    $\mathcal{U} \leftarrow \mathcal{U} \cup \{U(\mathbf{a}_i)\}$
$\mathcal{I} \leftarrow$ Set of unconnected nodes
$\omega_{\text{node}} \leftarrow \text{median}(\mathcal{U}) + 2 \cdot \text{sMAD}(\mathcal{U})$ // Eq. 13.
$R_{\text{noise}} \leftarrow \frac{|\mathcal{I}|}{|\mathcal{S}|}$ // Eq. 14.
$\lambda_{\text{node}} \leftarrow \bar{\mathcal{U}}_{\text{del}} \cdot \frac{|\mathcal{B}_{\text{del}}|}{|\mathcal{B}_{\text{del}}|+|\mathcal{S}\setminus\mathcal{I}|} + \omega_{\text{node}} \cdot \left(1 - \frac{|\mathcal{B}_{\text{del}}|}{|\mathcal{B}_{\text{del}}|+|\mathcal{S}\setminus\mathcal{I}|}\right) \cdot (1 - R_{\text{noise}})$ // Eq. 15.
for *all nodes in* $\mathcal{S}$ **do**
  $\mathbf{a}_i \leftarrow i^{th}$ node in $\mathcal{S}$
  condition 1 $\leftarrow$ at least one edge exists in the network
  condition 2 $\leftarrow U(\mathbf{a}_i) > \lambda_{\text{node}}$
  condition 3 $\leftarrow \mathbf{a}_i$ has no edges
  if *condition 1 AND condition 2 AND condition 3* **then**
    delete node $\mathbf{a}_i$
    $\mathcal{B}_{\text{del}} \leftarrow \mathcal{B}_{\text{del}} \cup \{\mathbf{a}_i\}$
    $\mathcal{U}_{\text{del}} \leftarrow \mathcal{U}_{\text{del}} \cup U(\mathbf{a}_i)$
for *all nodes in* $\mathcal{S}$ **do**
  $\mathbf{a}_i \leftarrow i^{th}$ node in $\mathcal{S}$
  $IT(\mathbf{a}_i) \leftarrow IT(\mathbf{a}_i) + 1$

3.3. Calculation of similarity threshold, adding and merging nodes

A new training case can either be added as a new, unconnected node, or it can be merged with its first nearest neighbor. Which action is taken depends on a *similarity threshold*. We define the similarity threshold for a winner node $\mathbf{n}_i$, $i = 1, 2$, as follows:

$$\tau(\mathbf{n}_i) = \begin{cases} \max\limits_{\mathbf{a}_j \in \mathcal{N}_i} \| \mathbf{n}_i - \mathbf{a}_j \| & \text{if } \mathcal{N}_i \neq \varnothing \\ \min\limits_{\mathbf{a}_j \in \mathcal{S}\setminus\{\mathbf{n}_i\}} \| \mathbf{n}_i - \mathbf{a}_j \| & \text{otherwise} \end{cases} \quad (1)$$

where $\mathcal{N}_i$ is the set of nodes that are directly linked to node $\mathbf{n}_i$, and $\mathbf{a}_j$ is a node in the network. Eq. (1) means that if the node $\mathbf{n}_i$ is linked to some other nodes (i.e., $\mathcal{N}_i \neq \varnothing$), then the threshold is the maximum distance between $\mathbf{n}_i$ and these nodes. If $\mathbf{n}_i$ is not connected to any other nodes (i.e., $\mathcal{N}_i = \varnothing$), then the threshold is the distance between $\mathbf{n}_i$ and its second nearest neighbor.

When a new training case $\mathbf{x}$ is presented to the network, the network finds the first and the second nearest neighbor based on a distance measure; here, we use the Euclidean distance. The first nearest neighbor will be referred to as the first winner, $\mathbf{n}_1$, and the second nearest neighbor will be referred to as the second winner, $\mathbf{n}_2$. The case $\mathbf{x}$ is added as a new node if at least one of the following conditions holds:

$$\| \mathbf{x} - \mathbf{n}_1 \| \geq \tau(\mathbf{n}_1) \quad (2)$$

$$\| \mathbf{x} - \mathbf{n}_2 \| \geq \tau(\mathbf{n}_2) \quad (3)$$

For example, in Fig. 2a, the first winner for $\mathbf{x}$ is $\mathbf{n}_1 = \mathbf{b}$. It is linked to two nodes, $\mathbf{a}$ and $\mathbf{c}$, with distances corresponding to the lengths of the edges $e_{\mathbf{ab}}$ and $e_{\mathbf{bc}}$. Here, $\tau(\mathbf{n}_1) = e_{\mathbf{ab}}$. For the second winner, $\mathbf{n}_2$, we obtain $\tau(\mathbf{n}_2) = e_{\mathbf{bc}}$. Since $\| \mathbf{x} - \mathbf{n}_1 \| \geq \| \mathbf{a} - \mathbf{b} \|$ and $\| \mathbf{x} - \mathbf{n}_2 \| \geq \| \mathbf{b} - \mathbf{c} \|$, $\mathbf{x}$ is added as a new node (Fig. 2b). The ratio-

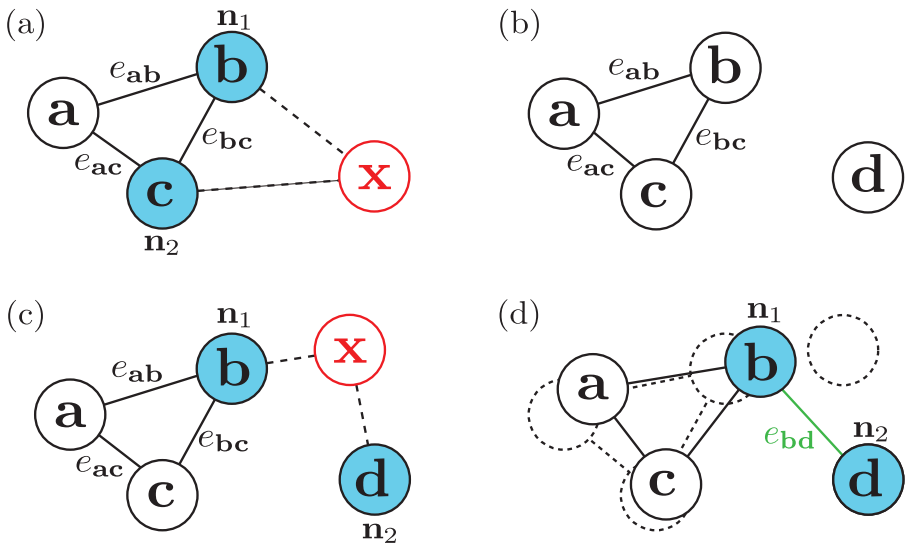

Fig. 2. (a) Simplified network of three nodes, **a, b**, and **c**, and a new case, **x**. The first winner node is **b** and the second winner node is **c**. (b) **x** is added as a new node, **d**, because it is far away from the winner nodes (Eq. (1)). (c) Another new case **x** is presented to the network. The first winner node is **b** and the second winner node is **d**. As **x** is close to its first winner node, merging occurs in (d). Also, an edge, $e_{\mathbf{bd}}$, is created between the first and second winner node.

nale is as follows. If **x** is quite different from the other nodes, then it might represent new knowledge that should be retained.[2]

If the new case **x** is *not* added as a new node, then it is merged with its first winner as follows. At iteration t, the coordinates of the first winner are $\mathbf{n}_{1,t} = (n_{11,t}, n_{12,t}, \ldots, n_{1p,t})$. The new coordinates of this node are calculated as follows.

$$\mathbf{n}_{1,t+1} = \mathbf{n}_{1,t} + \frac{\mathbf{x} - \mathbf{n}_{1,t}}{WT(\mathbf{n}_{1,t+1})} \tag{4}$$

where $WT(\mathbf{n}_{1,t})$ denotes the *winning time* of node $\mathbf{n}_1$ at iteration t. When a node appears for the first in the network, its winning time is initialized with 1. Each time that the node is selected as the first winner, its winning time is incremented by 1. For example, consider the node **b** in Fig. 2c, and assume that it has been selected as the first winner for the second time. This means that $WT(\mathbf{n}_1) = 3$. Given the new training case **x**, the new coordinates of $\mathbf{n}_1$ become $\mathbf{n}_{1,t+1} = \mathbf{n}_{1,t} + \frac{1}{3}(\mathbf{x} - \mathbf{n}_{1,t})$. Loosely speaking, the first nearest neighbor is pulled towards the new case, and its new coordinates are between its old coordinates and the coordinates of the new case. The more often a node was the first winner, the larger its *WT*, and the less influence has the new case **x** on the coordinates of that node. Conceptually, we can say that with each time a node merges with a new case, the larger the inertia of that node becomes.

Furthermore, the nodes that are directly linked to $\mathbf{n}_1$ are also pulled towards **x**. Let **a** be such a directly linked node. Then its new coordinates at iteration $t+1$ are calculated as follows:

$$\mathbf{a}_{t+1} = \mathbf{a}_t + \frac{\mathbf{x} - \mathbf{a}_t}{\eta \cdot WT(\mathbf{a}_{t+1})} \tag{5}$$

where η is the pull factor. The larger this factor, the higher the inertia of the linked nodes (Fig. 2d). In our experiments, we set $\eta = 100$.

3.4. Linking nodes

In SOINN+, a node can either be linked to other nodes, or it can exist as an unconnected node (singleton). SOINN+ tries to link those nodes that are likely to represent signal and not noise. An edge can only be created between the first and second winner node after a new case has been merged with its first winner. The linking depends on the *trustworthiness* of a node. The trustworthiness of a node **a** is denoted as $T(\mathbf{a})$ and defined as follows.

$$T(\mathbf{a}) = \frac{WT(\mathbf{a}) - 1}{\max(WT) - 1} \tag{6}$$

where $\max(WT)$ is the maximum winning time of all nodes. For example, consider the node that has maximum winning time (i.e., it has been selected most often as the first winner). Consequently, our trust in this node is maximal, with $T = 1$. By contrast, a node that has never been selected as the first winner node has winning time of 1, and its trustworthiness is therefore 0.

Consider now only those first and second winners that were linked. Denote the similarity thresholds of these nodes as $\tau_{1,i}$ and $\tau_{2,i}$, respectively, where i is a counter for the number of links created so far. The variances of these thresholds are $\sigma_1^2 = Var(\tau_{1,\cdot})$ and $\sigma_2^2 = Var(\tau_{2,\cdot})$. The network creates an edge between the first and second winnner, $\mathbf{n}_1$ and $\mathbf{n}_2$, if at least one of the following conditions is met:

$$\tau(\mathbf{n}_1) \cdot (1 - T(\mathbf{n}_1)) < \bar{\tau}_1 + 2 \cdot \sigma_1 \tag{7}$$

$$\tau(\mathbf{n}_2) \cdot (1 - T(\mathbf{n}_2)) < \bar{\tau}_2 + 2 \cdot \sigma_2 \tag{8}$$

where $\bar{\tau}_1$ and $\bar{\tau}_2$ are the arithmetic means of the similarity thresholds of the first and second winners, respectively.

After a new case has been merged with the first winner, there are two possible scenarios. First, there already exists an edge between the first and second winner. In that case, the lifetime of the edge is reset to 1. Second, no edge exists between the first and second winner. In that case, an edge is created if one of the conditions in Eq. (7) or Eq. (8) is met; otherwise, the first and second winner remain unconnected. The only exception is at the initialization of the network: the first and second edges are created unconditionally. Fig. 3 illustrates how nodes are created and linked.

Fig. 3 a illustrates the initialization of the network with three nodes, **a, b**, and **c**, and their respective winning times. In Fig. 3b, a new training case, **x**, is presented to the network, and its first and second nearest neighbors (here, **a** and **c**) are retrieved. The winning time of node **a** is updated, $WT(\mathbf{a}) = 2$. As **x** is relatively close to its nearest neighbors, **a** and **c**, it is merged with its first nearest

[2] Clearly, it is also possible that **x** represents noise. The node deletion process will take care of this later.

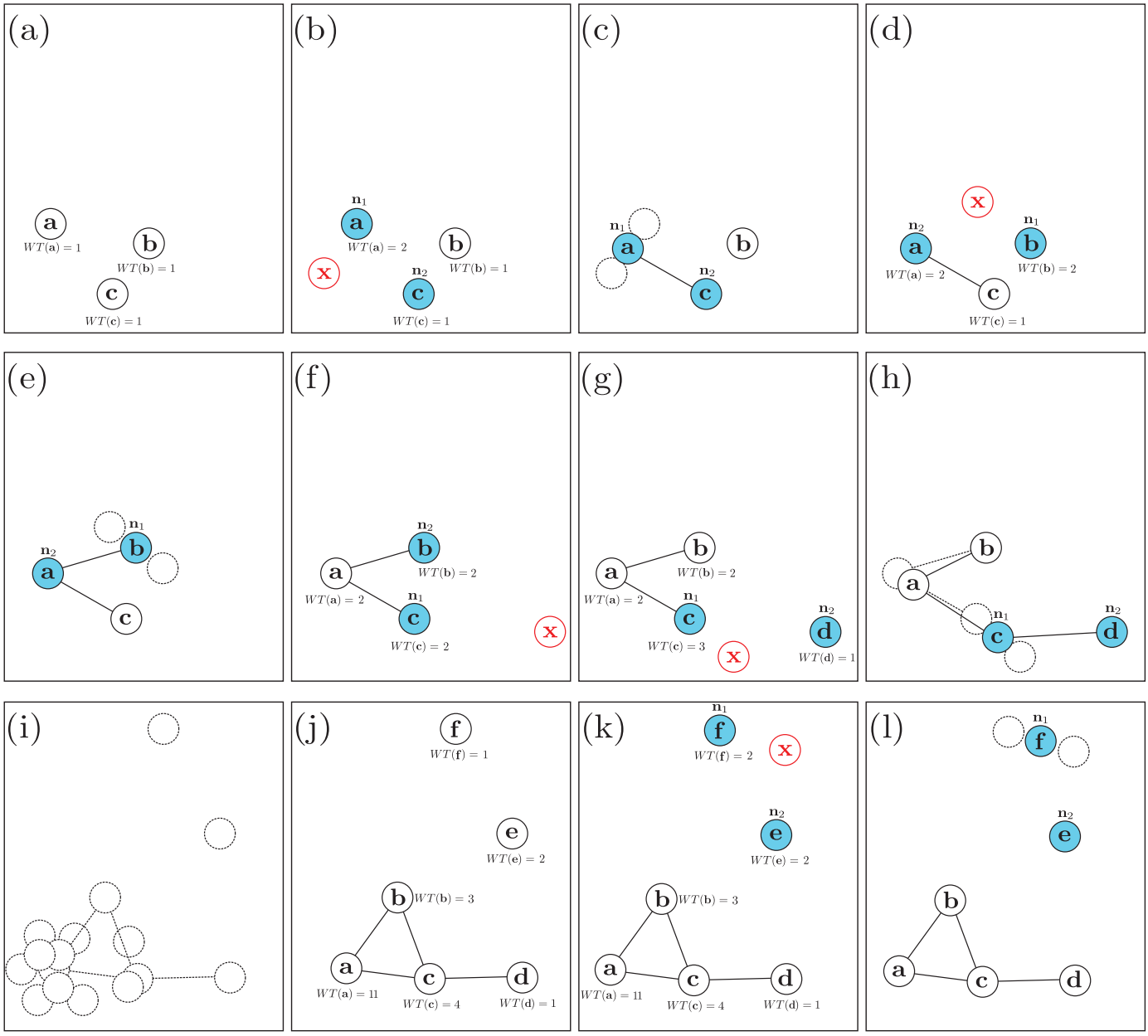

Fig. 3. Illustration of node adding, node merging, and node linking. (a) The network is initialized with three randomly selected training cases. (b) A new training case, **x**, arrives, and its first and second nearest neighbor (**a** and **c**, respectively) are retrieved. (c) As **x** is close to **a**, both nodes are merged, and an edge between **a** and **c** is created because the network is still in the initialization phase. (d) Another new training case, **x**, is presented to the network; the first nearest neighbor is now **b**, and the second nearest neighbor is **a**. (e) As **x** is close to **b**, both nodes are merged, and an edge between **a** and **b** is created. (f) Another new training case, **x**, is presented to the network. As it is far away from the other nodes, it is added as a new node, **d**. (g) Another new training case, **x**, arrives, and the nearest neighbors are retrieved (here, **c** and **d**) (h) As **x** is close to **c**, both nodes are merged, and the node **a**, being connected to **c**, is pulled towards the newly merged node. (i) This panel shows a fast-forward of the network evolution; here, many training cases (represented by dashed circles) arrived in the immediate vicinity of node **a**, causing the relatively large winning time of 11 of node **a** in panel (j). (k) Another new training case **x** arrives, and its two nearest neighbors are retrieved (here, **f** and **e**, respectively). (l) As **x** is close to **f**, both nodes are merged. As the condition for creating an edge between **f** and **e** is not met (see the main text for details), these two nodes remain unconnected.

neighbor (cf. Eqs. (2) and(3)) (Fig. 3c). As the network is still being initialized, an edge is created between **a** and **c**, with $\tau_{1,1} = \tau(\mathbf{n}_1)$ and $\tau_{2,1} = \tau(\mathbf{n}_2)$.

In Fig. 3d, another training case **x** is presented to the network. The nearest neighbors are now **b** (first winner) and **a** (second winner). As **b** is the first winner, its winning time is updated, $WT(\mathbf{b}) = 2$. Since **x** is relatively close to its first and second nearest neighbor, it is merged with **b** (Fig. 3e). Furthermore, as the network is still being initialized, an edge is created unconditionally between the first ($\mathbf{n}_1 = \mathbf{b}$) and second ($\mathbf{n}_2 = \mathbf{a}$) winner, with $\tau_{1,2} = \tau(\mathbf{n}_1)$ and $\tau_{2,2} = \tau(\mathbf{n}_2)$.

In Fig. 3f, the new case is relatively far away from its nearest neighbors (i.e., Eqs. (2) and(3) hold), so it is added as a new node and labeled as **d** in Fig. 3g. The new case **x** in Fig. 3g is close to its nearest neighbors and therefore merged with **c** in Fig. 3h. Two edges have already been created, so the initialization is finished. To check whether the two winner nodes, **c** and **d**, should be linked, we calculate $\bar{\tau}_1 = \frac{1}{2}(\tau_{1,1} + \tau_{1,2})$, $\bar{\tau}_2 = \frac{1}{2}(\tau_{2,1} + \tau_{2,2})$, $\sigma_1^2 = Var(\tau_{1,1}, \tau_{1,2})$, and $\sigma_2^2 = Var(\tau_{2,1}, \tau_{2,2})$. The trustworthiness of node **c** is calculated as $T(\mathbf{c}) = \frac{WT(\mathbf{c})-1}{\max(WT)-1} = \frac{3-1}{3-1} = 1$. Now, $\tau(\mathbf{n}_1) = \tau(\mathbf{c})$. Since $\tau(\mathbf{c}) \cdot (1 - T(\mathbf{c})) = \tau(\mathbf{c}) \cdot (1 - 1) = 0 <$

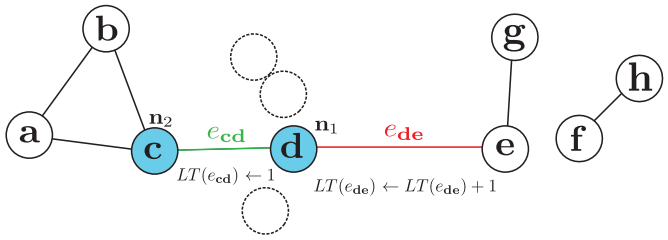

Fig. 4. Illustration of the separation of node clusters. The network consists of eight nodes belonging to two clusters, which are connected through the edge $e_{\mathbf{de}}$. The dotted circles represent new training cases, which are being merged with node **d**. Each merging results in an increment of the lifetime of the edge $e_{\mathbf{de}}$. If the lifetime exceeds the threshold λ_{edge}, it is deleted and the clusters are separated. By contrast, the lifetime of the edge between the first and second winner, $e_{\mathbf{cd}}$, is reset to 1 after each merging and therefore not deleted.

$\bar{\tau}_1 + c \cdot \sigma_1$, the edge between **c** and **d** is created, with $\tau_{1,3} = \tau(\mathbf{c})$ and $\tau_{2,3} = \tau(\mathbf{d})$.

Fig. 3 i shows a "fast-forward" in which many nodes were added and merged and edges were created. As most new cases were located around the vicinity of node **a**, the winning time of this node is now maximal; here, we assume that $WT(\mathbf{a}) = 11$ (Fig. 3j).

In Fig. 3k, the first and second nearest neighbors of the new case **x** are retrieved. The new case is merged with its first nearest neighbor in Fig. 3l. To determine whether an edge should be created between **f** and **e**, we calculate $\bar{\tau}_1$, $\bar{\tau}_2$, σ_1^2, and σ_2^2 based on the current $\tau_{1,\cdot}$ and $\tau_{2,\cdot}$. The trustworthiness of the first winner node is calculated as $T(\mathbf{f}) = \frac{WT(\mathbf{f})-1}{\max(WT)-1} = \frac{2-1}{11-1} = 0.1$. Here, $\tau(\mathbf{f}) \cdot (1 - 0.1) = 0.9\tau(\mathbf{f}) > \bar{\tau}_1 + c \cdot \sigma_1$, and therefore no edge between **f** and **e** is created.

3.5. Deleting edges

We propose a mechanism for edge deletion for the following two reasons. First, it is possible that the network creates edges between nodes of different clusters. However, edges should exist only between those nodes that are conceptually related to each other, such as members of the same class. If an edge was created between nodes of different clusters (for example, the edge $e_{\mathbf{de}}$ in Fig. 4), then this edge should be deleted. Second, by deleting edges, some nodes can become unconnected and therefore candidates for deletion, which mimics "forgetting" in natural connectionist systems.

The *lifetime* (*LT*) of an edge plays a key role in the deletion process. When an edge is created, its lifetime is set to 1. When a node becomes the first winner, the lifetime of each edge that connects to it is incremented by 1. If an edge between the first and second winner already exists, then the lifetime of this edge will be reset to 1. This process is illustrated in Fig. 4.

In Fig. 4, the three dotted circles represent new cases that are merged with their first winner node, **d**. As the edge $e_{\mathbf{cd}}$ exists between the first and second winner node, the lifetime of this edge is reset to 1 after each merging. The lifetime of all other edges between the first winner and other nodes are incremented by 1 after each merging. In the example, there exists only one such edge, $e_{\mathbf{de}}$. This edge connects nodes of different clusters: the nodes $\{\mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{d}\}$ and $\{\mathbf{e}, \mathbf{f}, \mathbf{g}, \mathbf{h}\}$. Therefore, this edge is a candidate for deletion, which depends on the lifetime of this edge.

In ASOINN, an edge is deleted if its lifetime exceeds a fixed, user-defined threshold a_d (Shen & Hasegawa, 2008). We propose a different approach as follows. First, we retrieve all edges through which the first winner node can be reached. In Fig. 4, these are all edges except the edge between **f** and **h**. Let the set $\mathcal{A}$ contain the lifetimes of these edges. We wish to identify those lifetimes that are exceptionally high (i.e., outliers). We use a standard approach for identifying an outlier based on the interquartile range,

$$\omega_{\text{edge}} = \mathcal{A}_{0.75} + 2 \cdot \text{IQR}(\mathcal{A}) \tag{9}$$

where $\mathcal{A}_{0.75}$ is the 75th percentile of the lifetimes in $\mathcal{A}$ and $\text{IQR}(\mathcal{A})$ is the interquartile range.

Let $\mathcal{A}_{\text{del}}$ denote the set of lifetimes of only those edges that were deleted so far. We define the threshold λ_{edge} as follows:

$$\lambda_{\text{edge}} = \bar{\mathcal{A}}_{\text{del}} \cdot \frac{|\mathcal{A}_{\text{del}}|}{|\mathcal{A}_{\text{del}}| + |\mathcal{A}|} + \omega_{\text{edge}} \cdot \left(1 - \frac{|\mathcal{A}_{\text{del}}|}{|\mathcal{A}_{\text{del}}| + |\mathcal{A}|}\right) \tag{10}$$

where $\bar{\mathcal{A}}_{\text{del}}$ is the arithmetic mean of lifetimes in $\mathcal{A}_{\text{del}}$.

Every time that the lifetime of an edge is incremented, the network checks whether the new lifetime exceeds λ_{edge} or not. If it does, then the corresponding edge is deleted.

The threshold in Eq. (10) is a function of the lifetimes of the previously deleted and undeleted edges. The rationale is as follows. At the initialization stage of the network, there are no deleted edges, and $|\mathcal{A}_{\text{del}}| = 0$, so that $\lambda_{\text{edge}} = \omega_{\text{edge}}$. This means that those edges with an unusually high lifetime are candidates for deletion. As the network evolves, nodes are merged and edges are created. As more and more edges are being deleted, $|\mathcal{A}_{\text{del}}|$ becomes larger, which gives more weight to the average lifetime of already deleted edges. For example, consider the network at an early stage, with $|\mathcal{A}_{\text{del}}| = 0$ and $|\mathcal{A}| = 3$, which means that $\lambda_{\text{edge}} = \omega_{\text{edge}}$. Then, as more and more edges are being deleted, suppose that $|\mathcal{A}_{\text{del}}| = 10$ and $|\mathcal{A}| = 10$. This means that $\lambda_{\text{edge}} = \frac{1}{2}\bar{\mathcal{A}}_{\text{del}} + \frac{1}{2}\omega_{\text{edge}}$. So the threshold results from an equal weighting of the average lifetime of deleted nodes and the criterion for measuring an outlier lifetime. Eventually, as the network further evolves, consider $|\mathcal{A}_{\text{del}}| = 900$ and $|\mathcal{A}| = 100$. Now, $\lambda_{\text{edge}} = \frac{900}{1000}\bar{\mathcal{A}}_{\text{del}} + \frac{100}{1000}\omega_{\text{edge}}$. To summarize, when a lot of edges have been deleted, the threshold is mainly based on the average lifetime of deleted edges. On the other hand, if only a few edges have been deleted so far, then the threshold is mainly based on the outlierness of a lifetime. In Eq. (10), $\frac{|\mathcal{A}_{\text{del}}|}{|\mathcal{A}_{\text{del}}|+|\mathcal{A}|}$ acts as the corresponding weight factor. The effect of λ_{edge} in Eq. (10) is that edges between different, non-overlapping clusters of nodes are deleted, hence, clusters are separated. Thereby, the network can accommodate cases of different classes without conflating them.

3.6. Deleting nodes

When the network is trained on noisy data, it is possible that nodes are created that represent noise rather than signal, specifically at the initialization of the network. We therefore propose a node pruning strategy. We posit that any node that has not been connected to any other nodes for a reasonably long time is a candidate for deletion. We define the *idle time* of a node **a**, denoted by $IT(\mathbf{a})$, as the number of iterations that **a** was *not* selected as the first winner. Each time that **a** is selected as the first winner, its idle time is reset to 0.

Let the idle time of all connected nodes be elements of the set $\mathcal{I}$. We define the concept of *unutility* of a node **a** as the ratio of its idle time and winning time,

$$U(\mathbf{a}) = \frac{IT(\mathbf{a})}{WT(\mathbf{a})} \tag{11}$$

A high value of unutility means that the corresponding node was rarely selected as the first winner, whereas a low value means that the node was relatively often selected as the first winner.

We now propose a threshold for the deletion of nodes. Let the set $\mathcal{U}$ contain the unutilities of all connected nodes of the network. The median absolute deviation (sMAD) is a measure of the dispersion of data, which is more robust to outliers than the standard

deviation. The *scaled median absolute deviation* (sMAD) of the elements in $\mathcal{U}$ is defined as

$$\text{sMAD}(\mathcal{U}) = b \cdot \text{median}(|\forall u_i - \text{median}(\mathcal{U})|) \tag{12}$$

where the scaling factor b is needed to make the estimator consistent for the parameter of interest (Rousseeuw & Croux, 1993). For a consistent estimator of the standard deviation under an assumed Gaussian distribution, b has to be set to 1.4826 (Rousseeuw & Croux, 1993). Based on sMAD($\mathcal{U}$), we define the parameter ω_{node} as follows:

$$\omega_{\text{node}} = \text{median}(\mathcal{U}) + 2 \cdot \text{sMAD}(\mathcal{U}) \tag{13}$$

Let $\mathcal{I}$ be the set of *unconnected* nodes and let $\mathcal{S}$ be the set of *all* nodes in the network. Consider now the ratio between unconnected nodes and all nodes,

$$R_{\text{noise}} = \frac{|\mathcal{I}|}{|\mathcal{S}|} \tag{14}$$

Let $\mathcal{B}_{\text{del}}$ denote the set of deleted nodes. The threshold for deleting a node is defined as follows:

$$\lambda_{\text{node}} = \bar{\mathcal{U}}_{\text{del}} \cdot \frac{|\mathcal{B}_{\text{del}}|}{|\mathcal{B}_{\text{del}}| + |\mathcal{S} \setminus \mathcal{I}|} + \omega_{\text{node}} \cdot \left(1 - \frac{|\mathcal{B}_{\text{del}}|}{|\mathcal{B}_{\text{del}}| + |\mathcal{S} \setminus \mathcal{I}|}\right) \cdot (1 - R_{\text{noise}}) \tag{15}$$

where $\bar{\mathcal{U}}_{\text{del}}$ denotes the average unutility of the deleted nodes. To delete a node **a**, the following three conditions must be met:

1. at least one edge must exist in the network
2. $U(\mathbf{a}) > \lambda_{\text{node}}$
3. **a** must be an unconnected node

We now explain the rationale for this pruning strategy and for the threshold in Eq. (15). When the network is initialized, there are no deleted nodes, so $|\mathcal{B}_{\text{del}}| = 0$. This means that $\lambda_{\text{node}} = \omega_{\text{node}} \cdot (1 - R_{\text{noise}})$. Consider a network that is being initialized, and assume that there are only unconnected nodes. This means that $R_{\text{noise}} = 1$. Consequently, $\lambda_{\text{node}} = 0$. This would imply that all nodes have to be deleted, since their unutility is higher than 0. To prevent this from happening, we start the node deletion only after the first edge has been created (cf. condition 1 above).

The rationale for λ_{node} is similar to that of λ_{edge}. The factor $\frac{|\mathcal{B}_{\text{del}}|}{|\mathcal{B}_{\text{del}}|+|\mathcal{S}\setminus\mathcal{I}|}$ acts as a weight: the more nodes we delete, the larger this factor, and the stronger the influence of $\bar{\mathcal{U}}_{\text{del}}$ on λ_{node}, i.e., the stronger the influence of the average unutility of nodes. On the other hand, if only a few nodes have been deleted so far, then we want to downweight the influence of $\bar{\mathcal{U}}_{\text{del}}$ and instead give more weight to ω_{node}.

The effect of the factor $(1 - R_{\text{noise}})$ is the following. If an evolved (i.e., not just initialized) network contains a lot of unconnected nodes, then it is likely that many of them represent noise. Therefore, we want to facilitate the deletion of nodes. This means that λ_{node} should be relatively low. On the other hand, if the network contains only a few unconnected nodes, there is comparatively less noise. Now λ_{node} should be relatively large because we want to delay the deletion of nodes and thereby give the unconnected nodes the chance to become connected. In other words, when there are only a few unconnected nodes, the network can afford to take a more tolerant stance towards nodes that might be noise and give them a chance to prove that they are signal by becoming connected to other nodes as training continues. This idea is illustrated in Fig. 5. Fig. 5a contains $|\mathcal{S}| = 3$ connected and $|\mathcal{I}| = 1$ unconnected nodes, so $(1 - R) = (1 - \frac{1}{4}) = 0.75$. Fig. 5b also contains three connected and but nine unconnected node, so $(1 - \frac{9}{12}) = 0.25$. Therefore, λ_{node} is three times smaller in Fig. 5b. In Fig. 5a, the network can afford a more lenient pruning strategy than in Fig. 5b.

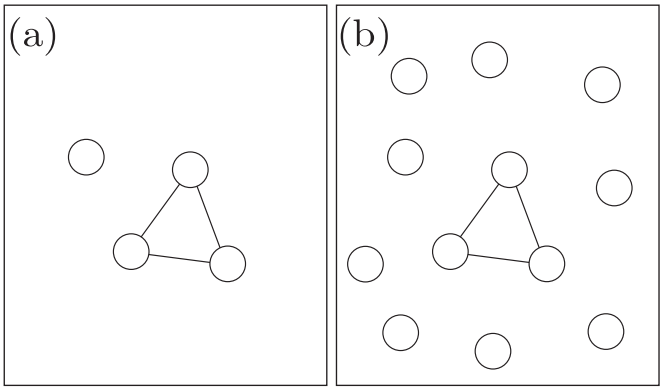

Fig. 5. Illustration of the pruning rationale. (a) A network with 1 unconnected and 3 connected nodes; (b) a network with 9 unconnected and 3 connected nodes. In (a), the network can afford to adopt a relatively more lenient pruning strategy than in (b). In (a), the same unconnected node might still be noise, but it is given the chance to connect to other nodes and thereby "prove" that it is in fact signal.

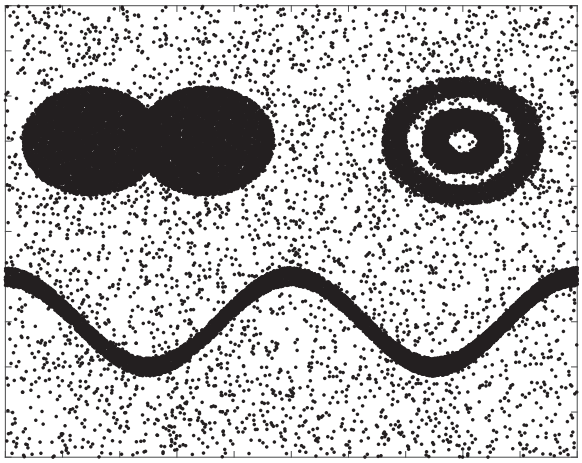

Fig. 6. Synthetic data set Synth5C: five classes with 35 000 real bivariate cases and 10% noise (3885 of 38 885 cases).

4. Experiments and results

We were interested in the following questions:

1. How does SOINN+ compare to its closest relatives, ASOINN (Shen & Hasegawa, 2008) GWR (Marsland et al., 2002), and Gamma-GWR (Parisi et al., 2017), in unsupervised learning with respect to (*i*) robustness to noise, (*ii*) catastrophic forgetting, and (*iii*) network structure, specifically the number of nodes required to represent the training data, under stationary data distributions? How do they compare when the data arrive in streams with sudden and recurring concept drifts?
2. How does SOINN+ perform against stream clustering algorithms with respect to (*i*) robustness to noise, (*ii*) maintenance of clusters under concept drift, and (*iii*) cluster purity?

To address the first question, we generated a synthetic data of 2-dimensional data points, so that the resulting networks can be easily visualized. The synthetic data set, called Synth5C, comprises five classes of uniformly distributed cases, as shown in Fig. 7a. The first class is a circle of radius $r_1 = 12$, with center $c_1 = (15, 70)$. The first class contains a total of 7500 cases. The second, overlapping class is a circle of radius $r_2 = 12$, with center $c_2 = (35, 70)$. The second class also has 7500 cases. The third class is the outer ring with an inner radius of $R_i = 11$ and an outer radius of $R_o = 13$. The center of the ring is $c_3 = (80, 70)$. The third class has a to-

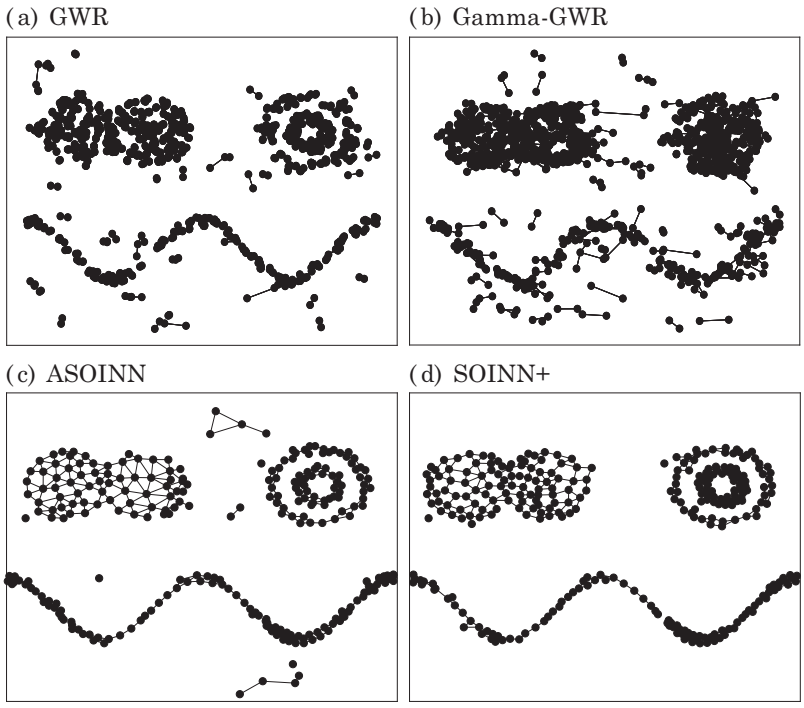

Fig. 7. Training with stationary Synth5C data set. (a) GWR network with 7501 nodes; (b) Gamma-GWR network with 7950 nodes; (c) ASOINN network with 268 nodes; (d) SOINN+ network with 303 nodes.

tal of 5000 cases. The fourth class is the inner ring, with an inner radius $r_i = 4$ and an outer radius of $r_o = 6$. This class has a total of 5000 cases. The fifth class is a band defined by two cosine curves, where the upper curve is $f_1(x) = 10\cos(0.04\pi x) + 31$ and the lower curve is $f_2(x) = 10\cos(0.04\pi x) + 29$. This class has a total of 10 000 cases, with 2500 cases $x_i \in [0, 33]$, 2500 cases $x_i \in [33, 66]$, and 5000 cases $x_i \in [66, 100]$. Furthermore, 10% of the data are noise (i.e., 3885 cases). The entire data set has 38 850 cases.

To address the second question, we compared SOINN+ with five stream clustering methods based on five synthetic data sets, including Synth5C. There exist a plethora of stream clustering methods, which can be categorized into partitioning, hierarchical, density-based, grid-based, and model-based methods (Amini, Wah, & Saboohi, 2014). Here, we used ClusterGenerator (Bifet, Holmes, Kirkby, & Pfahringer, 2010), CluStream (Aggarwal, Yu, Han, & Wang, 2003), StreamKM++ (Ackermann et al., 2012), and CluTree (Kranen, Assent, Baldauf, & Seidl, 2009), which find (hyper-)spherical clusters. DenStream is a density-based algorithm that can find clusters of arbitrary shapes. In our experiments, we used the algorithms implemented in the Massive Online Analysis (MOA) suite (Bifet et al., 2010). Supplementary materials (including Matlab code) to reproduce our experiments are available at https://osf.io/6dqu9/.

4.1. Comparison of SOINN+, ASOINN, GWR, and Gamma-GWR using stationary data

We used the entire Synth5C data set to train the models. The resulting networks are shown in Fig. 7a-c. To build the GWR and Gamma-GWR networks, we used the code provided by Parisi et al. (2017); Parisi, Weber, and Wermter (2015). At each iteration, the age of an edge was incremented by 1. Edges whose age exceeded the default value of $\mu_{\max} = 600$ were deleted, and unconnected nodes were removed. The maximum number of nodes was set to $+\infty$ to give the network the chance to remember all inputs. We used the default values for all other network parameters and trained each network for one epoch.

In this stationary analysis, both GWR and Gamma-GWR could learn the underlying data distributions to some degree, but included many noise cases as nodes (Fig. 7a,b). By contrast, both ASOINN and SOINN+ found topological mappings of the input data that correspond well to the underlying distributions. SOINN+ is slightly more robust to noise than ASOINN, which created edges between noise cases. The SOINN+ network consists of only 303 nodes (less than 1% of the training cases), which are sufficient for a compact, practically noise-free representation of the training cases.

4.2. Comparison of SOINN+, ASOINN, GWR, and Gamma-GWR using data streams with concept drift

Next, we partitioned Synth5C into seven streams, where each stream contains a number of cases from different classes (Table 1). Hence, the streams are characterized by sudden and recurring concept drifts.

The first class contains a total of 7500 cases, with 5000 cases arriving in stream #1 and 2500 cases arriving in stream #4. The second, overlapping class has also 7500 cases, with 5000 cases arriving in stream #2 and 2500 cases arriving in stream #3. The third class has a total of 5000 cases, with 2500 cases arriving in stream #3 and 2500 cases coming in stream #6. The fourth class has a total of 5000 cases, with 2500 arriving in stream #4 and 2500 arriving in stream #5. The fifth class has a total of 10 000 cases, with 2500 cases $x_i \in [0, 33]$ (class 5a) arriving in stream #5, 2500 cases $x_i \in [33, 66]$ (class 5b) arriving in stream #6, and 5000 cases $x_i \in [66, 100]$ (class 5c) arriving in stream #7. Furthermore, each stream also contains 10% of noise (i.e., 555 noise cases).

Fig. 8 shows the seven data streams (panels on the left). Both ASOINN and SOINN+ were able to learn a meaningful representation of the input data. Note that the number of classes was not revealed to the models *a priori*; nonetheless, they learned the intrinsic structure of the classes without overwriting previously learned structures. For example, after both models had processed the data from stream #1, the resulting networks were not corrupted when the models were presented with data from stream #2. Cases from

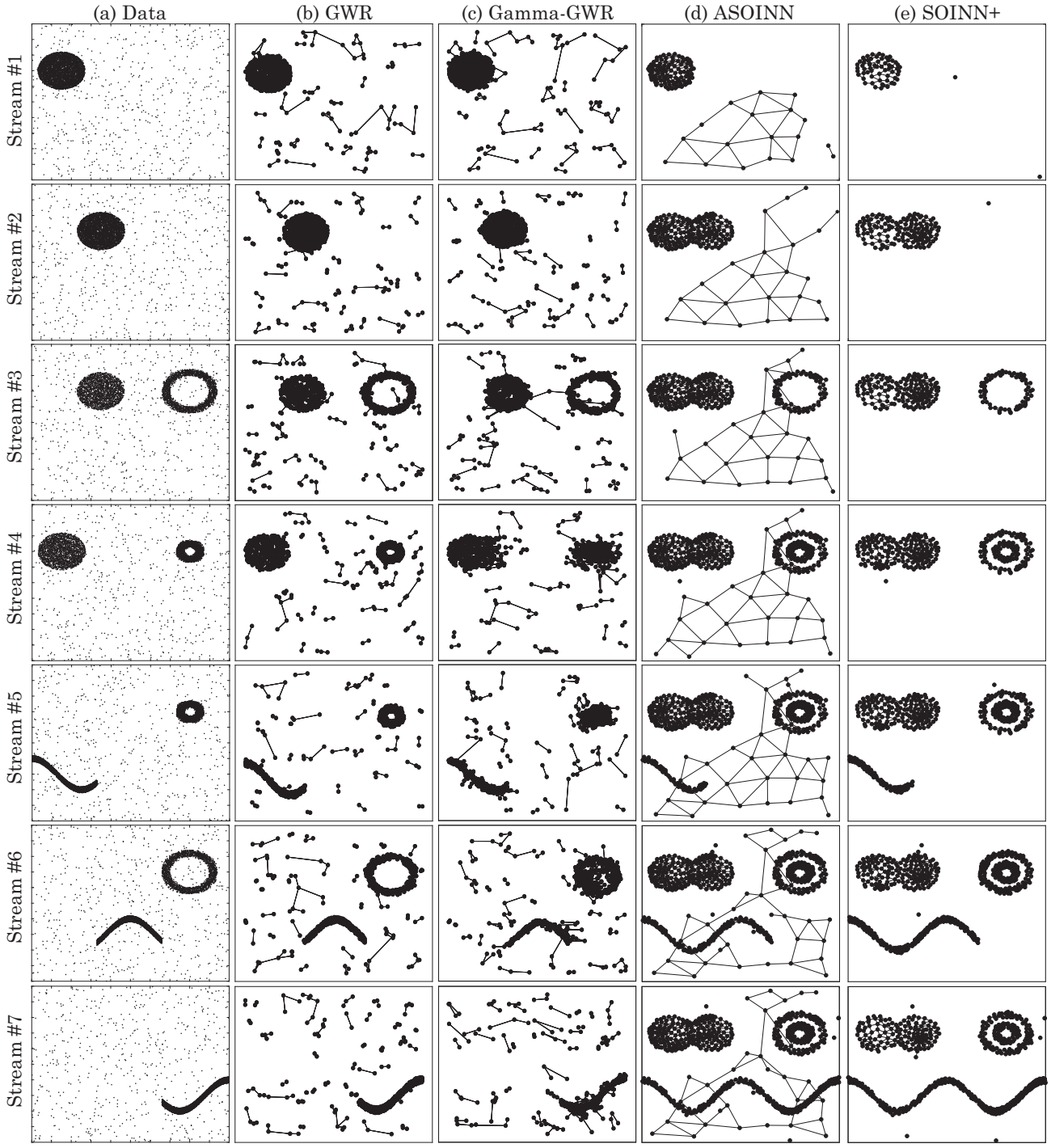

Fig. 8. (a) The panels on the left show the data from the seven data streams, and the remaining panels show the networks after processing each stream. (a) The final (i.e., after processing all seven streams) GWR network consists of 1950 nodes. (b) The final Gamma-GWR network consists of 2074 nodes. (d) The final ASOINN network consists of 611 nodes. (e) The final SOINN+ network consists of 739 nodes. The evolution of the ASOINN and SOINN+ networks can be seen in two videos, which are available at https://osf.io/5rezh/.

Table 1
Seven sequential data streams and the number of cases from the five different classes of Synth5C.

	Class 1	Class 2	Class 3	Class 4	Class 5a	Class 5b	Class 5c	Noise
Stream #1	5000	0	0	0	0	0	0	555
Stream #2	0	5000	0	0	0	0	0	555
Stream #3	0	2500	2500	0	0	0	0	555
Stream #4	2500	0	0	2500	0	0	0	555
Stream #5	0	0	0	2500	2500	0	0	555
Stream #6	0	0	2500	0	0	2500	0	555
Stream #7	0	0	0	0	0	0	5000	555

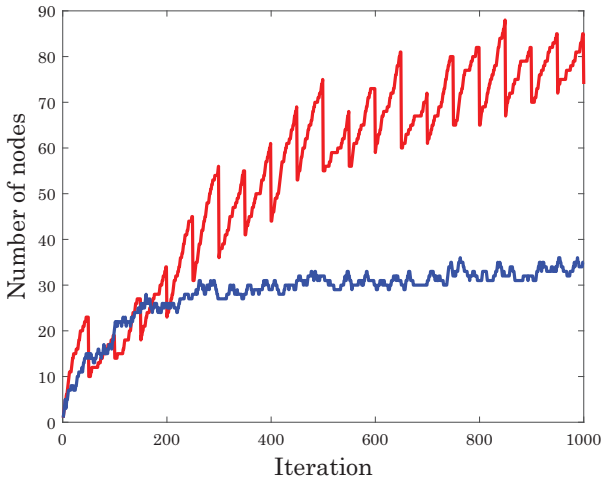

Fig. 9. Number of nodes per iteration in ASOINN (red curve) and SOINN+ (blue curve) for the first 1000 cases from Fig. 8. In ASOINN, nodes are deleted periodically (here, every 50 iterations), causing the zig-zag curve. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

new streams are accommodated into the already existing networks. Whereas ASOINN includes noise as unconnnected nodes starting from stream #1, SOINN+ is more resilient to noise. After processing all streams, the final ASOINN network consists of 611 nodes, while the final SOINN+ network consists of 739 nodes. The evolution of the networks can be seen in two videos, which are available at https://osf.io/5rezh/.

GWR produced networks that reflect the underlying data distributions; however, new streams eradicated previously learned structures. For example, after processing the 5555 cases of stream #2, the network based on stream #1 has become erased. We tried to prevent this catastrophic forgetting by setting the maximum age of an edge, $\mu_{\max}$, to $+\infty$, but then too many noise cases were added to the network as connected nodes. In this experiment, GWR was not resilient to noise, as it created edges for noise cases in all data streams. Gamma-GWR performed similarly: learned structures were eradicated by new data streams, and edges between noise cases were created.

SOINN+ "forgets" more gracefully than ASOINN, as shown in Fig. 9. Here, the red curve shows the number of nodes in ASOINN, and the blue curve shows the number of nodes in SOINN+ for the first 1000 cases from Fig. 8. The periodic sharp decreases in the number of nodes in ASOINN are due to the user-defined parameters for node deletion (which were set to occur after every 50 iterations). Conceptually, this means that the network "forgets" at fixed intervals. This leads to sudden "jumps" in the network structure (cf. Fig. 9). By contrast, the curve for SOINN+ is smoother, which corresponds more to how natural cognitive systems tend to forget (French, 1999).

4.3. Comparison of SOINN+ and stream clustering methods

How does SOINN+ perform against stream clustering methods with respect to detecting real clusters? Can the algorithms maintain clusters across data streams, or do previously discovered clusters become "forgotten"? How well do the algorithms cope with noise? To address these questions, we used again the seven streams of Synth5C. We performed the experiments in MOA (Bifet et al., 2010) by presenting sequentially and cumulatively the data streams to each algorithm. Fig. 10 shows the clustering results. When the ground truth is known (as in this example), clustering algorithms are usually compared based on extrinsic performance measures. A common evaluation measure is the *purity* (Eq. 16), which is the average of the fraction of items belonging to the majority class in each cluster (Ghesmoune et al., 2016),

$$\text{purity} = \frac{1}{k}\sum_{i=1}^{k}\frac{m_i}{n_i} \tag{16}$$

where k denotes the number of clusters, m_i is the number of the majority (i.e., dominant) cases in the i^{th} cluster, and n_i is the total number of cases in the i^{th} cluster. The larger the purity, the better the clustering result. The maximum purity is 1.00.

ClusterGenerator correctly identified the first circular class (stream #1, Fig. 10) and maintained this cluster after processing stream #2. However, ClusterGenerator failed to maintain its first cluster after processing stream #3. The two concentric rings were identified and maintained until after processing stream #7. The data in the cosine band (streams #5 to #7) was discovered; however, the spherical shape of the cluster does not reflect the distribution well. Based on a visual inspection, CluStream, StreamKM++, and ClusTree perform worse than ClusterGenerator because clusters are found in noise, and real clusters are not well maintained across data streams. Among the investigated methods, DenStream found the clusters that correspond best to the ground truth. As a density-based clustering method, DenStream can detect clusters of arbitrary shapes and cope with noise, in contrast to the partitioning or hierarchical clustering methods. The final clusters by DenStream are similar to the result by SOINN+.

SOINN+ produced clusters whose purity was on par with, or better than, that of the compared methods, specifically after processing all seven data streams: the average cluster purity is 0.92 for SOINN+, compared to only 0.89 of the next best method, DenStream. However, there is an important caveat: the purity measure is influenced by noise. SOINN+ tries to discard noise cases, whereas the stream clustering methods do not. For example, consider the result by CluStream after processing the first data stream (first row, second column in Fig. 10). Two clusters are found, but only one of them corresponds to the ground truth. The cluster containing noise cases is misleading, but it has perfect purity because it contains only noise cases.

Next, we analyzed the following four benchmark data sets for streaming algorithms:

1. *SEA* consists of 60 000 cases and 3 features. Only the first two features are relevant for a separation into two classes. A sudden concept drift occurs every 15 000 cases (Street & Kim, 2001).
2. *1CDT* consists of 16 000 cases and two features. The number of classes is two (Souza, Silva, Gama, & Batista, 2015). Concept drift occurs every 400 cases.
3. *Keystroke* is a collection of password keystrokes (Souza et al., 2015). The data set consists of 1600 cases of 4 classes, and each case is described by 10 features. Concept drift occurs every 200 cases.
4. *Powersupply* contains 29 928 cases of 24 classes, and each case is described by 2 features (Dau et al., 2018; Zhu, 2010). The data set contains the three-year power supply records of an Italian electricity company. The concept drift depends on the season, weather, hours of the day, and the differences between working and non-working days; thus, concept drift occurs at irregular intervals.

We partitioned each data set into streams according to the underlying concept drifts, i.e., the SEA data set was partitioned into four streams of 15 000 cases each; 1CDT was partitioned into 40 streams of 400 cases each; and Keystroke was partitioned into 8 streams of 200 cases. As the concept drift occurs irregularly in Powersupply, we decided to partition this data set into 10 streams of 3000 cases.

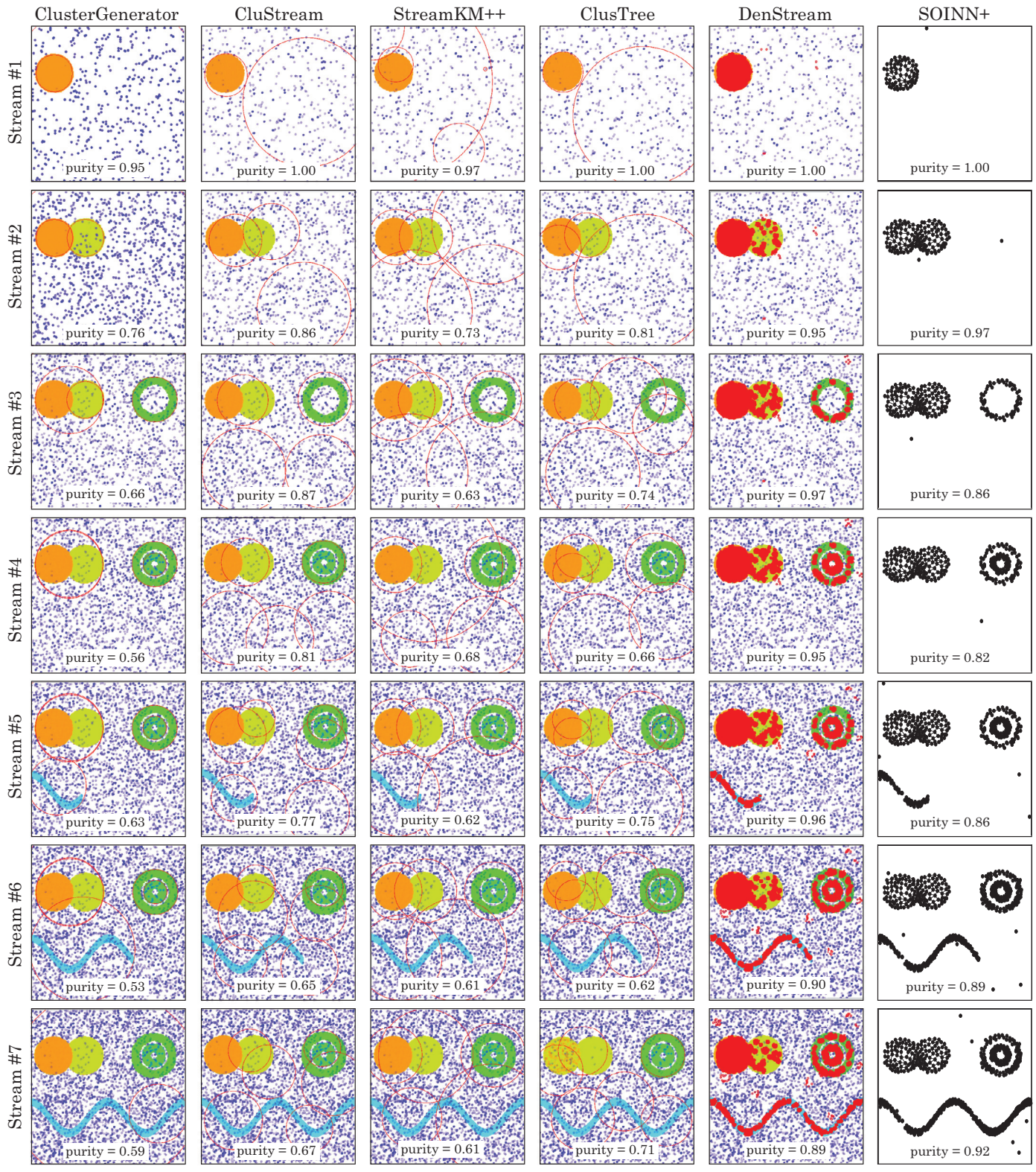

Fig. 10. Comparison of SOINN+ and five stream clustering methods: ClusterGenerator, CluStream, StreamKM++, ClusTree, and DenStream. Clusters are represented by red circles. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

Each algorithm was trained as described above for the data set Synth5C, i.e., first, we presented the data from the first stream and let the algorithm discover the clusters. We then calculated the cluster purity for each algorithm. Next, we added the data from the second stream, let the algorithms discover the clusters, and then we calculated the purity again. We proceeded analogously for all streams. For all data sets, SOINN+ achieved a cluster purity on par with, or better than, the competing algorithms (Fig. 11). Interestingly, the purity of clusters produced by DenStream and SOINN+ is comparable, except for the data set Powersupply, where SOINN+ achieved a remarkably better performance. A possible reason is that this data set contains a relatively large number ($n = 24$) of

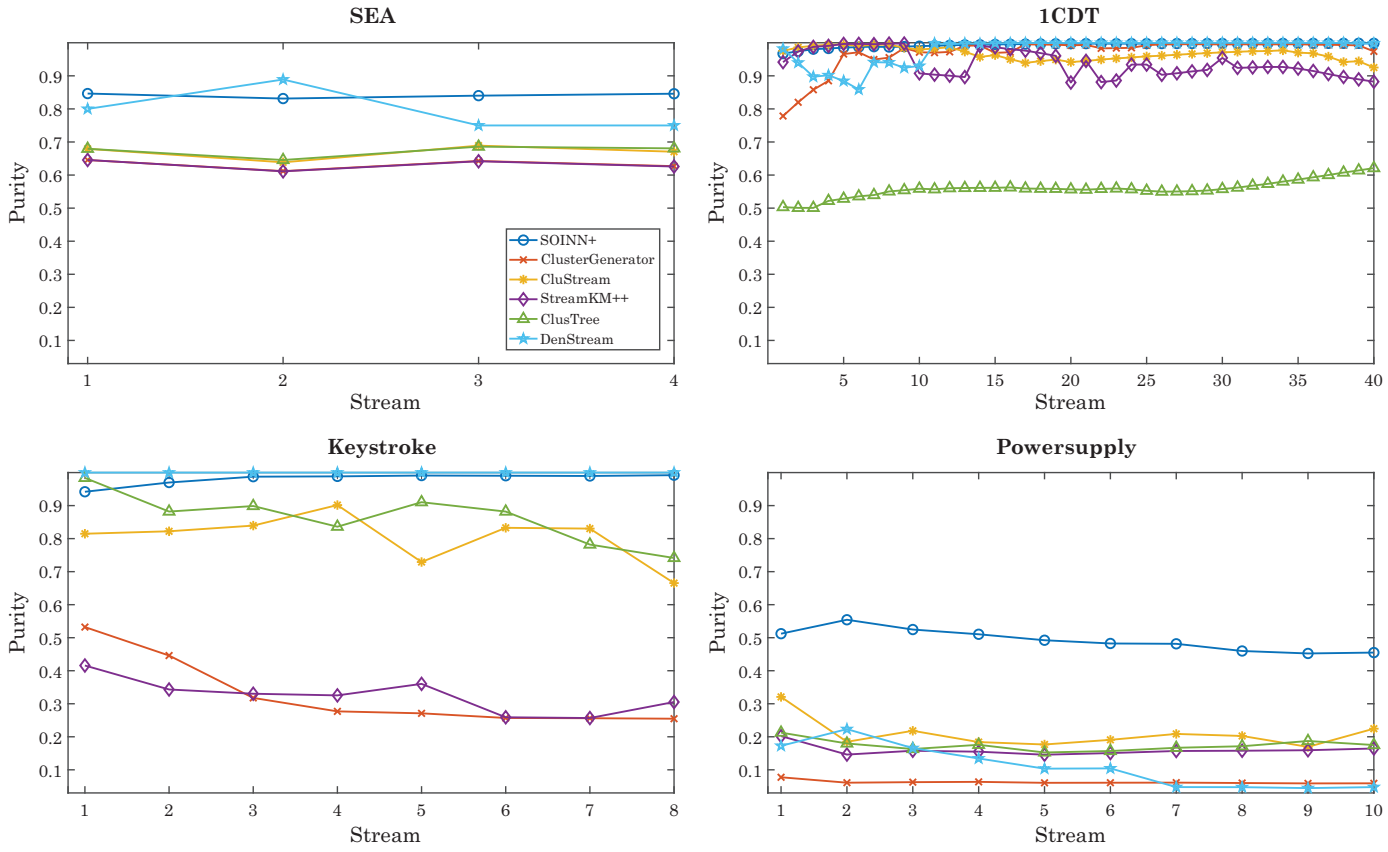

Fig. 11. Comparison of SOINN+ and five stream clustering methods (ClusterGenerator, CluStream, StreamKM++, ClusTree, and DenStream) on four data sets (SEA, 1CDT, Keystroke, and Powersupply) based on cluster purity.

partially overlapping classes, which may be merged into the same cluster by DenStream, so that the resulting purity is decreased.

5. Discussion and conclusions

The hallmark of a continuous learning system is the ability to acquire new information, to refine existing knowledge without catastrophic forgetting, and to apply knowledge across domains (Parisi et al., 2019). Here, we proposed SOINN+, a new self-organizing incremental neural network for unsupervised learning from both stationary and non-stationary data. Like previous versions of SOINN, our proposed method learns a topological mapping of the input data to a network structure. As new data become available over time, the network topology adapts to new input by adding or deleting nodes, creating or deleting edges, or a combination of both. However, SOINN+ is fundamentally different from previous generations of SOINN and related methods, such as Growing When Required (GWR) networks, with respect to how edges are created, and how nodes and edges are deleted. We formulated three new concepts that are key factors for a more graceful forgetting in SOINN+: idle time of a node, trustworthiness of a node, and (un-)utility of a node. We believe that when—and how—the network deletes its nodes and edges are crucial factors for mitigating catastrophic forgetting. In SOINN+, the deletion of network structures is considered an intrinsic part of the learning process. Our improved edge deletion also allows for a better separation of clusters of nodes, which is desirable for accommodating cases of new classes, since these should not be merged with already existing node clusters.

We compared SOINN+ with similar neural networks, GWR (Marsland et al., 2002), Gamma-GWR (Parisi et al., 2017), and ASOINN (Shen & Hasegawa, 2008), for both stationary and non-stationary unsupervised learning. SOINN+ was the most robust to noise and able to find topological representations that correspond well to the true data distributions. Although the total number of classes or clusters was not known a priori, SOINN+ could accommodate new cases without compromising previously learned network structures. The further experiments with five benchmark data sets showed that SOINN+ is not only able to reliably detect real clusters in noisy data streams, but also to remember these clusters under concept drifts. SOINN+ produced clusters with a purity that was either on par with, or better than, that of five established stream clustering methods.

However, SOINN+ has several limitations. An open problem is the measure of similarity that is used to add new nodes. The current SOINN+ algorithm uses the Euclidean distance, but this metric is not suitable for high-dimensional data or data sets consisting of both numeric and categorical attributes. In our ongoing work, we are investigating different distance metrics, notably the fractional distance, to address this issue. Another scope of our future work is supervised learning. Currently, SOINN+ is designed for unsupervised learning (topology learning, clustering), but the model could be extended to supervised learning as well. Our idea here is to use the network nodes as prototypes for classification based on a *k*-neareast-neighbor approach.

Declaration of Competing Interests

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Credit authorship contribution statement

Chayut Wiwatcharakoses: Conceptualization, Methodology, Software, Data curation, Writing - original draft. **Daniel Berrar:** Conceptualization, Methodology, Writing - original draft, Writing - review & editing, Supervision.

Acknowledgments

CW was supported by a scholarship from the Japanese Ministry of Education, Culture, Sports, Science and Technology. We thank the anonymous reviewers for their detailed and constructive comments on our manuscript.

Supplementary material

Program code to reproduce our experiments, data sets, and supplementary videos showing the network evolution are available at https://osf.io/6dqu9/.

Supplementary material associated with this article can be found, in the online version, at doi:10.1016/j.eswa.2019.113069.

References

Ackermann, M. R., Märtens, M., Raupach, C., Swierkot, K., Lammersen, C., & Sohler, C. (2012). StreamKM++: A clustering algorithm for data streams. *Journal of Experimental Algorithmics, 17*, 2.4:2.1–2.4:2.30.

Aggarwal, C. C., Yu, P. S., Han, J., & Wang, J. (2003). A framework for clustering evolving data streams. In J.-C. Freytag, P. Lockemann, S. Abiteboul, M. Carey, P. Selinger, & A. Heuer (Eds.), *Proceedings of the 29th international conference on very large databases* (pp. 81–92). San Francisco: Morgan Kaufmann.

Amini, A., Wah, T. Y., & Saboohi, H. (2014). On density-based data streams clustering algorithms: A survey. *Journal of Computer Science and Technology, 29*, 116–141.

Bashivan, P., Schrimpf, M., Ajemian, R., Rish, I., Riemer, M., & Tu, Y. (2018). Continual learning with self-organizing maps. In *Proceedings of the 2nd continual learning workshop, neural information processing systems* (pp. 1–6).

Bifet, A., Holmes, G., Kirkby, R., & Pfahringer, B. (2010). MOA: Massive online analysis. *Journal of Machine Learning Research, 11*, 1601–1604.

Bruske, J., & Sommer, G. (1995). Dynamic cell structure learns perfectly topology preserving map. *Neural Computation, 7*, 845–865.

Dau, H. A., Keogh, E., Kamgar, K., Yeh, C.-C. M., Zhu, Y., Gharghabi, S., Ratanamahatana, C. A., Yanping, Hu, B., Begum, N., Bagnall, A., Mueen, A., & Batista, G. (2018). The UCR time series classification archive. https://www.cs.ucr.edu/~eamonn/time_series_data_2018/, accessed 1 July 2019.

Ditzler, G., Roveri, M., Alippi, C., & Polikar, R. (2015). Learning in nonstationary environments: a survey. *IEEE Computational Intelligence Magazine, 10*, 12–25.

Flesch, T., Balaguer, J., Dekker, R., Nili, H., & Summerfield, C. (2018). Comparing continual task learning in minds and machines. *Proceedings of the National Academy of Sciences, 115*, E10313–E10322.

French, R. (1999). Catastrophic forgetting in connectionist networks: Causes, consequences and solutions. *Trends in Cognitive Sciences, 3*, 128–135.

Fritzke, B. (1994a). Growing cell structures–a self-organizing network for unsupervised and supervised learning. *Neural Networks, 7*, 1441–1460.

Fritzke, B. (1994b). A growing neural gas network learns topologies. In *Proceedings of the 7th international conference on neural information processing systems* (pp. 625–632). Cambridge, MA, USA: MIT Press.

Ghesmoune, M., Lebbah, M., & Azzag, H. (2016). A new growing neural gas for clustering data streams. *Neural Networks, 78*, 36–50.

Ghomeshi, H., Gaber, M. M., & Kovalchuk, Y. (2019). EACD: Evolutionary adaptation to concept drifts in data streams. *Data Mining and Knowledge Discovery, 33*, 663–694.

Grossberg, S. (1987). Competitive learning: From interactive activation to adaptive resonance. *Cognitive Science, 11*, 23–63.

Hamker, F. (2001). Life-long learning cell structures—continuously learning without catastrophic interference. *Neural Networks, 14*, 551–573.

He, X., Kojima, R., & Hasegawa, O. (2007). Developmental word acquisition and grammar learning by humanoid robots through a self-organizing incremental neural network. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics), 37*, 1357–1372.

Kamiya, Y., Ishii, T., Shen, F., & Hasegawa, O. (2017). An online semi-supervised clustering algorithm based on a self-organizing incremental neural network. In *Proceedings of the international joint conference on neural networks* (pp. 1–6).

Kawewong, A., Honda, Y., Tsuboyama, M., & Hasegawa, O. (2010). Reasoning on the self-organizing incremental associative memory for online robot path planning. *IEICE Transactions on Information and Systems, E93.D*, 569–582.

Kirkpatrick, J., Pascanu, R., Rabinowitz, N., Veness, J., Desjardins, G., Rusu, A. A., & Hadsell, R. (2017). Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences, 114*, 3521–3526.

Kranen, P., Assent, I., Baldauf, C., & Seidl, T. (2009). Self-adaptive anytime stream clustering. In *Proceedings of the 9th IEEE international conference on data mining* (pp. 249–258).

Li, Z., & Hoiem, D. (2016). Learning without forgetting. In B. Leibe, J. Matas, N. Sebe, & M. Welling (Eds.), *Proceedings of the computer vision–ECCV*. In *Lecture Notes in Computer Science: vol. 9908* (pp. 614–629). Springer.

Lomonaco, V., & Maltoni, D. (2017). CORe50: A new dataset and benchmark for continuous object recognition. In S. Levine, V. Vanhoucke, & K. Goldberg (Eds.), *Proceedings of the 1st annual conference on robot learning*. In *Proceedings of Machine Learning Research: vol. 78* (pp. 17–26). PMLR.

Marsland, S., Shapiro, J., & Nehmzow, U. (2002). A self-organizing network that grows when required. *Neural Networks, 15*, 1041–1058.

McClelland, J., McNaughton, B., & O'Reilly, R. (1995). Why there are complementary learning systems in the hippocampus and neocortex: Insights from the successes and failures of connectionist models of learning and memory. *Psychological Review, 102*, 419–457.

McCloseky, M., & Cohen, N. (1989). Catastrophic interference in connectionist networks: the sequential learning problem. *The Psychology of Learning and Motivation, 24*, 104–169.

Nakamura, Y., & Hasegawa, O. (2017). Nonparametric density estimation based on self-organizing incremental neural network for large noisy data. *IEEE Transactions on Neural Networks and Learning Systems, 28*, 8–17.

Okada, S., Kobayashi, Y., Ishibashi, S., & Nishida, T. (2010). Incremental learning of gestures for human-robot interaction. *AI & Society, 25*, 155–168.

O'Reilly, R., Bhattacharyya, R., Howard, M., & Ketz, N. (2014). Complementary learning systems. *Cognitive Science, 38*, 1229–1248.

Parisi, G. I., Kemker, R., Part, J. L., Kanan, C., & Wermter, S. (2019). Continual lifelong learning with neural networks: A review. *Neural Networks, 113*, 54–71.

Parisi, G. I., Tani, J., Weber, C., & Wermter, S. (2017). Lifelong learning of human actions with deep neural network self-organization. *Neural Networks, 96*, 137–149.

Parisi, G. I., Tani, J., Weber, C., & Wermter, S. (2018). Lifelong learning of spatiotemporal representations with dual-memory recurrent self-organization. *Frontiers in Neurorobotics, 12*, 78.

Parisi, G. I., Weber, C., & Wermter, S. (2015). Self-organizing neural integration of pose-motion features for human action recognition. *Frontiers in Neurorobotics, 9*, 3.

Part, J., & Lemon, O. (2016). Incremental on-line learning of object classes using a combination of self-organizing incremental neural networks and deep convolutional neural networks. In *Proceedings of the workshop on bio-inspired social robot learning in home scenarios at the IEEE/RSJ international conference on intelligent robots and systems (IROS)* (pp. 1–6). Daejeon, Korea

Richardson, F., & Thomas, M. (2008). Critical periods and catastrophic interference effects in the development of self-organizing feature maps. *Developmental Science, 11*, 371–389.

Robins, A. (1995). Catastrophic forgetting, rehearsal, and pseudorehearsal. *Connection Science, 7*, 123–146.

Rousseeuw, P. J., & Croux, C. (1993). Alternatives to the median absolute deviation. *Journal of the American Statistical Association, 88*, 1273–1283.

Rusu, A. A., Rabinowitz, N. C., Desjardins, G., Soyer, H., Kirkpatrick, J., Kavukcuoglu, K., Pascanu, R., & Hadsell, R. (2016). Progressive neural networks. CoRR, abs/1606.04671, pp. 1–14.

Shen, F., & Hasegawa, O. (2006). An incremental network for on-line unsupervised classification and topology learning. *Neural Networks, 19*, 90–106.

Shen, F., & Hasegawa, O. (2008). A fast nearest neighbor classifier based on self-organizing incremental neural network. *Neural Networks, 21*, 1537–1547.

Shen, F., Ogura, T., & Hasegawa, O. (2007). An enhanced self-organizing incremental neural network for online unsupervised learning. *Neural Networks, 20*, 893–903.

Shen, F., Yu, H., Sakurai, K., & Hasegawa, O. (2011). An incremental online semi-supervised active learning algorithm based on self-organizing incremental neural network. *Neural Computing and Applications, 20*, 1061–1074.

Sodhani, S., Chandar, S., & Bengio, Y. (2018). Towards training recurrent neural networks for lifelong learning. CoRR, abs/1811.07017, pp. 1–35.

Souza, V. M. A., Silva, D. F., Gama, J., & Batista, G. E. A. P. A. (2015). Data stream classification guided by clustering on nonstationary environments and extreme verification latency. In *Proceedings of SIAM international conference on data mining* (pp. 873–881).

Street, W. N., & Kim, Y. S. (2001). A streaming ensemble algorithm (SEA) for large-scale classification. In *Proceedings of the 7th ACM SIGKDD international conference on knowledge discovery and data mining* (pp. 377–382).

Sudo, A., Sato, A., & Hasegawa, O. (2009). Associative memory for online learning in noisy environments using self-organizing incremental neural network. *IEEE Transactions on Neural Networks, 20*, 964–972.

Wiwatcharakoses, C., & Berrar, D. (2019). Self-organizing incremental neural networks for continual learning. In *Proceedings of the 28th international joint conference on artificial intelligence* (pp. 6476–6477).

Yoon, J., Yang, E., Lee, J., & Ju Hwang, S. (2017). Lifelong learning with dynamically expandable networks. CoRR, abs/1708.01547, pp. 1–11.

Zhang, H., Xiao, X., & Hasegawa, O. (2014). A load-balancing self-organizing incremental neural network. *IEEE Transactions on Neural Networks and Learning Systems, 25*, 1096–1105.

Zhu, X. (2010). Stream data mining repository. http://www.cse.fau.edu/~xqzhu/stream.html, accessed 1 July 2019.