Continual General Chunking Problem and SyncMap

Danilo Vasconcellos Vargas[*]
Kyushu University
Japan
vargas@inf.kyushu-u.ac.jp

Toshitake Asabuki
OIST
Japan
toshitake.asabuki2@oist.jp

Abstract

Humans possess an inherent ability to chunk sequences into their constituent parts. In fact, this ability is thought to bootstrap language skills to the learning of image patterns which might be a key to a more animal-like type of intelligence. Here, we propose a continual generalization of the chunking problem (an unsupervised problem), encompassing fixed and probabilistic chunks, discovery of temporal and causal structures and their continual variations. Additionally, we propose an algorithm called SyncMap that can learn and adapt to changes in the problem by creating a dynamic map which preserves the correlation between variables. Results of SyncMap suggest that the proposed algorithm learn near optimal solutions, despite the presence of many types of structures and their continual variation. When compared to Word2vec, PARSER and MRIL, SyncMap surpasses or ties with the best algorithm on 77% of the scenarios while being the second best in the remaing 23%.

1 Introduction

Humans are able to rapidly detect patterns in sequences [2], [35]. By detecting and chunking together patterns found, even without a supervised signal, humans are able to classify sounds, images and other information signals [26], [5]. Therefore, this unsupervised learning process is known to bootstrap many of the initial cognitive capabilities such as natural language processing, speech recognition and even image recognition.

Here, motivated by this general learning ability we propose the continual general chunking problem. To evaluate the quality of an algorithm in the proposed problem a set of tests are defined, including chunks with fixed time series, chunks based on graphs with probabilistic transitions, overlapping chunks, chunks with probabilistic cycles, temporal and causal structures identification and continual scenarios. In other words, the continual general chunking problem generalizes the chunking problem to the identification of temporal and causal structures. All this taking into consideration continual learning (change of the underlying structure as well as the probabilistic distribution of variables throughout the experiment).Thus, the proposed problem joins the neuroscientific/psychologic chunking problem to the discovery of causal/temporal structure and unsupervised feature learning of time series.

To tackle the continual general chunking problem, we propose an algorithm, that without any parameter adjustment between problems, works for all the continual general chunking problems and achieves near optimal solutions. In other words, it can cope with probabilistic and fixed chunks as well as causal structures. This algorithm is inspired by Hebbian learning and negative feedback signals. It works by first converting inputs into spikes with slow decaying rate and creating a map on which signals self-organize through neuron group attraction/repeal forces. By creating a dynamic in which signals that activate together or deactivate together are attracted to a common center, the

[*]Lab: http://lis.inf.kyushu-u.ac.jp, homepage: http://danilovargas.org

Preprint. Under review.

self-organizing dynamics are able to create clusters of correlated signals. It is worth noticing that attraction of the activated signals (which is closely related to Hebbian learning) by itself is not enough. Attraction of both activated and deactivated signals are necessary for the dynamics to reach the cohesive behavior described here.

Our contributions. In this paper, a general problem is proposed called Continual General Chunking Problem as well as an algorithm to solve it called SyncMap. The key contributions are as follows:

- We generalize various problems from neuroscience and computer science (i.e., chunking problem, discovery of causal/temporal communities, unsupervised feature learning of time series and their continual variations) into a problem called Continual General Chunking Problem (CGCP). CGCP is defined formally and experiments are developed to evaluate an algorithm's performance. Chunking problem alone encompasses problems from learning image features to sounds as shown in [1]. It originates from detecting repetitive patterns of neural spike sequences, but its primitives are thought to be widely used in the brain. Discovery of causal and/or temporal communities was explored in [33] with applications in [18]. Here we shown how all these problems and their continual variations can be seen as a single one.
- We propose an algorithm for tackling CGCP called SyncMap. SyncMap is a different type of self-organizing map with dynamics of attraction between all nodes that activate or deactivate at the same time. It shares with other self-organizing systems such as Self-Organizing Map [22] and Novelty Map [36] only the idea of using a map as all other dynamics and intent differ. Moreover, its self-organizing dynamics enables it to flexibly respond to changing environments which is a challenge for most algorithms that optimize loss functions.
- Beyond generalizing the problems, we consider (perhaps for the first time) continual variations of them in the CGCP. The challenge here is to respond quickly to the environment, adapting previous learned structures and correlations that have changed. This is motivated by the constant adaptation spotted in neural cells which can rapidly switch behavior according to environmental changes [8].
- Experiments on fixed chunks, probabilistic chunks and temporal structures suggest that SyncMap reaches near optimal solutions. The same is true for continual variations of them, i.e., when such probabilistic chunks or temporal structures change throughout the experiment. Moreover, we extend the tests for detecting temporal structures of real world scenarios.

2 Related Work

Chunking. Through the process, called "chunking", the brain attains compact representation of sequences, which is thought to reduce the complexity of temporal information processing and associated cost [31]. Chunking in the brain computation is crucial to achieve high-order functions that require hierarchical sequence processing, such as motor-skill learning [14], [19] and language acquisition [4], [13]. Cognitive experiments suggests that children learn words as chunks [32]. This process is thought to contribute to higher-order learning process, and be fundamental mechanisms that children identify words from speech [15], [9]. Recent study with human subjects has shown that chunking occurs even when the sequence has co-occurring structure rather than a repetitive pattern [33]. The influential chunking (word segmentation) algorithm, PARSER has been proposed, which extract the all frequently appeared n-grams within the sequence [28]. Despite PARSER works well for simple chunking tasks, yet fail to detect chunks if the transition probability over all elements are uniform[33]. Recently, biology-inspired sequence learning model, called Minimization of Regularised Information Loss (MRIL) has been proposed. MRIL is applicable to broad range of chunking task including uniform transitions. These studies suggest that chunking is a fundamental process, yet the mechanism of which is still elusive. Albeit the multitude of applications of chunking and the wide interest of neuroscientists on the subject, this subject was not fully explored from a machine learning perspective. Chunking can be found in some articles focusing on natural language processing. Sometimes chunking is modeled as a supervised pre-processing step [37]. In other cases, it is used for unsupervised grammar induction [30]. In both cases the problem taking into account differs from the original task independent one defined in neuroscience.

Unsupervised learning for sequences. Unsupervised learning for sequences usually extract features which can predict future input [24], [7],[17], [23]. These features, albeit descriptive of the sequences and useful for classification of sequences, do not uncover the chunks present. Related to unsupervised learning for sequences, contrastive predictive encoding (CPE) is a peculiar learning algorithm which makes use of a probabilistic contrastive loss, inducing the latent space to encode maximally useful information in its representation [16]. It requires the sequence of samples to have some sort of order and could use a general chunking algorithm to present images that are coherent to a certain class. In this manner, CPE and its applied problem formulation is complementary rather than competing with the problem proposed here. Moreover, self-organizing maps and their variations tackle a related but different problem [11]. They can learn topological representations of time series and static data while chunks are temporal correlations between variables, therefore, their intent differ.

Word embeddings. To transform words and paragraphs into vectors of numbers, word embeddings are used in natural language processing [20], [24]. Some of them are enriched with information specific to natural language processing such as FastText [3] or are contextualized word embeddings such as ELMo [29] and BERT [10]. However, prediction-based word2vec embeddings [24] and co-occurrence matrix based GloVe embeddings [27] can be also used for more general problems similar to the one presented in this paper.

Causal graphs. As part of the general chunk problem, this paper aims to discover temporal and causal structures in sequences which are related to causal graphs. The detection of temporal structures has many practical applications and can be used as well for facilitating the learning of causal graphs by identifying confounders, identifying correlated variables, among others [21], [6]. Interestingly, some terms from causality studies also take place in chunking problems albeit in different ways. For example if variables x and y are confounded by a given variable z, if both x and y pertains to a different chunk, z is an overlapping variable. This overlapping variable can be identified as a different chunk which should facilitate the learning of causal graphs.

3 Continual General Chunking Problem

We define continual general chunking as the problem of extracting co-occurring states from a time series of which the underlying generation process changes depending on the task as well as throughout the task. Here, we first describe the input time series structure considered in this problem.

Our input sequence consists of state variables, transitioning by first-order Markov chain, of which the element of transition matrix is defined as: $\pi_{ab} = Pr[s_{t+1} = b|s_t = a]$, where s_t is the state variable at time t and a and b are the labels of states. Here, note that each state s_t is a vector. In our sequence, each state belongs to either a fixed chunk or a probabilistic one. In a fixed chunk, given the state at the current time, the state at the next time step is selected deterministically. Let a and b be elements of a fixed chunk, with the direction a to b, then the transition matrix should satisfy: $\pi_{ab} = 1$. Note that since $\Sigma_b \pi_{ab} = 1$, the state i never transits to other states.

In the probabilistic chunk, the state at the next time step is stochastically selected from the set of states in the chunk to which the current state belongs. Let a be the state within a probabilistic chunk and $V(a)$ the set of states that state a connects. Then, the transition from a is given as:

$$\pi_{ab} = \begin{cases} 1/|V(a)|, & b \in V(a) \\ 0, & else, \end{cases} \tag{1}$$

where $|V(a)|$ is the size of the set.

The co-occurring states in the sequences of which the transition probabilities between any states are uniform are known as temporal communities. Although our input structure is similar to temporal community structure, it also includes fixed chunks with nonuniform transition probabilities. Therefore, we use the terminology "probabilistic chunk".

In our continual setting, the causal structure behind the generation process, hence the transition matrix can change over time. In this paper, we assume that the input sequence can have m structures, and the label of transition matrixes, that determines the causal structure is termed as "tasks". Assume the set of transition matrixes, $\Pi = \{\pi^{(1)}, \pi^{(2)}, \ldots, \pi^{(m)}\}$. At the time when the tasks are switched, one task is randomly selected from the set Π, and the time series is continuously generated. We set

$m = 2$ and since the generation process itself changes, the set of state variables that constitute each chunk also change.

4 SyncMap

Inspired by how Hebbian volume learning complements Hebbian, taking into consideration the effects of nearby neurons learning[25] as well as how feedback systems affect the learning of algorithms for chunking problems [1]; here we propose a map in which neurons in a group can learn together to represent interrelated concepts . The main idea here is to use a simplified Hebbian learning together with feedback dynamics to create a projection that encodes the probability of two variables activating at the same time with their distances in this new generated space, i.e., encoding the relative probabilistic correlation between variables as distances between them. Thus, variables that activate together will have their respective projections drawn to their middle point while variables that do not activate together will be, consequently, pulled away from each other.

SyncMap is divided into two steps: (a) the activation of nodes and their update inside the map (b) a clustering phase on the learned map, revealing their clusters. The following subsections explain these steps in detail.

4.1 Input Encoding

Let $s_t = s_{1,t}, s_{2,t}, ..., s_{n,t}$ be the set of state values at time t for $s_t \in \{0, 1\}^n : \sum^n s_t = 1$. The input encoding is a exponentially decaying vector x_t with the same size as the number of states:

$$x_{i,t} = \begin{cases} s_{i,ta} * e^{-0.1*(t-ta)}, & ta < m * tstep \\ 0, & otherwise \end{cases} \tag{2}$$

where ta is the most recent state transition to state s_i. State transitions happen every $tstep$ steps and variables which have their time of activation greater than $m * tstep$ are set to 0. Consequently, the system can only remember the last m variables presented. All experiments here are conducted using $m = 2$ and $tstep = 10$.

This representation of input can also be encoded differently without any change in result, e.g., by storing the last m inputs directly. In other words, the details of implementation for the input is not important for the method itself. It suffices to remember the last m states.

4.2 Dynamics

First all inputs $x_{i,t}$ have a corresponding weight $w_{i,t}$ initialized to a random position in the map $w \in \mathbb{R}^k$, creating a pair $(x_{i,t}, w_{i,t})$. $w_{i,t}$ is a k dimensional variable and k is a parameter defined a priori (the dimension of the map).

Every time a new input x_t is presented, each of its constituents $x_{i,t}$ are divided into activated or recently activated (positive) and non-recently activated (negative) inputs. Specifically, all inputs are converted into two sets: positive PS_t and negative NS_t sets. Inputs with value greater than 0.1 are a member of PS_t. Otherwise, inputs are a member of NS_t. In other words, the following holds true:

$$PS_t = \{i | x_{i,t} > 0.1\} \tag{3}$$
$$NS_t = \{i | x_{i,t} \leq 0.1\} \tag{4}$$

If the cardinality of both sets are greater than one, i.e. $|PS_t| > 1$ and $|NS_t| > 1$; then the center of both sets are calculated as follows:

$$cp_t = \frac{\sum_{i \in PS_t} w_{i,t}}{|PS|} \tag{5}$$

$$cn_t = \frac{\sum_{i \in NS_t} w_{i,t}}{|NS|}, \tag{6}$$

in which cn_t and cp_t are respectively the centers of NS_t and PS_t sets. If the cardinality of both sets are not greater than nothing is updated in this iteration.

After the center of both sets are calculated, the position $w_{i,t}$ of each input is updated.

$$\phi(i,t) = \begin{cases} 1, & i \in PS_t \\ 0, & i \in NS_t \end{cases} \tag{7}$$

$$w_{i,t+1} = w_{i,t} + \alpha * \frac{\phi(i,t) * cp_t + (1-\phi(i,t)) * cn_t}{\|w_{i,t} - cp_t\|} \tag{8}$$

where α is the learning rate. Subsequently, all values of $w_{i,t+1}$ are normalized to be within a hypersphere of radius $r = 1$.

4.3 Clustering

The clustering step happens after the dynamics described in Section 4 is repeated for each input. In the clustering step, the projected map w is clustered to determine effectively the chunks/communities. Here, DBSCAN [34] is used for this procedure because it does not require the number of clusters as input. The required parameter is the density of clusters eps, which is somewhat fixed for a given hypersphere of radius r, and the minimum size of each cluster mc which can also be set independent of problem. eps and mc are set to respectively 3 and 2. Having said that, other clustering algorithms together with the use of clustering analysis techniques for discovering number of clusters should be equally or even more effective. For simplicity we are narrowing the scope of this paper to DBSCAN alone.

5 Experiments

The experiments compose a total of nine different tests encompassing fixed chunks, mixed structures, their continual variations, long chunks, overlapped chunks and real world scenarios. Both overlapped chunks and real world scenarios have two distinctive tasks.

Settings. In all experiments, the learning algorithm is first exposed to 100000 samples of the problem and followed by an extraction of the chunks predicted. This is also true for continual variations, where the problem changes after 100000 samples are inputted, with the second problem also presenting 100000 samples before the run is finished. All tests are run on a MacBook Pro 10.15.5 2.3Ghz 16GB laptop as they demand little computational effort.

Here we compare four distinct algorithms: SyncMap (proposed one), MRIL, PARSER and Word2vec. SyncMap's parameters $'alpha$ and k are fixed to respectively 0.1 and 3. Regarding the PARSER algorithm, since it finds possible n-grams, hence not whole chunks, we first excluded the unnecessary long n-grams ($n > 6$), and concatenated the rest of the short segments, of which share the same element. These resultant segments were regarded as "chunks" that PARSER extracted. To evaluate how a word embedding algorithm would fair in CGCP, we include a skip-gram Word2vec embedding modified to work in the context of CGCP. A dense deep neural network model was used as model for the Word2vec with a latent dimension of 3 and an output layer with softmax and size equal to the number of inputs. The chosen training parameters are 10 epochs, $1e-3$ learning rate and 64 batch size with a mean squared error as loss. These parameters were the best performing without unnecessary slowdown after a dozen trials. A window of 100 steps was used to calculate the output probability of skip gram. The input was kept the same as SyncMap, since variations of non-decaying ones performed (perhaps surprisingly) worse. Therefore, the window for Word2vec include 10 state transitions; equivalent of 100 time steps. Regarding MRIL, we used five output neurons for all tasks, with the learning rate $1e-3$. For comparison, we used the same decaying input as SyncMap, rather than the Poisson spiking input used in the original setting of MRIL. We grouped the output neurons showing correlation larger than 0.5, and determining chunk by assigning a index of groups that maximally respond to each input.

Fixed Chunks. The problem considered here has four fixed chunks each containing three different variables. Transition between chunks happen at the end of the chunk sequence, i.e., after the third variable inside the chunk is presented. A chunk can transition to any other chunk with equal probability.

Overlapped and Long Chunks. In one hand, overlapped chunks evaluate the capability of systems to perceive chunks that share some variables. On the other hand, long chunks evaluate if the frequency of chunks affect the system. Both overlapped and long chunks are probabilistic chunks.

For the overlapped chunks, two problems are tested: overlap1 and overlap2. Overlap1 have two chunks composed respectively of variables a,b,c,d,e and d,e,f,g,h while overlap2 has chunks with respectively variables a,b,c,d,e and a,b,c,d,e,f,g,h,i,j. In other words, they either share variables of are a subset of each other. For the long chunk experiment, each chunk has four non-repeating variables a,b,c,d and e,f,g,h presented randomly each step, however, the duration of the first chunk is four transitions while the second has a stochastic duration of $5+unif(0,20)$ transitions. $unif(min,max)$ defines a uniform distribution within the half-open interval $[min,max)$.

Mixed Structures. In this problem, both probabilistic chunks together with fixed one are present in the system. The graph with the structure and transitions of the problem is shown in the left of Fig. 1. It has 2 probabilistic chunks and one fixed chunk.

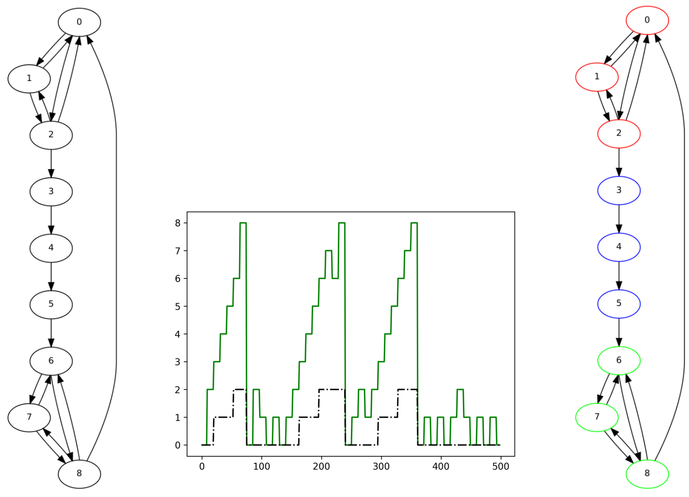

Figure 1: Problem description (up left), learned input-output map (up middle) and respective learned clustering (up right). The problem is a cyclic directed graph with node numbers as indicated. Each node number consists of a given input which is activated and inputted by a random walk over the graph (see the Section 4.1 for more details). The learned input-output map is as follows: input (red line) and learned output (dash-dotted line). In direct correspondence to the learned output, it is possible to cluster the nodes with the colors shown (up right). The learned map is shown in Fig 2.

Continual Variations. Two continual variations of previous problems are proposed: continual fixed and continual mixed. They are variations of respectively the fixed chunk and the mixed structures. In both variations, variables are permuted between chunks when tasks are changed. For the continual variation of the fixed chunk, in the first task the configuration of chunks is (a,b,c), (d,e,f), (g,h,i), (j,k,l). In the second task, the fixed chunks become respectively (a,k,i), (g,e,j), (d,h,c) and (f,b,l). Regarding the continual mixed problem, Fig. 3 shows how the structure of the problem changes throughout.

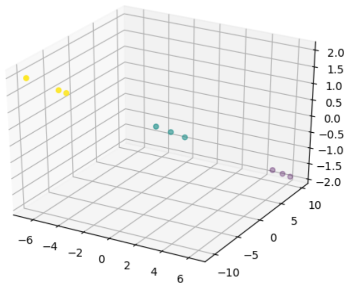

Figure 2: The learned map by SyncMap together with colors showing the chunks learned for problem described in Fig 1.

Real world scenarios. We test in two variations of a real world scenario. Specifically, the recognition of probabilistic chunks in the first-order Markov model of theme transitions for humpback whales' song types [12]. Since the transition between nodes are not given, they are considered equally probable. Sequence 1 and 2 are defined as respectively the graphs A and B from Fig 1 from supplementary works.

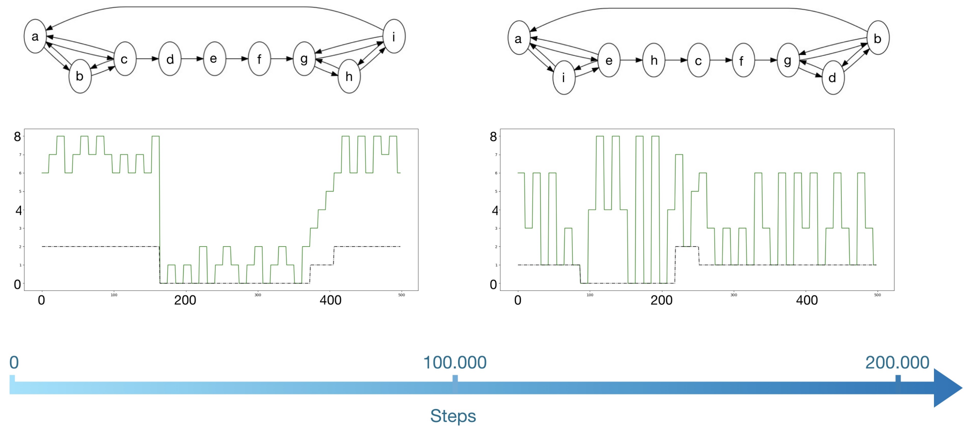

Figure 3: Illustration of the continual version of the mixed structures problem and an example of SyncMap's learning output. The structure changes after 100000 steps and then the experiment ends after 200000 steps. The upper graphs show the start (left) and end (right) problem structures and their respective variables. The start and end outputs of SyncMap are shown at the bottom.

Table 1: Mutual information comparison over SyncMap, PARSER, Word2vec and MRIL in Fixed chunks, Long chunks and Mixed structures settings. Best results and results within the standard deviation of the best are in bold.

Model	**Fixed Chunks**	**Long Chunks**	**Mixed Structures**
PARSER	0.95±0.07	0.16±0.27	0.36±0.40
Word2vec	**1.0**±**0.0**	0.68±0.22	0.78±0.07
MRIL	0.86±0.13	0.76±0.17	0.85±0.16
Ours: SyncMap	0.97±0.09	**0.97**±**0.06**	**0.95**±**0.06**

5.1 Results and Analysis

For all tests, we measured the normalized mutual information scores for Wor2vec, MRIL, PARSER and SyncMap (Tables 1, 2 and 3). The proposed algorithm SyncMap performed better overall. It surpassed or performed within the standard deviation of other algorithms in 7 out of the 9 tests. PARSER and Word2vec performed similarly, but had only 3 out of the 9 results that were comparable with the others. MRIL was better in only one of the tests but if SyncMap is removed from the comparison it performs slightly better than both PARSER and Word2vec. In other words, SyncMap and MRIL are both general algorithms while Word2vec and PARSER have some specific problems they are very good at.

Regarding SyncMap, it performs similarly for long chunks and mixed structures (Table 1). Long chunks frequency is less of an issue because once variables are sufficient far away in the projected map, the update becomes weaker as well as deactivating together has also a similar attraction force. Moreover, SyncMap considers only one-to-one correlations and create groups through the emergent attraction/repulsion behavior, consequently, the nature of the chunk or structure do not matter in this regard. Continual variations of the problems (Table 2) suggests that SyncMap is capable of adapting to changes in the structure without any observed deleterious effect. This is expected since SyncMap steadily updates the correlation between variables projected into the map w. Once the dynamics reach an equilibrium the updates affect less the map distribution, however, a change in the underlying problem structure affects the place of attractors and therefore naturally put the system in an unstable state initiating the adaptation. SyncMap performs better than the other algorithms in overlapped chunks problems (Table 3). However, there is still ground for improvement here as it

Table 2: Mutual information comparison over SyncMap, PARSER, Word2vec and MRIL in continual fixed and continual mixed settings. Best results and results within the standard deviation of the best are in bold.

Model	**Continual fixed**		**Continual mixed**	
	Task 1	Task 2	Task 1	Task 2
PARSER	0.97±0.06	**0.96±0.07**	0.47±0.40	0.28±0.38
Word2vec	1.0±0.0	0.29±0.20	0.76±0.08	0.0±0.0
MRIL	0.80±0.16	0.53±0.12	0.85±0.15	0.32±0.16
Ours: SyncMap	0.93±0.13	**0.89±0.15**	0.97±0.04	**0.99±0.03**

Table 3: Mutual information comparison over SyncMap, PARSER, Word2vec and MRIL in Overlapped chunks and Real world scenarios. Sequence1 and sequence2 correspond to supplementary work's Fig 1's A and B, respectively (problem defined in [12]). Best results and results within the standard deviation of the best are in bold.

Model	**Overlap1**	**Overlap2**	**Sequence1**	**Sequence2**
PARSER	**0.77±0.42**	0.30±0.46	**0.27±0.11**	0.57±0.1
Word2vec	0.21±0.28	0.06±0.09	**0.28±0.03**	**0.79±0.08**
MRIL	0.03±0.18	0.0±0.0	**0.38±0.11**	0.51±0.10
Ours: SyncMap	**0.70±0.19**	**0.64±0.10**	**0.39±0.16**	0.61±0.06

cannot represent a hierarchical structure. In real world scenarios, SyncMap performs equally to all other in Sequence1 and is the second best in Sequence2. Increasing past state memory m should enable better performance.

Word2vec usually generates a map in which variables are more dispersed than the one produced by SyncMap which makes clustering difficult. For long chunks and mixed structures, the map becomes fuzzier and therefore the MI score drops accordingly. Moreover, it does not have a built-in adaptation which makes changes in the problem structure cause probabilities of nearby nodes to even out. Big overlaps do a similar effect, probabilities of nearby nodes are similar, it ends up recognizing all nodes as a single chunk. Real world sequences have mixed results, being the best in one and second worse on the other.

Regarding PARSER, it extracts chunks based on the bias of transition probabilities, therefore, performance deteriorates in time series with equal probability such as our long chunk and mixed structures. In the continual variations, PARSER performs well in both tasks involving the fixed problem since forgetting phase during learning enables it to adapt to the new environment (i.e., second task). Unlike the fixed case, it failed to learn in the mixed structures even in the first task, as shown previously. MI score for overlap2 becomes less than half of overlap1. We speculate this is because sequence of overlap2 has higher fraction of overlaps than sequence1. Therefore, the transition becomes much uniform which deteriorates the performance of PARSER. Since real world tasks have uniform transition part, the MI was lower than the fixed case.

Since MRIL detects spatio-temporal correlation over inputs, it showed better performance than PARSER and Word2vec if the sequence has uniform probability. MRIL learns not only the feedforward weights, but also lateral inhibition weight with the spike train correlation of output. Since each state in our sequence was presented only 10 time steps, MRIL failed to detect spike correlation by its slower timescales, hence showed MI less than 0.9 in all problems. Similarly, the MI of the second task in continual problems was lower than that of first one because lateral inhibition could not be trained efficiently. In the overlap tasks, since MRIL creates chunks including both non-overlapped and overlapped states, the MI was significantly low. In the real world problems, the MI in sequence1 was almost comparable to that of SyncMap, yet lowest in sequence2.

6 Conclusions

In this paper, we proposed both a general problem called CGCP and an algorithm called SyncMap which outperforms or ties with competitive algorithms from neuroscience and machine learning in 7 out of 9 tests. We expect that variations of SyncMap should further increase the gap to other algorithms in CGCP and will probably become a strong alternative to applications from natural language processing to image recognition. Future directions will investigate problems with noise, hierarchy and causal relations as well as tasks specific to language processing and image/action recognition.

Acknowledgment

This work was supported by JST, ACT-I Grant Number JP-50243 and JSPS KAKENHI Grant Number JP20241216, Japan.

References

[1] Toshitake Asabuki and Tomoki Fukai. Somatodendritic consistency check for temporal feature segmentation. *Nature communications*, 11(1):1–13, 2020.

[2] Tristan A. Bekinschtein, Stanislas Dehaene, Benjamin Rohaut, François Tadel, Laurent Cohen, and Lionel Naccache. Neural signature of the conscious processing of auditory regularities. *Proceedings of the National Academy of Sciences*, 106(5):1672–1677, 2009.

[3] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics*, 5:135–146, 2017.

[4] Marco Buiatti, Marcela Peña, and Ghislaine Dehaene-Lambertz. Investigating the neural correlates of continuous speech computation with frequency-tagged neuroelectric responses. *Neuroimage*, 44(2):509–519, 2009.

[5] Hermann Bulf, Scott P Johnson, and Eloisa Valenza. Visual statistical learning in the newborn infant. *Cognition*, 121(1):127–132, 2011.

[6] Ruichu Cai and Feng Xie. Triad constraints for learning causal structure of latent variables. *Advances in neural information processing systems*, 2019.

[7] David Clark, Jesse Livezey, and Kristofer Bouchard. Unsupervised discovery of temporal structure in noisy data with dynamical components analysis. In *Advances in Neural Information Processing Systems*, pages 14267–14278, 2019.

[8] Johannes C Dahmen, Peter Keating, Fernando R Nodal, Andreas L Schulz, and Andrew J King. Adaptation to stimulus statistics in the perception and neural representation of auditory space. *Neuron*, 66(6):937–948, 2010.

[9] Stanislas Dehaene, Florent Meyniel, Catherine Wacongne, Liping Wang, and Christophe Pallier. The neural representation of sequences: from transition probabilities to algebraic patterns and linguistic trees. *Neuron*, 88(1):2–19, 2015.

[10] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2018.

[11] Vincent Fortuin, Matthias Hüser, Francesco Locatello, Heiko Strathmann, and Gunnar Rätsch. Som-vae: Interpretable discrete representation learning on time series. In *ICLR 2019*, 2019.

[12] Ellen C Garland, Luke Rendell, Luca Lamoni, M Michael Poole, and Michael J Noad. Song hybridization events during revolutionary song change provide insights into cultural transmission in humpback whales. *Proceedings of the National Academy of Sciences*, 114(30):7822–7829, 2017.

[13] Timothy Q Gentner, Kimberly M Fenn, Daniel Margoliash, and Howard C Nusbaum. Recursive syntactic pattern learning by songbirds. *Nature*, 440(7088):1204–1207, 2006.

[14] Ann M Graybiel. The basal ganglia and chunking of action repertoires. *Neurobiology of learning and memory*, 70(1-2):119–136, 1998.

[15] Jessica F Hay, Bruna Pelucchi, Katharine Graf Estes, and Jenny R Saffran. Linking sounds to meanings: Infant statistical learning in a natural language. *Cognitive psychology*, 63(2):93–106, 2011.

[16] R Devon Hjelm, Alex Fedorov, Samuel Lavoie-Marchildon, Karan Grewal, Phil Bachman, Adam Trischler, and Yoshua Bengio. Learning deep representations by mutual information estimation and maximization. *arXiv preprint arXiv:1808.06670*, 2018.

[17] Aapo Hyvarinen and Hiroshi Morioka. Unsupervised feature extraction by time-contrastive learning and nonlinear ica. In *Advances in Neural Information Processing Systems*, pages 3765–3773, 2016.

[18] Pengfei Jiao, Wei Yu, Wenjun Wang, Xiaoming Li, and Yueheng Sun. Exploring temporal community structure and constant evolutionary pattern hiding in dynamic networks. *Neurocomputing*, 314:224–233, 2018.

[19] Xin Jin, Fatuel Tecuapetla, and Rui M Costa. Basal ganglia subcircuits distinctively encode the parsing and concatenation of action sequences. *Nature neuroscience*, 17(3):423–430, 2014.

[20] F. K. Khattak, S. Jeblee, C. Pou-Prom, M. Abdalla, C. Meaney, and F. Rudzicz. A survey of word embeddings for clinical text. *Journal of Biomedical Informatics*, 10(4):100057., 2019.

[21] Murat Kocaoglu, Amin Jaber, Karthikeyan Shanmugam, and Elias Bareinboim. Characterization and learning of causal graphs with latent variables from soft interventions. In *Advances in Neural Information Processing Systems*, pages 14346–14356, 2019.

[22] Teuvo Kohonen. Essentials of the self-organizing map. *Neural networks*, 37:52–65, 2013.

[23] Qi Lei, Jinfeng Yi, Roman Vaculin, Lingfei Wu, and Inderjit S Dhillon. Similarity preserving representation learning for time series clustering. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence*, pages 2845–2851. AAAI Press, 2019.

[24] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[25] Graeme J Mitchison and Nicholas V Swindale. Can hebbian volume learning explain discontinuities in cortical maps? *Neural computation*, 11(7):1519–1526, 1999.

[26] Gergő Orbán, József Fiser, Richard N Aslin, and Máté Lengyel. Bayesian learning of visual chunks by human observers. *Proceedings of the National Academy of Sciences*, 105(7):2745–2750, 2008.

[27] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. *Glove: Global vectors for word representation*. Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP), 2014.

[28] Pierre Perruchet and Annie Vinter. Parser: A model for word segmentation. *Journal of memory and language*, 39(2):246–263, 1998.

[29] M. E. Peters, M. Neumann, M. Iyyer, M. Gardner, C. Clark, K. Lee, and L. Zettlemoyer. Deep contextualized word representations. In *Proceedings of NAACL-HLT*, pages 2227–2237. 2018.

[30] Elias Ponvert, Jason Baldridge, and Katrin Erk. Simple unsupervised grammar induction from raw text with cascaded finite state models. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1*, pages 1077–1086. Association for Computational Linguistics, 2011.

[31] Pavan Ramkumar, Daniel E Acuna, Max Berniker, Scott T Grafton, Robert S Turner, and Konrad P Kording. Chunking as the result of an efficiency computation trade-off. *Nature communications*, 7(1):1–11, 2016.

[32] Jenny Saffran and Diana Wilson. From syllables to syntax: Multilevel statistical learning by 12-month-old infants. *Infancy*, 4:273–284, 04 2003.

[33] Anna C Schapiro, Timothy T Rogers, Natalia I Cordova, Nicholas B Turk-Browne, and Matthew M Botvinick. Neural representations of events arise from temporal community structure. *Nature neuroscience*, 16(4):486, 2013.

[34] Erich Schubert, Jörg Sander, Martin Ester, Hans Peter Kriegel, and Xiaowei Xu. Dbscan revisited, revisited: why and how you should (still) use dbscan. *ACM Transactions on Database Systems (TODS)*, 42(3):1–21, 2017.

[35] Melanie Strauss, Jacobo D. Sitt, Jean-Remi King, Maxime Elbaz, Leila Azizi, Marco Buiatti, Lionel Naccache, Virginie van Wassenhove, and Stanislas Dehaene. Disruption of hierarchical predictive coding during sleep. *Proceedings of the National Academy of Sciences*, 112(11):E1353–E1362, 2015.

[36] Danilo Vasconcellos Vargas, Hirotaka Takano, and Junichi Murata. Novelty-organizing team of classifiers in noisy and dynamic environments. In *2015 IEEE Congress on Evolutionary Computation (CEC)*, pages 2937–2944. IEEE, 2015.

[37] Feifei Zhai, Saloni Potdar, Bing Xiang, and Bowen Zhou. Neural models for sequence chunking. In *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

7 Appendix A

Experiments on real world problems used the graph defined by Figure 4.

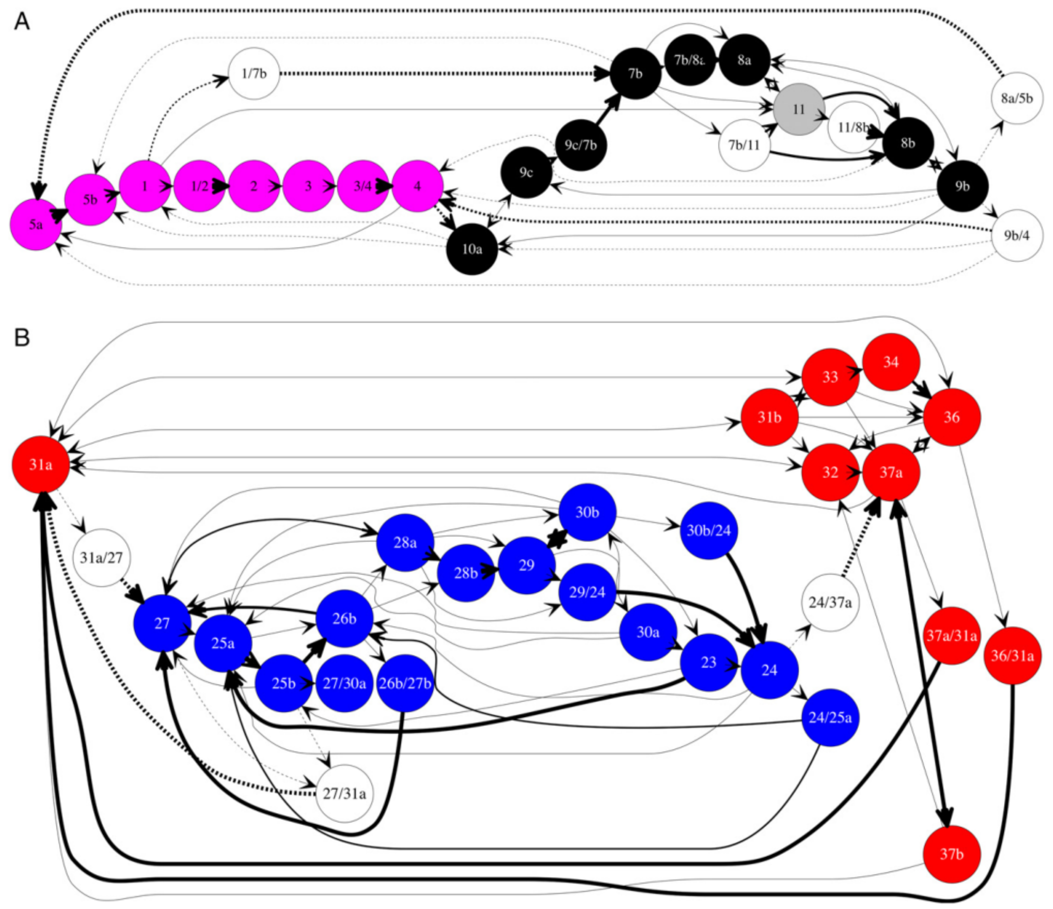

Figure 4: First-order Markov Chain of humpback whales' song types [12].

8 Appendix B - Mapping Comparison (Encoding / Word Embedding)

Figure 5 shows the difference between maps learned for both SyncMap and Word2vec. In general, SyncMap's learned map are less widely spread and therefore clusters, if present, are clearer. Clustering in this space is also made easier.

9 Appendix C - Time Analysis

Table 4 shows the time required to compute each of the methods. SyncMap is the second fastest after PARSER. It is impressive that SyncMap can be faster than word2vec even when word2vec is probably the algorithm which

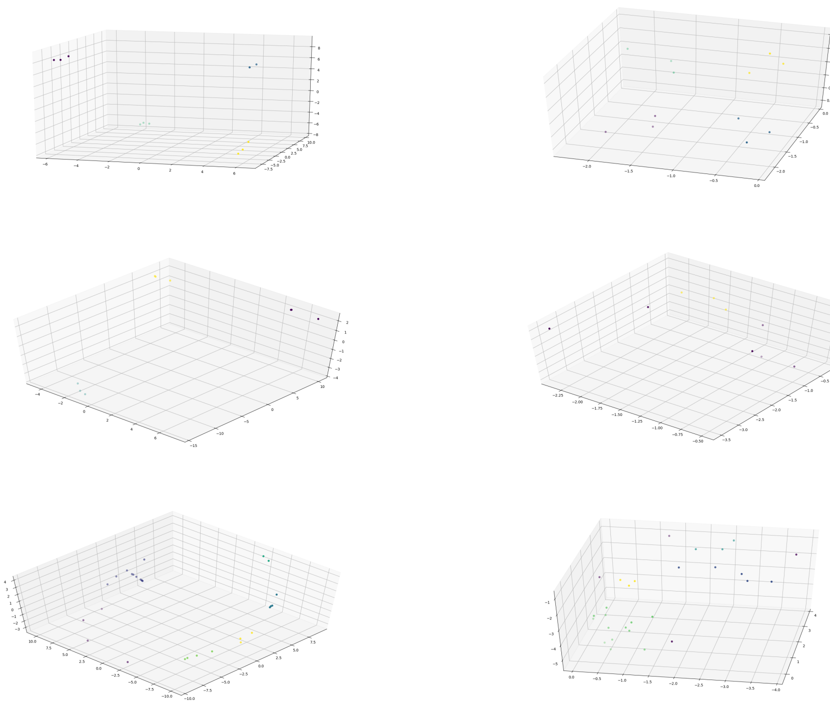
Figure 5: Examples of maps learned for SyncMap (left column) and Word2vec (right column). Rows from top to bottom show tests on respectively fixed chunks, mixed chunks and sequence2.

Table 4: Computation time comparison over SyncMap, PARSER, Word2vec and MRIL.

Model	SyncMap	PARSER	Word2vec	MRIL
Computation Time[s]	8.60±0.21	1.74±0.05	13.832±0.54	25.5±0.56

has the highest investment on efficiency improvement from the algorithms tested. Both SyncMap and MRIL should improve considerably if parallelization, GPGPU programming and other techniques are employed. For example, all neurons in SyncMap can be updated at the same time.

10 Appendix D - SyncMap's Parameter Variations

Tables 5, 6 and 7 show the performance of SyncMap on the same experiments but with different settings. Results suggest that SyncMap is robust to changes in parameters, with mostly smooth changes.

Table 5: Mutual information of several SyncMap variations in continual settings. The variables inside brackets have the following meanings. The first value is the adaptation rate. The second value modifies the adaptation rate, it can be either fixed (value: fixed) or multiplied by the number of outputs (value: out). The third value indicates the dimensions of map w.

Model	**Continual fixed**		**Continual mixed**	
	Task 1	Task 2	Task 1	Task 2
Ours: SyncMap (0.1, fixed, 3d)	0.93±0.13	0.89±0.15	0.97±0.04	0.99±0.03
Ours: SyncMap (0.01, fixed, 3d)	0.99±0.01	1.0±0.0	0.99±0.03	0.92±0.08
Ours: SyncMap (0.01, out, 3d)	0.89±0.11	0.85±0.15	0.99±0.03	0.98±0.05
Ours: SyncMap (0.001, out, 3d)	0.98±0.06	0.96±0.18	0.98±0.05	0.92±0.07
Ours: SyncMap (0.1, fixed, 2d)	0.70±0.08	0.69±0.06	0.97±0.07	0.98±0.04
Ours: SyncMap (0.01, fixed, 2d)	0.91±0.13	0.80±0.34	0.96±0.05	0.93±0.06
Ours: SyncMap (0.01, out, 2d)	0.81±0.13	0.80±0.13	0.98±0.04	0.99±0.03
Ours: SyncMap (0.001, out, 2d)	0.86±0.17	0.70±0.37	0.97±0.07	0.93±0.08

Table 6: Mutual information of several SyncMap variations in Fixed chunks, Long chunks and Mixed structures settings. The variables inside brackets have the following meanings. The first value is the adaptation rate. The second value modifies the adaptation rate, it can be either fixed (value: fixed) or multiplied by the number of outputs (value: out). The third value indicates the dimensions of map w.

Model	**Fixed Chunks**	**Long Chunks**	**Mixed Structures**
Ours: SyncMap (0.1, fixed, 3d)	0.97±0.09	0.97±0.06	0.95±0.06
Ours: SyncMap (0.01, fixed, 3d)	0.99±0.06	0.84±0.13	0.98±0.04
Ours: SyncMap (0.01, out, 3d)	0.90±0.13	0.96±0.05	0.98±0.05
Ours: SyncMap (0.001, out, 3d)	0.96±0.10	0.90±0.12	0.98±0.05
Ours: SyncMap (0.1, fixed, 2d)	0.71±0.10	0.97±0.06	0.97±0.04
Ours: SyncMap (0.01, fixed, 2d)	0.88±0.17	0.86±0.12	0.97±0.05
Ours: SyncMap (0.01, out, 2d)	0.81±0.14	0.97±0.05	0.99±0.03
Ours: SyncMap (0.001, out, 2d)	0.88±0.17	0.82±0.15	0.97±0.06

Table 7: Mutual information of several SyncMap variations in Overlapped chunks and Real world scenarios. The variables inside brackets have the following meanings. The first value is the adaptation rate. The second value modifies the adaptation rate, it can be either fixed (value: fixed) or multiplied by the number of outputs (value: out). The third value indicates the dimensions of map w.

Model	**Overlap1**	**Overlap2**	**Sequence1**	**Sequence2**
Ours: SyncMap (0.1, fixed, 3d)	0.70±0.19	0.64±0.10	0.39±0.16	0.61±0.06
Ours: SyncMap (0.01, fixed, 3d)	0.70±0.16	0.71±0.10	0.34±0.1	0.51±0.07
Ours: SyncMap (0.01, out, 3d)	0.80±0.15	0.61±0.05	0.33±0.11	0.56±0.07
Ours: SyncMap (0.001, out, 3d)	0.70±0.14	0.74±0.14	0.37±0.11	0.56±0.07
Ours: SyncMap (0.1, fixed, 2d)	0.69±0.22	0.62±0.08	0.38±0.12	0.63±0.05
Ours: SyncMap (0.01, fixed, 2d)	0.55±0.26	0.67±0.09	0.32±0.12	0.45±0.07
Ours: SyncMap (0.01, out, 2d)	0.80±0.18	0.64±0.09	0.34±0.12	0.57±0.06
Ours: SyncMap (0.001, out, 2d)	0.55±0.27	0.68±0.11	0.38±0.12	0.55±0.09