LETTER **Communicated by Claude Berrou**

On Neural Associative Memory Structures: Storage and Retrieval of Sequences in a Chain of Tournaments

Asieh Abolpour Mofrad
asieh.abolpour-mofrad@oslomet.no
Selmer Center, Department of Informatics, University of Bergen, 5020 Bergen, Norway

Samaneh Abolpour Mofrad*
Samaneh.Abolpour.Mofrad@hvl.no
Department of Computer Science, Electrical Engineering, and Mathematical Sciences, Western Norway University of Applied Sciences, 5063 Bergen, Norway, and Mohn Medical Imaging and Visualization Center, Haukeland University Hospital, 5021 Bergen, Norway

Anis Yazidi
Anis.Yazidi@oslomet.no
Department of Computer Science, OsloMet, Oslo Metropolitan University, 0130 Oslo, Norway and Department of Plastic and Reconstructive Surgery, Oslo University Hospital, 0318 Oslo, Norway

Matthew Geoffrey Parker
Matthew.Parker@ii.uib.no
Selmer Center, Department of Informatics, University of Bergen, 5020 Bergen, Norway

Associative memories enjoy many interesting properties in terms of error correction capabilities, robustness to noise, storage capacity, and retrieval performance, and their usage spans over a large set of applications. In this letter, we investigate and extend tournament-based neural networks, originally proposed by Jiang, Gripon, Berrou, and Rabbat (2016), a novel sequence storage associative memory architecture with high memory efficiency and accurate sequence retrieval. We propose a more general method for learning the sequences, which we call feedback tournament-based neural networks. The retrieval process is also extended to both directions: forward and backward—in other words, any large-enough segment of a sequence can produce the whole sequence. Furthermore, two retrieval algorithms, cache-winner and explore-winner, are introduced to increase the retrieval performance. Through simulation results, we shed light on the strengths and weaknesses of each algorithm.

*Equally contributed with the first author.

Neural Computation **33, 2550–2577** (2021)
https://doi.org/10.1162/neco_a_01417

1 Introduction

Neural associative memory is a type of neural network capable of memorizing (learning) a set of patterns and retrieving them from their corresponding noisy or incomplete versions. The term *association* refers to the linkage of two or more pieces of information. Hopfield neural network (Hopfield, 1982) was among the first designed artificial neural network with autoassociative memories able to retrieve information given only some partial clues as well as reconstruct perturbed patterns. Hopfield neural networks have some drawbacks, such as being biologically implausible, due to the fully connected structure, low efficiency, and spurious memories (see, e.g., Hoffmann, 2019). To improve Hopfield networks, many variants have been proposed (see, e.g. Maurer, Hersch, & Billard, 2005; Berrou & Gripon, 2010; Krotov & Hopfield, 2016; Kim, Park, & Kahng, 2017). Due to the sparse coding in the brain (for sparse coding, see Olshausen & Field, 2004; Rinkus, 2010), sparse associative memories are considered more biologically plausible models (Gripon, Heusel, Löwe, & Vermet, 2016; Hoffmann, 2019).

Gripon and Berrou (2011) proposed novel sparse neuro-inspired associative memories that organize neurons into clusters and memorize patterns using the concept of cliques (see Hopfield, 2008, for another clique-based network model of associative memory). This model, also referred to as the GB model or clustered cliques networks (CCNs), is rooted in information theory (Gripon & Berrou, 2012) and bears similarity to the Willshaw-type model (Willshaw, Buneman, & Longuet-Higgins, 1969), where sparse patterns and binary connections are considered. These models have been further developed in the literature (Aliabadi, Berrou, Gripon, & Jiang, 2014; Boguslawski, Gripon, Seguin, & Heitzmann, 2014; Jarollahi, Onizawa, Gripon, & Gross, 2014; Jarollahi, Gripon, Onizawa, & Gross, 2015; Jiang, Marques, Kirsch, & Berrou, 2015; Jiang, Gripon, Berrou, & Rabbat, 2016; Mofrad, Ferdosi, Parker, & Tadayon, 2015; Mofrad, Parker, Ferdosi, & Tadayon, 2016; Mofrad & Parker, 2017; Berrou & Kim-Dufor, 2018) and used in many applications, such as solving feature correspondence problems (Aboudib, Gripon, & Coppin, 2016), devising low-power, content-addressable memory (Jarollahi et al., 2015), oriented edge detection in image (Danilo et al., 2015), image classification with convolutional neural networks (Hacene, Gripon, Farrugia, Arzel, & Jezequel, 2019), and finding all matches of a probe in a database (Hacene, Gripon, Farrugia, Arzel, & Jezequel, 2017), to mention a few. Furthermore, they were implemented on a general-purpose graphical processing unit (GPU) (Yao, Gripon, & Rabbat, 2014), in 65-nm CMOS (Larras, Chollet, Lahuec, Seguin, & Arzel, 2018), and in distributed smart sensor architectures (Larras & Frappé, 2020). Therefore, CCN models can be referred to as an important brain-inspired memory system (Berrou, Dufor, Gripon, & Jiang, 2014) that became a basis for a wide range of research in associative memory models.

Learning and retrieval of temporal sequences in neural networks is a fundamental property of human intelligence and has been studied through different approaches (Brea, Senn, & Pfister, 2011; Hawkins, George, & Niemasik, 2009; Maurer et al., 2005; Jiang et al., 2016). Tournament-based neural network (TNN) (Jiang et al., 2016) is an extension of the clique-based approach to associative memories, which have oriented connections and therefore the ability to store sequential information (see Marques, Hacene, Lassance, & Horrein, 2017, for an implementation on the GPU). The novel structure of TNN is not only a sequence storage with high memory efficiency, but also a more compatible model with the neuronal signal propagation in the brain via oriented connections (see Hawkins et al., 2009; Hawkins & Ahmad, 2016, for biologically plausible memory sequence structures).

In this letter, we improve the TNN architecture by proposing a more general structure, named Feedback TNN, as well as more accurate retrieval algorithms. The original TNN can be considered a special case of Feedback TNN, with zero feedback connections. For retrieval, fewer random selections during retrieval result in lower component and sequence error at the end. The Cache-Winner retrieval revisits and changes some previous randomly selected components in case an error is detected during retrieval. Explore-Winner reduces the randomness in decisions by considering the consequences of each decision. The idea behind the Cache-Winner technique can be illustrated in simple terms by an analogy with human decision making. Imagine a person who makes a decision fast and then, if he realizes a mistake, tries to resolve it by manipulating past decisions. An analogy for Explore-Winner is a rather careful decision maker who investigates the consequences of all possible decisions at the time and then makes the best decision. In terms of achieving accurate sequence retrieval, both proposed retrieval techniques are superior to the Winner, which literally makes a random decision in the case of equal chance situations and continues without further actions even when realizing a mistake later.

It is also known that the brain is able to follow previously stored sequences, from any given point forward, and somewhat backward (Hawkins & Blakeslee, 2007). The other contribution of this letter is introducing the Feedback-Backward retrieval method, which makes our model more biologically plausible. Using Feedback-Backward retrieval, the model gains the ability to retrieve the entire sequence, given a sub-sequence, no matter its location. The Feedback-Backward retrieval is more compatible with the Feedback TNN, but works well with the original TNN, as we show in the results. Backward retrieval, therefore, adds more capabilities to these types of sequence storage structures and makes them more similar to brain functioning.

The letter is organized as follows. In section 2, we briefly survey the CCN and TNN structures. In section 3, different learning and retrieval algorithms are explained. The simulation results are provided in section 4, and in section 5, discussion and concluding remarks are presented.

2 Background

First, we describe the clustered clique-based neural network structure in section 2.1. These types of networks are able to store and retrieve the fixed length patterns. Next, in section 2.2, we survey tournament-based neural networks, which have the ability to store and retrieve sequences.

2.1 Clustered Clique Networks (CCNs). In CCNs the way the neurons are organized within clusters, and the sparsity of the encoding used for storing patterns in cliques result in large storage diversity (i.e., number of storable patterns), high capacity (i.e., the amount of storable information), and strong robustness against erasures and errors (Gripon & Berrou, 2011; Jarollahi et al., 2015; Gripon et al., 2016).

Formally, the structure of CCNs consists of n neurons divided into c clusters with the possibility of different sizes. The input patterns are formed from a predefined alphabet $\mathcal{A}$ where the number of neurons in each cluster matches the size of used alphabet $|\mathcal{A}|$. For simplicity, all clusters are considered to have the same number of neurons, say, $l = n/c$, and therefore the same alphabet size $|\mathcal{A}| = l$. The jth neuron in the ith cluster is denoted by n_{ij} and it has an associated value, $v(n_{ij})$, equals one if it is activated and zero otherwise, where $1 \leq i \leq c$ and $1 \leq j \leq l$. Let $\mathcal{P}$ be the set of patterns to be stored, where pattern $p \in \mathcal{P}$ contains c subpatterns: $p = p_1 p_2 \cdots p_c$; for $p_i \in \mathcal{A}$.

The learning process starts by assigning a unique set of neurons—one per cluster—to each $p \in \mathcal{P}$:

$$p = p_1 p_2 \cdots p_c \rightarrow (f(p_1), f(p_2), \ldots, f(p_c))$$
$$\text{where} \quad f : \{p_i\} \rightarrow \{n_{ij} | 1 \leq j \leq l\}.$$

Learning proceeds by activation of the selected neurons, $v(n_{ij}) = 1$, and forming a clique by connecting the selected c active neurons to each other through binary edges. As a result, the learning process generates a set of binary edges,

$$\mathcal{W} = \{\omega_{(ij)(i'j')} | \text{ if } i \neq i' \text{ and } \exists\, p \in \mathcal{P} \text{ s.t. } f(p_i) = n_{ij} \text{ and } f(p_{i'}) = n_{i'j'}\},$$

where $\omega_{(ij)(i'j')}$ is an edge between n_{ij} and $n_{i'j'}$. The edge $\omega_{(ij)(i'j')}$ belongs to $\mathcal{W}$ independent from the number of patterns that use both n_{ij} and $n_{i'j'}$ neurons but only if there exists such a pattern. Figure 1 illustrates the storing process in clique-based networks.

The recall or retrieval phase of a possibly distorted version of a learned pattern, $\hat{p}$, is based on finding the closest match from $\mathcal{P}$. Depending on the type of distortion, various retrieval methods might be used (see Aboudib, Gripon, & Jiang, 2014); however, in general, the recall procedure consists of

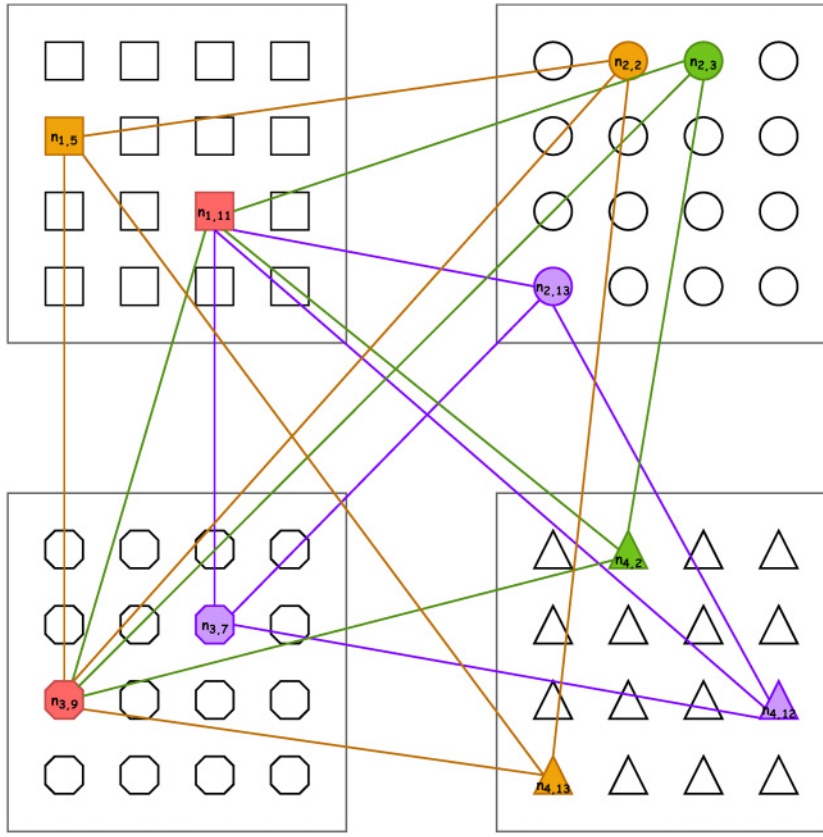


Figure 1: The learning process of three patterns in a network with $c = 4$ clusters and $l = 16$ neurons per cluster. Node $n_{i,j}$ refers to the jth neuron in the ith cluster. Each clique represents one of the three (4, 1, 8, 12), (10, 2, 8, 1), and (10, 12, 6, 11) patterns with yellow, green, and purple, respectively. Colored nodes refer to the activation of neurons for at least one pattern. The red nodes, $n_{1,11}$ and $n_{3,9}$, belong to two patterns. Note that it is not possible to retrieve the patterns by finding a unique clique using only one of these red nodes.

local and global phases. The local phase aims to find the most probable neurons in different clusters, using information from $\hat{p}$ or incoming connections from previously activated neurons, and activate them, that is, $v(n_{ij}) = 1$. The global phase is to recall the established edges in $\mathcal{W}$ that have an end in activated neurons. This procedure alternates between global and local retrieval to gradually complete the clique and therefore the pattern.

It is noteworthy that other sparse structures were presented by Aliabadi et al. (2014), according to which $c \ll \chi$, where $\chi = n/l$ denotes the number

of clusters and c was used to denote a smaller set of clusters for which a sparse pattern is mapped into. Retrieval in this case would be more complicated, and various scenarios could be considered (see, e.g., Aboudib et al., 2014; Jiang, 2014). For instance, the winner-take-all rule activates neurons with the highest activity (or maximum score), while losers-kicked-out rule eliminates active neurons with less activity using a threshold filter (see Jiang, 2014, for details).

2.2 Tournament-Based Neural Network (TNN). An extension of the CCNs (Jiang et al., 2016) is proposed by using directed edges between clusters in such a way that the network can store sequential information in a tournament-based[1] neural network. In a chain of tournaments of order c and degree r, denoted by $\mathcal{T}_r(c)$, each node is directed clockwise to its r consecutive neighbors; see Figure 2 with $c = 8$ and $r = 3$ for a sample chain of tournaments. A TNN can then be seen as a concatenation of tournaments of size $r + 1$.

In order to store a set of sequences, $\mathcal{S}$, in a chain of tournaments, we suppose that each sequence $s \in \mathcal{S}$ contains L component, that is, $s = s_1 s_2 \cdots s_L$; for $s_t \in \mathcal{A}, t = 1, 2, \ldots, L$, and $|\mathcal{A}| = l$.

By labeling clusters from 1 to c, the learning process could be explained as follows. First, a unique sequence of neurons must be assigned to each $s \in \mathcal{S}$ by using function $f = (f_1, \cdots, f_c)$, where $f_i, i = (t - 1 \bmod c) + 1$, maps a component s_t, to a unique neuron n_{ij} in cluster i:

$$f_i : \{s_t\} \to \{n_{ij} | 1 \le j \le l, \}, \ 1 \le i \le c;$$

therefore,

$$f(s) = (f_1(s_1), f_2(s_2), \ldots, f_c(s_c), \ldots, f_{(L-1 \bmod c)+1}(s_L)).$$

Learning continues by connecting neuron n_{ij} to neuron $n_{i'j'}$ at passage π as follows,

$$n_{ij} \to n_{i'j'}, \text{ if: } \begin{cases} f_i(s_{(i+(\pi-1)c)}) = n_{ij} \\ f_{i'}(s_{i'+(\pi-1)c}) = n_{i'j'} \end{cases} \text{ and, } 1 \le \delta_i(i') \le r, \tag{2.1}$$

where $\delta_i(i') = (i' - i) \bmod c$, and $1 \le \pi \le \lfloor \frac{L}{c} \rfloor$.

In general, for $s \in \mathcal{S}$, if the above conditions are satisfied for a given π such that $n_{ij} \to n_{i'j'}$, we set $N_{s,\pi}(n_{ij}, n_{i'j'}) = 1$, which means that n_{ij} is connected to $n_{i'j'}$, in sequence s; otherwise, we set $N_{s,\pi}(n_{ij}, n_{i'j'}) = 0$. In Figure 2,

[1] In graph theory, by assigning direction to all edges of a complete graph, a tournament can be achieved.

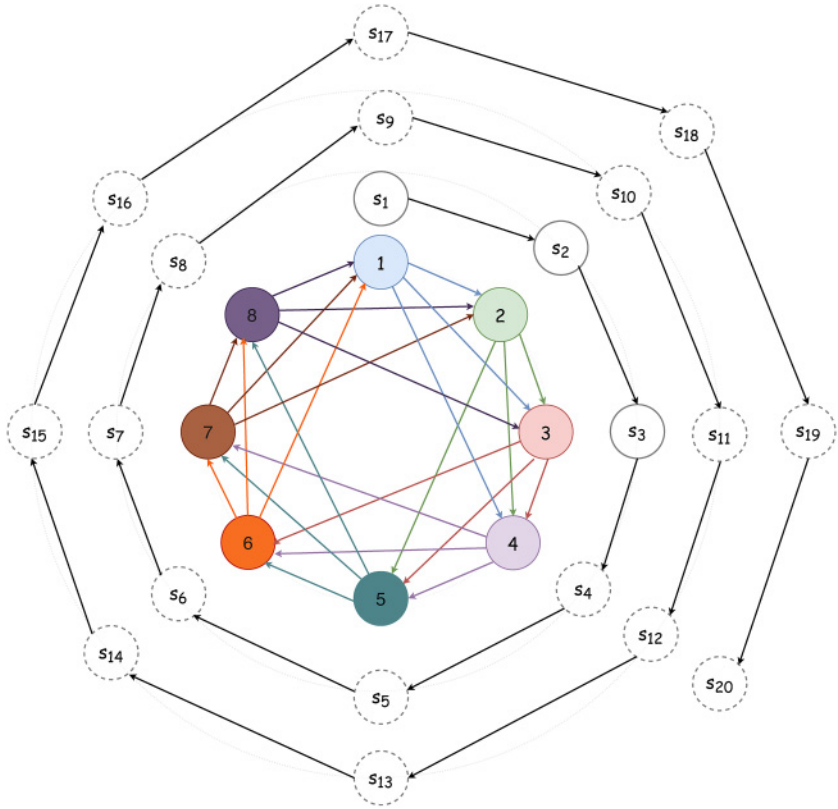


Figure 2: An illustration of a chain of tournaments, $\mathcal{T}_3(8)$, for storing sequences of length 20. The eight clusters are represented by colored circles, and each arrow represents a set of possible connections between nodes within the clusters. The clusters construct eight tournaments of size $r + 1 = 4$. For instance, clusters that have been shown with 1, 2, 3, 4 make one tournament starting from cluster 1, and clusters labeled with 7, 8, 1, 2 are involved in another tournament starting from cluster 7. A sequence of length 20 and the assigned clusters for each component s_i is represented around the network. Given the first r components (s_1, s_2, s_3) with solid circles, the retrieval algorithm could retrieve the rest sequentially using the tournament connections. This figure is based on Jiang et al. (2016, Figure 5).

s_2 is connected to s_3 in passage $\pi = 1$, but not to s_{11} (in passage $\pi = 2$), and s_{19} (in passage $\pi = 3$) in the same sequence s, for instance. So the neighboring connections are defined based on both s and π values.

At the end of the learning or storing process, the network has the following connections:

$$\mathcal{W} = \left\{ \omega_{(ij)(i'j')} | \text{ if } \exists s \in \mathcal{S}, \text{ and } \exists \pi \in \left[1 : \left\lfloor \frac{L}{c} \right\rfloor \right] \text{ s.t. } N_{s,\pi}(n_{ij}, n_{i'j'}) = 1 \right\}, \tag{2.2}$$

where $\omega_{(ij)(i'j')}$ is a directed edge from n_{ij} to $n_{i'j'}$ and $1 \leq i, i' \leq c,\ 1 \leq j, j' \leq l$ (see algorithm 1 for the learning process).

A stored sequence retrieval process could start with any sub-sequence of r consecutive components, and the activation of a component in the following cluster relies on the connections of r previous clusters. If the given sub-sequence is not the first r components of the sequence, the retrieval algorithm requires the information of the location of clusters. In Figure 2, the first three components s_1, s_2, and s_3 are shown with solid circles, and the components to be retrieved are shown with dashed circles.

The proposed retrieval procedure is sequential using a winner-takes-all decision at each step. For brevity, we call this retrieval *Winner* in the rest of letter (see algorithm 2).

3 Structures and Algorithms

The original learning and retrieval algorithms for TNN that were proposed by Jiang et al. (2016) are reported in section 3.1. In sections 3.1.1 and 3.1.2, the newly proposed retrieval algorithms Winner-Cache and Winner-Explore are provided, respectively. Feedback TNN structure along with its corresponding learning and retrieval algorithms, Feedback-Forward and Feedback-Backward, are presented in section 3.2. Finally, the error types that are used for evaluation of structures are addressed in section 3.3.

3.1 Learning and Retrieval Algorithms in TNN. TNN structure, which is explained in section 2.2, is summarized by algorithms 1 and 2 for the learning and retrieval phases, respectively.

For retrieval, the first r components of a previously learned sequence, $[s_1 : s_r]$ and the learned graph, G, are given, and the complete sequence starting with $[s_1 : s_r]$ is expected.

In algorithm 2, each of the given r components is mapped to its related neurons in the first r clusters. Note that each component value is a number from 0 to $l - 1$. Then the retrieval algorithm establishes the output edges from these r active neurons. The neurons in the destination cluster with highest input score will form the candidate set for the next component of the sequence. If there is just one candidate, it will be added to the retrieved sequence and activated for retrieving the next component. Otherwise, the component must be chosen randomly among the candidates.

3.1.1 Winner-Cache Retrieval in TNN. In the case of Winner-Cache algorithm, the learning phase is similar, but the retrieval is more advanced. As

Algorithm 1: Learning in TNN.

input : $c,\ k,\ r,\ L$ & $\mathcal{S}$
initialization
$l = 2^k$,
Generate directed graph G with $n = c \times l$ nodes structured in c clusters of size l.
Assign clusters indices from 1 to L cyclically (similar to Figure 2)
begin
  for $s \in \mathcal{S}$ **do**
    Activate the corresponding neurons to the sequence components;
    Connect each active neuron to the consecutive r active neurons.
output: G

Algorithm 2: Winner Retrieval in TNN.

input : G & $[s_1 : s_r]$
initialization
Activate r neurons in the first r clusters using $[s_1 : s_r]$
begin
  for $i \in [r+1 : L]$ **do**
    Establish the output edges from previous r active neurons in the sequence;
    Create a candidate set of nodes with maximum score in cluster i.
    if *len(candidate set)* $== 1$ **then**
      activate the only candidate node as winner and record it as s_i
    else if *len(candidate set)* > 1 **then**
      activate one of the candidate nodes randomly as winner and record it as s_i;
output: $s_{[s_1:s_r]}$ `// Retrieved sequence given` $[s_1 : s_r]$

reported in algorithm 3, a temporary cache memory is used in the cases where random selection among winners results into an error which is detected later (see Figure 3 for an illustration).

The Cache-Winner algorithm proceeds as follows. Whenever there is no unique candidate, the component is chosen randomly among the candidates, and other candidates will be recorded temporarily (up to assignment of the next *r* components). If the algorithm cannot find a candidate connected to all the previous *r* active neurons, the algorithm starts retrieval from the earliest nonempty cache memory by randomly choosing another member. For brevity, we refer to this retrieval as *Cache* in the rest of the letter.

3.1.2 Winner-Explore Retrieval for TNN. At this juncture, we introduce a retrieval technique that performs exploration within the forthcoming clusters to find a more accurate solution. As reported in algorithm 4, whenever

Algorithm 3: Winner-Cache Retrieval in TNN.

input : G & $[s_1 : s_r]$
initialization
Activate r neurons in the first r clusters using $[s_1 : s_r]$
begin
  $i = r$
  while $i < L$ **do**
    $i+ = 1$
    Establish the output edges from last r active neurons in the sequence;
    Create a candidate set of nodes with maximum score in cluster i.
    if *maximum score* $< r$ **then**
      search in the cache data of last r neurons ($[i - r : i - 1]$), find the first non-empty cache (j) and select a new member randomly. Update the cache by removing the new member and start retrieval from that point (j) again by putting $i = j$.
    if *len(candidate set)* $== 1$ **then**
      activate the only candidate node as winner and record it as s_i
    else if *len(candidate set)* > 1 **then**
      activate one of the candidate nodes randomly as winner and record it as s_i;
      Put the remaining members of the candidate set into a temporary cache;
      keep the cached data until the next r neurons are assigned.
output: $s_{[s_1:s_r]}$ // Retrieved sequence given $[s_1 : s_r]$

the candidate set in a cluster is not unique, by using the previous activated neurons, we produce possible candidates in the next clusters and consequently try to eliminate the current candidates by exploring the connections to the generated candidate sets (see Figure 4). The maximum number of clusters that can be investigated ($r_{explore}$) is upper-bounded by $r - 1$. However, as in section 4.1.1, one could limit the retrieval algorithm to explore shorter distances—for instance, setting $r_{explore} < c - r$ in order to reach each cluster at most once for a specific component. Exploration involves searching for candidate sets in the following clusters and then trying to eliminate the number of candidates in the current cluster. The two techniques for this part are called forward technique and clique technique. In forward technique, any candidate that is not connected to at least one node in the following clusters will be deleted from the candidate set. Therefore, it is possible to find a unique candidate by reducing the size of the candidate set. The clique technique is more advanced since it removes the candidates that are not in a tournament of largest possible size. We use the term *clique* for this technique to differentiate it from the learning on chain of tournaments.

The retrieval process, as reported in algorithm 4, searches for a candidate set in one cluster at each iteration: first by using the forward technique and then applying the clique technique. In the case of a nonunique option, the

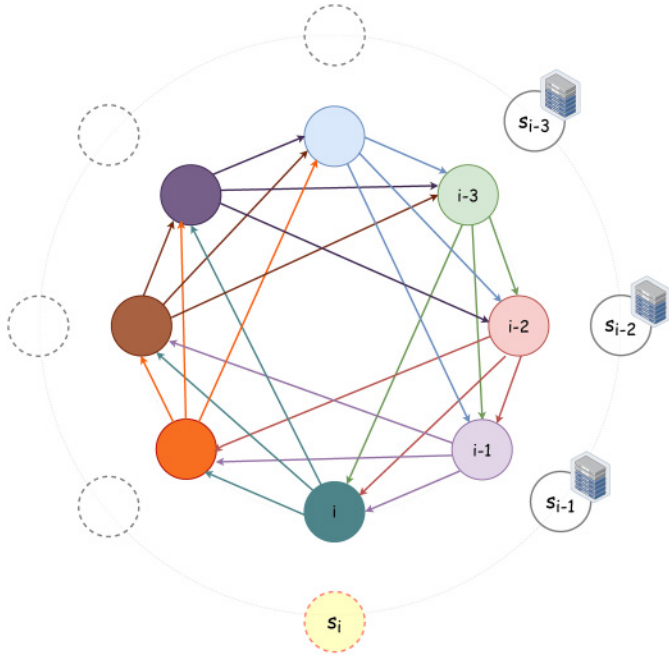


Figure 3: The mechanism of using temporary cache memory in the Winner-Cache retrieval. The component s_i, in yellow, represents the point in the retrieval where none of the nodes in cluster i has a score equal to $r = 3$ from the three previously activated neurons. This means that s_{i-1}, s_{i-2}, and s_{i-3} do not belong to any of the stored sequences. Starting from cache memory in cluster $i - 3$ for component s_{i-3}, if there is an alternative candidate to be activated, we change the component and start retrieving the sequence from that point. If in s_{i-3} the cache memory is empty, the algorithm checks for s_{i-2} and then s_{i-1}. At the end, if there is no alternative, or using the alternatives does not help, the candidate set for component s_i will be one of the winners—a node with maximum score.

algorithm proceeds by adding a new candidate set in the following cluster, and so on. The search for a unique candidate stops whenever a unique option is found or all the clusters for exploration are taken into computation.

3.2 Feedback TNN Structure. In this structure, the learning phase sets tournaments with forward and backward connections. Each node in a tournament of size $r+1$ has r_{fwd} links to the forthcoming clusters and receives r_{fbk} links from the forthcoming $[r_{fwd} + 1 : r]$ active neurons, where $0 \leq r_{fbk} \leq r_{fwd}$ and $r = r_{fwd} + r_{fbk}$ (see Figure 5). The original TNN can be seen as a feedback TNN with zero feedback links ($r_{fbk} = 0$).

Algorithm 4: Winner-Explore Retrieval in TNN.

input : G & $[s_1 : s_r], r_{explore}$
initialization
Activate r neurons in the first r clusters using $[s_1 : s_r]$
begin
  for $i \in [r+1 : L]$ **do**
    Establish the output edges from last r active neurons in the sequence;
    Create a candidate set of nodes with maximum score in cluster i.
    if *len(candidate set)* $== 1$ **then**
      activate the only candidate node as winner and record it as s_i
    else if *len(candidate set)* > 1 **then**
      for $j \in [1 : r_{explore}]$ **do**
        Create a candidate set in cluster $i + j$ using the $r - j$ activated nodes prior to i;
        Construct a sub-graph of G with nodes of candidate sets in cluster i up to cluster $i + j$;
        Update the candidate set in cluster i by keeping nodes with maximum output edges in sub-graph
        if *len(candidate set)* $== 1$ **then**
          activate the only candidate node as winner and record it as s_i. `// Forward technique worked.`
        else if *len(candidate set)* > 1 **then**
          Find all tournaments in the sub-graph including nodes from candidate set in cluster i with size $j + 1$;
          Update the candidate set in cluster i so that only candidates in such tournaments remain;
        if *len(candidate set)* $== 1$ **then**
          activate the only candidate node as winner and record it as s_i. `// Clique technique worked.`
        else if *len(candidate set)* $== 0$ ***or*** $j == r - 1$ **then**
          Return the last non-empty candidate set as the final candidate set for cluster i;
output: $s_{[s_1:s_r]}$ `// Retrieved sequence given` $[s_1 : s_r]$

For storing sequence $s \in \mathcal{S}$, where $s = s_1 s_2 \cdots s_L$, the clockwise connections in the network will be

$$n_{ij} \rightarrow n_{i'j'}, \text{ if: } \begin{cases} f_i(s_{i+(\pi-1)c}) = n_{ij} \\ f_{i'}(s_{i'+(\pi-1)c}) = n_{i'j'} \end{cases} \text{ and, } 1 \leq \delta_i(i') \leq r_{fwd}, \tag{3.1}$$

and for counterclockwise connections,

$$n_{ij} \leftarrow n_{i'j'}, \text{ if: } \begin{cases} f_i(s_{i+(\pi-1)c}) = n_{ij} \\ f_{i'}(s_{i'+(\pi-1)c}) = n_{i'j'} \end{cases} \text{ and, } r_{fwd} \leq \delta_i(i') \leq r, \tag{3.2}$$

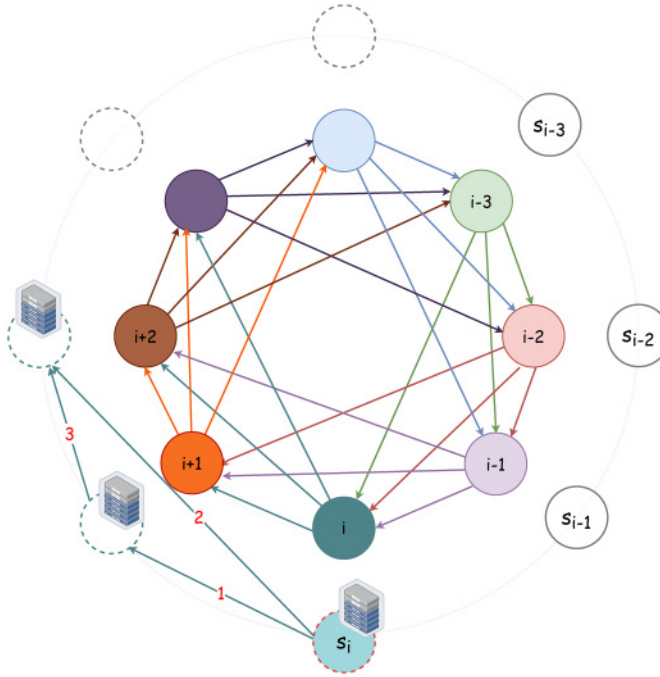


Figure 4: Using the exploration technique to eliminate the number of components that are chosen randomly among the winners in Winner-Explore retrieval algorithm. Suppose that by using the edges from $r = 3$ previous nodes equivalent to s_{i-3}, s_{i-2}, and s_{i-1} to find s_i component, more than one option is found for the candidate set in cluster i. In this case, $r_{explore} = r - 1 = 2$ previous components, that is, s_{i-2} and s_{i-1}, are used to create a candidate set in cluster $i + 1$. In the forward technique, the algorithm checks which candidates for component i are connected to at least one of the nodes in the candidate set in cluster $i + 1$ (using links labeled with 1). If there is still more than one option, a candidate set in cluster $i + 2$ will be constructed using s_{i-1}. Again, using the forward technique, the connections between candidates in cluster i and the candidate sets in the following $i + 1$ and $i + 2$ clusters are used to eliminate the options (labeled with 1 and 2). If still no unique option is available, the clique technique will be used, which searches for the possible cliques of size 3 (using all the links labeled with 1, 2, and 3). Since $r_{explore} = 2$, if there is no unique candidate in cluster i within the cliques, the process stops and the winner will be chosen randomly.

where $1 \leq \pi \leq \lfloor \frac{L}{c} \rfloor$. In general, for $s \in \mathcal{S}$, if the above conditions are satisfied for a given π such that $n_{ij} \rightarrow n_{i'j'}$, we set $N_{s,\pi}(n_{ij}, n_{i'j'}) = 1$. Similarly we set $N_{s,\pi}(n_{i'j'}, n_{ij}) = 1$ if $n_{ij} \leftarrow n_{i'j'}$; otherwise we set $N_{s,\pi}(n_{ij}, n_{i'j'}) = 0$ and $N_{s,\pi}(n_{i'j'}, n_{ij}) = 0$.

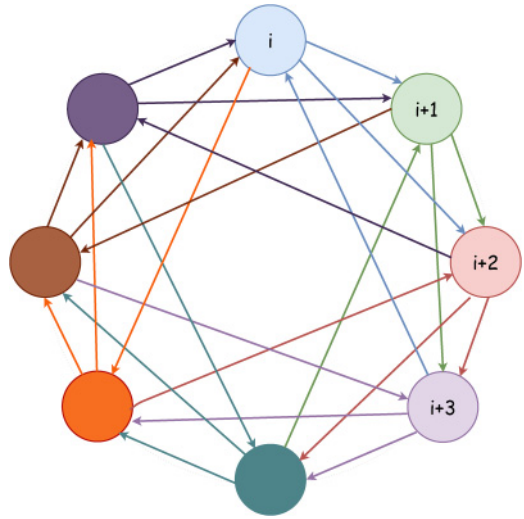


Figure 5: In the chain of tournament structure with feedback connections, the first r_{fwd} connections of each tournament are clockwise, and the next r_{fbk} connections are counterclockwise. In this illustration, $r_{fwd} = 2$ and $r_{fbk} = 1$.

At the end of the learning or storing process, the network has the following connections,

$$\mathcal{W} = \left\{ \omega_{(ij)(i'j')} | \text{ if } \exists s \in \mathcal{S}, \text{ and } \exists \pi \in \left[1 : \left\lfloor \frac{L}{c} \right\rfloor \right] \text{ s.t. } N_{s,\pi}(n_{ij}, n_{i'j'}) = 1 \right\}, \tag{3.3}$$

where $\omega_{(ij)(i'j')}$ is a directed edge from n_{ij} to $n_{i'j'}$ and $1 \leq i, i' \leq c$, $1 \leq j, j' \leq l$ (see algorithm 5 for the learning process). In Figure 5, activated neurons in cluster i are connected to the activated neurons in clusters $i+1$ and $i+2$ clockwise, whereas activated neurons in cluster $i+3$ are connected to the activated neurons in cluster i counterclockwise.

3.2.1 Retrieval in Feedback TNN. Here we introduce two retrieval algorithms, Feedback-Forward (algorithm 6) and Feedback-Backward (algorithm 7), which can retrieve a complete sequence from any given segment. To do so, we need a prematching process to find the clusters on which the given sequence segment was stored (see Figure 6 for an illustration of Feedback-Forward and Feedback-Backward processes).

The Feedback-Forward algorithm (hereafter Forward) retrieves the sequence given the first r components of it. This retrieval is performed in two phases: first, by using the r_{fwd} connections, and then if the winning

Algorithm 5: Learning in Feedback TNN.

input : $c,\ k,\ r,\ r_{fwd},\ L$ & $\mathcal{S}$
initialization
$l = 2^k,\ r_{fbk} = r - r_{fwd}$
Generate directed graph G with $n = c \times l$ nodes structured in c clusters of size l.
Assign clusters indices from 1 to L cyclically (see Figure 5 for labeling)
begin
  for $s \in \mathcal{S}$ **do**
    Activate the corresponding neurons to the sequence;
    Connect each active neuron (say in cluster i) to the active neurons in the next r_{fwd} clusters ($[i+1 : i+r_{fwd}]$);
    Connect each active neuron to the previous r_{fbk} active neurons in clusters $[i-r : i-r_{fwd}-1]$;
output: G

Algorithm 6: Feedback-Forward Retrieval in Feedback TNN.

input : G & $[s_1 : s_r]$
initialization
Activate r neurons in the first r clusters using $[s_1 : s_r]$
Assign clusters indices from 1 to L cyclically
begin
  for $i \in [r+1 : L]$ **do**
    Establish the output edges from previous r_{fwd} active neurons in the sequence;
    Create a candidate set of nodes with maximum score in cluster i.
    if *len(candidate set)* $== 1$ **then**
      activate the only candidate node as winner and record it as s_i
    else if *len(candidate set)* > 1 **then**
      A sub-graph of G with nodes from candidate set in cluster i, and previous r_{fbk} active neurons in clusters $[i-r : i-r_{fwd}]$ is constructed;
      The new candidate set for cluster i is updated by keeping the nodes which have maximum output edges in the sub-graph;
      Select one node from the updated candidate set as winner and record it as s_i;
output: $s_{[s_1:s_r]}$ `// Retrieved sequence given` $[s_1 : s_r]$

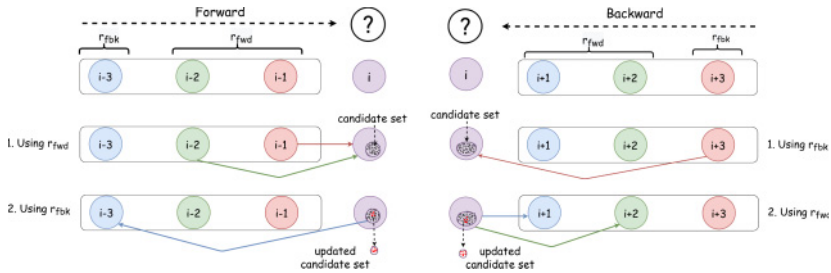
candidate is not unique, the r_{fbk} connections are used to eliminate the number of candidates, as reported in algorithm 6.

The Feedback-Backward algorithm (hereafter Backward), retrieves the sequence given the last r components of a sequence. As reported in algorithm 7, the algorithm first uses the r_{fbk} input edges to make an initial candidate set, and then the output edges from the candidate set are used to eliminate the number of candidates.

Algorithm 7: Feedback-Backward retrieval in Feedback TNN.

input : G & $[s_{L-r+1} : s_L]$
initialization
Activate r neurons in the related r clusters using $[s_{L-r+1} : s_L]$
Assign clusters indices from 1 to L cyclically
begin
  for $i \in [L - r : 1; -1]$ **do**
    Establish the output edges from r_{fdk} active neurons in clusters $[i + r_{fwd} : i + r]$;
    Create a candidate set of nodes with maximum score in cluster i.
    if *len(candidate set)* $== 1$ **then**
      activate the only candidate node as winner and record it as s_i
    else if *len(candidate set)* > 1 **then**
      A sub-graph of G with nodes from candidate set in cluster i, and the next r_{fwd} active neurons is constructed;
      The new candidate set for cluster i is updated by keeping the nodes with maximum score (maximum output edges) in the sub-graph;
      Select one node from the updated candidate set as winner and record it as s_i;

output: $s_{[s_{L-r+1}:s_L]}$ // Retrieved sequence given $[s_{L-r+1} : s_L]$



(a) For Forward retrieval in Feedback TNN, first a candidate set in cluster i is created using the connections from active neuron in clusters $i-1$ and $i-2$ (since $r_{fwd} = 2$). If there is a unique winner candidate, the algorithm stops, otherwise a sub-graph is constructed with the candidate set and the active neuron in cluster $i-3$ (since $r_{fbk} = 1$). The candidate set will be updated by keeping nodes with maximum score.

(b) For Backward retrieval in Feedback TNN, first a candidate set in cluster i is created using the connections from active neuron at cluster $i+3$ (since $r_{fbk} = 1$). If there is a unique winner candidate, the algorithm stops, otherwise a sub-graph is constructed with the candidate set and the active neurons in clusters $i+1$ and $i+2$ (since $r_{fwd} = 2$). The candidate set will be updated by keeping nodes with maximum score.

Figure 6: Consider the structure in Figure 5 where $r_{fwd} = 2$ and $r_{fbk} = 1$. Given a segment of $r = 3$ components, forward and backward retrieval processes are illustrated respectively in panels a and b.

Note that Winner (algorithm 2) can be seen as a special case of Forward (algorithm 6) when $r_{fwd} = r$ and $r_{fbk} = 0$. In Figure 6a, only the first step that uses r_{fwd} is applicable. On the other hand, in the case of the original TNN, the Backward algorithm starts with a candidate set of size l and makes a subgraph with the given $r_{fwd} = r$ components, since $r_{fbk} = 0$ and there is no input connection. In Figure 6b, only the second step that uses r_{fwd} is applicable.

3.3 Error Types. Based on the argument of Jiang et al. (2016), two different error types could be distinguished: an error type that is due to prior retrieval errors in simulation and an error type that is structural and is caused by an excessive network density. The structural error type could happen even if all the previous r components are given correctly.

Component error rate (CER) and sequence error rate (SER) address the simulation error; CER is defined as the ratio of the number of incorrect components over the number of total retrieved components, whereas SER is defined as the number of sequences that failed to be retrieved correctly over the total number of sequences.

Structural component error rate (S-CER) and structural sequence error rate (S-SER) address the structural error. According to Jiang et al. (2016), the S-CER can be estimated as the error rate at a single retrieval step when the previously provided r components are correct,

$$P_{S-CER} = 1 - (1 - d^r)^{l-1}, \tag{3.4}$$

where d is the network density, which is the ratio of number of established connections during the storage process over all possible connections that the network structure allows. The density is calculated (in Jiang et al., 2016, equation 7) as

$$d = 1 - \left(1 - \frac{1}{l^2}\right)^{\frac{|S|L}{c}}. \tag{3.5}$$

At the sequence level, S-SER is estimated (in Jiang et al., 2016, equation 9) as

$$P_{S-SER} = 1 - (1 - d^r)^{(l-1)(L-r)} \tag{3.6}$$

Note that the density in the Feedback TNN structure is the same as the density of the original TNN structure (see equation 3.5). This is due to the fact that the density is calculated based on the probability of having a connection between two nodes, and in the case of Feedback TNN, just the directions of some connections are changed while their number remains the same.

Moreover, based on the definition of structural errors, equations 3.4 and 3.6 are valid for Cache, Explore, and Feedback TNN retrievals.

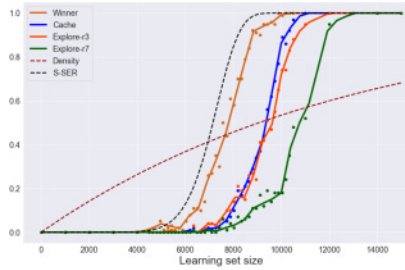
4 Simulation Results

In this section, the simulation results for different algorithms are presented in order to show the robustness of storage and to compare different structures. Learning processes for TNN and Feedback TNN structures (see algorithms 1 and 5, respectively) are considered when $c = 20$, $k = 8$, $l = 2^8 = 256$, $r = 12$, $r_{fwd} = 6$, $r_{fbk} = r - r_{fwd} = 6$, and $L = 100$. Regarding the retrieval, four scenarios—*Winner* (algorithm 2), *Cache* (algorithm 3), *Explore* (algorithm 4), *Forward* (algorithm 6), and *Backward* (algorithm 7)—are simulated and compared.

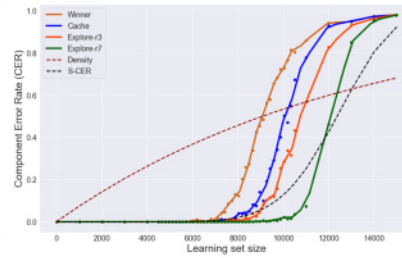
The sequences in the learning set are different in at least one of the first r components. For instance, a learning set of size 1000 is a set of 1000 sequences that all are different in at least one component in the first 12 components. To see if the memorized sequences can be retrieved, 100 of the learned sequences are randomly chosen from each learning set. To reduce the randomness effect, we fixed the 100 choices of sequences in the learning set of each size (varies from 10 to 15,000), in simulations for all the retrieval algorithms.

4.1 TNN Retrieval Results. Figure 7 depicts the error rate for a range of learning set sizes, for different retrieval algorithms: Winner (algorithm 2), Cache (algorithm 3), Explore with $r_{explore} = 3$, and $r_{explore} = 7$ (algorithm 4). To illustrate the power of the algorithms with respect to the structure of the network, the calculated density and structured error are also plotted. It is clear from the results that retrieval with the exploration when $r_{explore} = 7$ is far better than the rest of the scenarios. For instance, when the learning set is composed of 10,000 sequences, each of size 100, the SER (see Figure 7a) for the Winner is one, which means that no sequence can be retrieved correctly with the original algorithm. While this value is about 0.7 for the algorithm with cache memory and about 0.6 when the exploration technique is used with $r_{explore} = 3$, the SER for exploration with $r_{explore} = 7$ is less than 0.2. This superiority of the exploration algorithm can easily be tracked in the CER results (see Figure 7b). For instance, for the same learning set, the CER for Winner is 0.75, for Cache it is 0.4, for Explore with $r_{explore} = 3$, it equals 0.3, and for Explore with $r_{explore} = 7$, it is near zero.

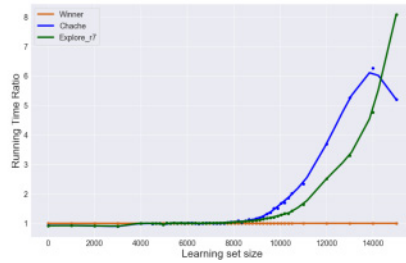
In Figure 7a, the simulated error values for all retrieval methods are less than S-SER, which is obtained from equation 3.6. This can be explained by the fact that the S-SER error estimation is based on the probability of having at least two nodes in a cluster that all the previous r components are connected to. In this case, for the simplest version of retrieval algorithms, Winner, one candidate will be chosen randomly. In other words, S-SER is an



(a) Sequence error rate (SER)



(b) Component error rate (CER)



(c) Running time ratio for Explore-$r7$ and Cache over Winner.

Figure 7: Comparison between retrieval algorithms on the TNN structure; Winner (algorithm 2), Cache (algorithm 3), Explore with $r_{explore} = 3$, and $r_{explore} = 7$ (algorithm 4). The running time ratios of Explore (with $r_{explore} = 7$) and Cache over Winner are reported in 7c.

upper bound for SER, and in the case that all the choices are unique (S-SER $= 0$), there will be no error (SER $= 0$). Although there is no guarantee that the randomly chosen candidate is the desired one, the SER value is slightly less than the S-SER. Obviously the more sophisticated retrieval algorithms, Cache and Explore, reduce the random selections and therefore the number of errors. The structure error is a function of network density, and as can be seen in Figure 7, higher density leads to higher structure error.

For S-CER (see Figure 7b), the argument is different, and the simulated error values in the retrieval process are higher than S-CER. To calculate S-CER, the assumption is that the previously retrieved components are correct, and S-CER estimates the probability of having at least two nodes that are fully connected to the previous r components. However, in the simulation, the values of some r previous components are faulty, and as a result, the decision is not based on correct components. Therefore, in a sequence retrieval, errors at each component could be propagated to the rest of retrieval, and simulated error CER will be higher than S-CER, which assumes the r components are correct.

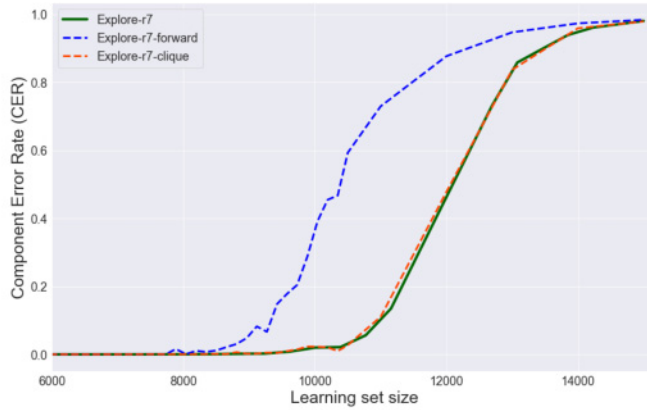
The reported results in Figure 7 suggest an Explore retrieval with a higher number of steps. Cache algorithm is also promising, but for large learning sets, it has a low speed. When the network density increases, the Cache retrieval process creates larger candidate sets for each component and therefore larger cache memory, and the algorithm might go through all the options to find the correct component. Explore must explore longer distances, the source of complexity in Explore. Figure 7c compares the simulation running time between Explore-*r*7 and Cache with Winner for different learning set sizes. The running time up to a learning set size of 8000 for all three algorithms is the same, while Explore-*r*7 and Cache perform far better than Winner. Compare the low performance of Winner (SER = 0.52) with the performance of Cache (SER = 0.08) and Explore-*r*7 (SER = 0.02). As another example, for learning set size 10,000, SER = 1 for Winner, while Explore-*r*7 has SER = 0.18 and running time ratio 1.2, and Cache has SER = 0.86 and running time ratio 1.7.

This shows that for reasonable error values (say, less than 0.1), the running time ratio is at the same level of Winner in both cases. Interestingly, the running time for Cache reaches a peak for a learning set of size 14,000 and thereafter starts to decline for larger learning sets, as shown in Figure 7c. This can be explained by the excessive density, so that the probability of having a full score candidate at each step increases, and therefore the algorithm cannot detect an error, which reduces the processing time for checking the Cache memory.

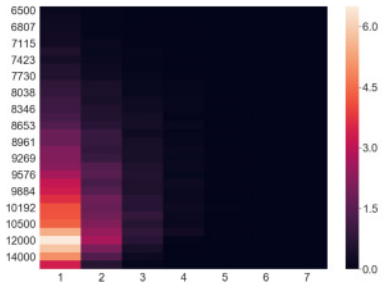
In Figure 7b, only the Explore algorithm with $r_{explore} = 7$ that investigates further clusters shows a lower error than S-CER until the density reaches about 0.6 and a learning set of size 12,000. We will have a closer look at the simulation results for the Explore algorithm.

4.1.1 More Investigation on the Explore Retrieval Algorithm. In Explore retrieval, by starting from distance one, the algorithm uses forward and clique techniques consecutively and increases the exploration distance until a unique candidate is found or the $r_{explore}$ limit is met. The clique technique is more powerful, but it is more computationally expensive than the forward technique. Figure 8a shows that when the clique technique alone (red dashed line) is used, the exploration performance does not change, while the forward technique alone (blue dashed line) is far less effective than the results by exploration algorithm. This is an expected result since the clique technique is endowed with the forward technique.

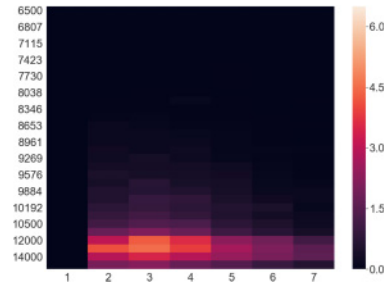
Figures 8b and 8c show the number of components that the forward and clique techniques successfully retrieved (unique winner), respectively, in the course of retrieving each sequence. The columns show the exploration distance, and the rows show the size of learning sets. It is noteworthy that the first column in Figure 8c is all zero, since for a distance one, a forward connection and a tournament of size 2 are the same, and the forward technique is prior to the clique technique in algorithm 6.



(a) CER for Explore algorithm compared with the cases that either Forward technique or Clique technique is used.



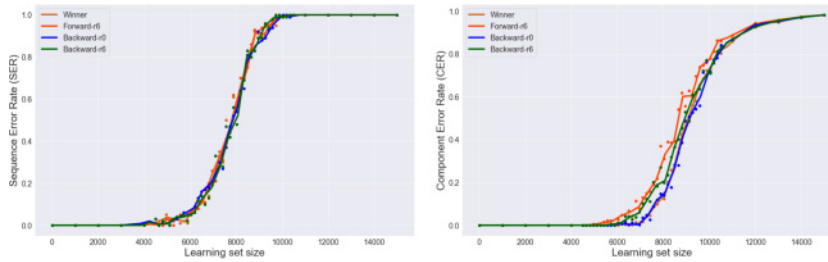
(b) Number of unique winner components which are found at $[1:7]$ exploration distances using Forward technique.



(c) Number of unique winner components which are found at $[1:7]$ exploration distances using Clique technique (with tournament sizes $[1:7]+1$)

Figure 8: Analysis of Explore retrieval. Forward technique versus the clique technique and the required exploration distance for finding a unique component. Results of learning set sizes between 6000 and 15,000 are depicted.

As reported in Figure 7b, the Winner handles the retrieval when the learning set sizes are up to 7000. Until this point, no exploration is demanded. But with larger sizes of the learning set and whenever it comes to the exploration phase, most of the cases can be retrieved with an exploration of distance one. This, however, does not mean that the best choice, in terms of time/accuracy trade-off, is $r_{explore} = 1$. When the sizes of learning sets get higher, the clique technique gets more involved. The higher sizes of candidate sets in exploration clusters increase the searching domain, with the result that the forward technique fails in retrieval and the clique technique starts to retrieve. Let us consider, for instance, the learning set sizes around



(a) Sequence error rate (b) Component error rate

Figure 9: Comparison between the original TNN learning method and the learning in Feedback TNN using Winner, Forward, and Backward retrievals.

12,000 to 13,000, the highest number of successful retrievals per sequence using Explore retrieval (Figure 7a). For these sizes, the CER error is high—for example, it is about 0.46 for learning set of size 12,000 and equals 0.85 when the learning set size is 13,000—and therefore the overall retrieval is not successful. Interestingly, the S-CER also beats CER at around 12,000 (see Figure 7b), which shows that high density cannot be managed with the exploration technique as well.

For learning sets of size 11,000, the CER for Explore-$r7$ is 0.074 (see Figure 7b), while without the exploration technique, the CER value equals one for learning set sizes larger than 10,000. Figures 8b and 8c show decreases in successful cases at exploration with higher distances, say 6 or 7, which suggests that extra exploration is not worth the computation. We found $r_{explore} = 7$ a suitable choice for this setting of parameters.

4.2 Feedback TNN Retrieval Results. Figure 9 shows the retrieval error of Feedback TNN learning when $r = 12$ & $r_{fwd} = 6$ (Forward-$r6$ and Backward-$r6$) together with the retrieval error of the original learning method (TNN) with Winner and Backward-$r0$ retrievals when $r = 12$. We start the Winner and Forward-$r6$ retrievals when the first $r = 12$ components are given and Backward-$r6$ and Backward-$r0$ when the last $r = 12$ components are given.

Figure 9a confirms that the sequence retrieval results in Feedback TNN can be as accurate as the original TNN memories. It is almost the same for CER (see Figure 9b); however, the results for the original TNNs are slightly better. We can explain this as a result of errors in recent previous $r_{fwd} = 6$ components. Consider the case that the algorithm finds a unique candidate for the current component based on last $r_{fwd} = 6$ components without considering the other $r_{fbk} = 6$ links, and selects it as the only winner, although it can be an incorrect candidate due to some errors in previous steps. However, if the algorithm uses all the r_{fwd} and r_{fbk} links, the candidate

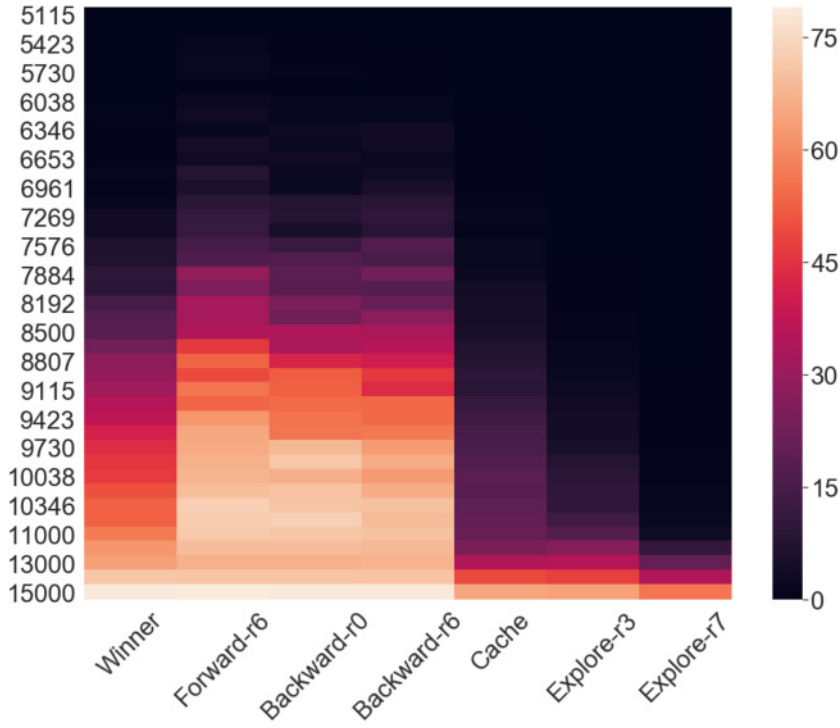


Figure 10: A comparison of number of random selection of winner candidates in different scenarios.

set might be composed of more components, which necessarily do not have the maximum possible score. In this case, the final candidate will be chosen randomly, and therefore there is a chance of correct component selection. This argument could similarly explain why CER for Backward-*r*0 is slightly better than Backward-*r*6. Note that the errors in Feedback TNN retrievals might cause more random choices in retrieving the rest of components (see section 4.3 for an analysis of randomly chosen components). Indeed, such errors do not increase SER, but CER could be affected, as seen in Figure 9b.

In summary, in Feedback TNN, the retrieval is faster than TNN, the SER performance is the same for both, but TNN could be slightly better in CER performance.

4.3 Randomness in Simulated Retrievals An Overall Look. Figure 10 provides a general overview on the number of cases on average that retrieval algorithms select the final component randomly from the candidate set. The success in reducing the number of cases with random decision

in Cache and Explore retrievals to achieve better retrieval performance is clearly shown in the last three columns related to these retrievals. For instance, when the learning set size is 11,000, nearly 50 components out of $L - r = 88$ are chosen randomly for Winner as the original retrieval algorithm, but it is about 20 for Cache, 15 for Explore with $r_{explore} = 3$, and almost zero for Explore with $r_{explore} = 7$. The number of random choices for Feedback TNN structure, both Forward and Backward, is slightly higher than Winner and Backward-$r0$. The argument is that the errors that appear due to the wrong unique retrieval produce more error afterward in the sequence, and therefore more random winner retrieval cases in total. We also can observe slightly more of random winner selections in the Backward-$r0$. This could be related to the learning set generation in our simulations. The sequences in a learning set are forced to be different in at least one of the first r components. Therefore, the Winner can start the retrieval with the unique sequence, while in the Backward-$r0$, more than one sequence can match with the given last r components.

5 Discussion and Conclusion

In this study, two contributions within the field of TNN structures were presented: a more general learning and retrieval structure, called Feedback TNN, and two more accurate retrieval algorithms compared to the Winner algorithm.

In Feedback TNN, each segment of sequence of length $r + 1$ is mapped into a tournament in $r + 1$ consecutive clusters where each neuron has r_{fbk} input edges and $r_{fwd} = r - r_{fbk}$ output edges. The proposed retrieval for the Feedback TNN operates in two phases, in a faster manner than TNN retrieval, and generates the same sequence error rate while producing a slightly weaker component error rate.

The original TNN can be considered a special case of Feedback TNN with zero feedback connections. Using feedback connections, we obtained results of sequence retrieval as precise as the original structure, with the possibility of faster retrieval. One might also divide the r forward connections into two parts, say, r_1 and r_2, and try to retrieve the component using the most recent r_1 active neurons, and if it is not possible to uniquely retrieve, use the rest of the r_2 neurons. More generally, one can try to retrieve by starting from the last active neuron and reduce the size of the candidate set (loser-kicks-out), and adding more active neurons to the retrieval process, until either one winner candidate remains or all the r active neurons are used.

By introducing Backward retrieval, we showed that it is possible to get a part of a sequence, no matter its location, and retrieve the rest. In this case, the retrieval algorithm must be able to first locate a tournament matching the given sub-sequence and later retrieve the whole sequence from both directions. Backward retrieval is compatible with both TNN and Feedback

TNN structures, but Feedback TNN with nonzero feedback links is preferable since the Backward retrieval algorithm can start with a smaller size candidate set.

In order to improve the retrieval accuracy for a given network, we suggested two algorithms with the overall strategy of limiting the number of random selections during retrieval. The Cache retrieval (see algorithm 3) uses a temporary cache memory for the last r components to record the candidate set of winners whenever the chosen winner is not unique. These cached alternatives are used whenever the algorithm detects an error by observing no candidate having a full score. The reported results in section 4 confirm the usefulness of this method. The more advanced, and successful, retrieval algorithm, algorithm 4, explores the forthcoming clusters to find a unique candidate in the current cluster. This algorithm somehow investigates the consequence of choosing each candidate by checking its connections to the possible future components and decides more judiciously. This algorithm produces the best results.

Explore-Winner is a more reliable retrieval method than Cache-Winner since it limits the number of random choices using the data in the forthcoming clusters, while Cache-Winner tries to correct the errors by testing other possibilities. Cache-Winner might be computationally expensive in higher densities where candidate sets of winners are larger and therefore larger sets are cached. Finding an optimal $r_{explore}$, for exploration distance limit, as shown in section 4.1.1, is a trade-off between time and accuracy. Although not reported in the simulations, both Cache-Winner and Explore-Winner can be used in Feedback TNN and for Backward retrieval.

Similar to the double-layer structure proposed by Jiang et al. (2016), it is possible to consider a hierarchical structure by adding an extra connectivity level. Moreover, similar to the technique used in Mofrad et al. (2016) a precoding could dramatically increase the storage and retrieval capacity by forcing patterns to be well separated and therefore reducing the common tournaments in different patterns.

Acknowledgments

We thank the anonymous reviewers for their constructive feedback, which helped to improve the quality of the letter. The source code of the simulations is made publicly available online under this link: https://github.com/Asieh-A-Mofrad/Tournament-Based-Sequence-Storage.

References

Aboudib, A., Gripon, V., & Coppin, G. (2016). A neural network model for solving the feature correspondence problem. In *Proceedings of the International Conference on Artificial Neural Networks* (pp. 439–446). Berlin: Springer.

Aboudib, A., Gripon, V., & Jiang, X. (2014). A study of retrieval algorithms of sparse messages in networks of neural cliques. In *Proceedings of COGNITIVE 2014: The 6th International Conference on Advanced Cognitive Technologies and Applications* (pp. 140–146).

Aliabadi, B. K., Berrou, C., Gripon, V., & Jiang, X. (2014). Storing sparse messages in networks of neural cliques. *IEEE Transactions on Neural Networks and Learning Systems*, *25*(5), 980–989.

Berrou, C., Dufor, O., Gripon, V., & Jiang, X. (2014). Information, noise, coding, modulation: What about the brain? In *Proceedings of the 8th International Symposium on Turbo Codes and Iterative Information Processing* (pp. 167–172). Piscataway, NJ: IEEE.

Berrou, C., & Gripon, V. (2010). Coded Hopfield networks. In *Proceedings of the 6th International Symposium on Turbo Codes and Iterative Information Processing* (pp. 1–5). Piscataway, NJ: IEEE.

Berrou, C., & Kim-Dufor, D.-H. (2018). A connectionist model of reading with error correction properties. In *Proceedings of the 7th Language and Technology Conference on Human Language Technology. Challenges for Computer Science and Linguistics*. Berlin: Springer.

Boguslawski, B., Gripon, V., Seguin, F., & Heitzmann, F. (2014). Huffman coding for storing non-uniformly distributed messages in networks of neural cliques. In *Proceedings of the 28th AAAI Conference on Artificial Intelligence*, vol. 1 (pp. 262–268). Palo Alto, CA: AAAI.

Brea, J., Senn, W., & Pfister, J.-P. (2011). Sequence learning with hidden units in spiking neural networks. In J. Shawe-Taylor, R. Zemel, P. Bartlett, F. Pereira, & K. Q. Weinberger (Eds.), *Advances in neural information processing systems*, *31* (pp. 1422–1430).

Danilo, R., Jarollahi, H., Gripon, V., Coussy, P., Conde-Canencia, L., & Gross, W. J. (2015). Algorithm and implementation of an associative memory for oriented edge detection using improved clustered neural networks. In *Proceedings of the 2015 IEEE International Symposium on Circuits and Systems* (pp. 2501–2504). Piscataway, NJ: IEEE.

Gripon, V., & Berrou, C. (2011). Sparse neural networks with large learning diversity. *IEEE Transactions on Neural Networks*, *22*(7), 1087–1096.

Gripon, V., & Berrou, C. (2012). Nearly-optimal associative memories based on distributed constant weight codes. In *Proceedings of the Information Theory and Applications Workshop* (pp. 269–273). Piscataway, NJ: IEEE.

Gripon, V., Heusel, J., Löwe, M., & Vermet, F. (2016). A comparative study of sparse associative memories. *Journal of Statistical Physics*, *164*(1), 105–129.

Hacene, G. B., Gripon, V., Farrugia, N., Arzel, M., & Jezequel, M. (2017). Finding all matches in a database using binary neural networks. In *Proceedings of COGNITIVE 2017: International Conference on Advanced Cognitive Technologies and Applications* (p. 67).

Hacene, G. B., Gripon, V., Farrugia, N., Arzel, M., & Jezequel, M. (2019). Budget restricted incremental learning with pre-trained convolutional neural networks and binary associative memories. *Journal of Signal Processing Systems*, *91*(9), 1063–1073.

Hawkins, J., & Ahmad, S. (2016). Why neurons have thousands of synapses, a theory of sequence memory in neocortex. *Frontiers in Neural Circuits*, *10*, 23.

Hawkins, J., & Blakeslee, S. (2007). *On intelligence: How a new understanding of the brain will lead to the creation of truly intelligent machines*. New York: Macmillan.

Hawkins, J., George, D., & Niemasik, J. (2009). Sequence memory for prediction, inference and behaviour. *Philosophical Transactions of the Royal Society of London B: Biological Sciences*, *364*(1521), 1203–1209.

Hoffmann, H. (2019). Sparse associative memory. *Neural Computation*, *31*(5), 998–1014.

Hopfield, J. J. (1982). Neural networks and physical systems with emergent collective computational abilities. In *Proceedings of the National Academy of Sciences*, *79*(8), 2554–2558.

Hopfield, J. J. (2008). Searching for memories, sudoku, implicit check bits, and the iterative use of not-always-correct rapid neural computation. *Neural Computation*, *20*(5), 1119–1164.

Jarollahi, H., Gripon, V., Onizawa, N., & Gross, W. J. (2015). Algorithm and architecture for a low-power content-addressable memory based on sparse clustered networks. *IEEE Transactions on Very Large Scale Integration Systems*, *23*(4), 642–653.

Jarollahi, H., Onizawa, N., Gripon, V., & Gross, W. J. (2014). Algorithm and architecture of fully-parallel associative memories based on sparse clustered networks. *Journal of Signal Processing Systems*, *76*(3), 235–247.

Jiang, X. (2014). *Storing sequences in binary neural networks with high efficiency*. PhD diss., Télécom Bretagne, Université de Bretagne Occidentale.

Jiang, X., Gripon, V., Berrou, C., & Rabbat, M. (2016). Storing sequences in binary tournament-based neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, *27*(5), 913–925.

Jiang, X., Marques, M. R. S., Kirsch, P.-J., & Berrou, C. (2015). Improved retrieval for challenging scenarios in clique-based neural networks. In *Proceedings of the International Work Conference on Artificial Neural Networks* (pp. 400–414). Berlin: Springer.

Kim, D.-H., Park, J., & Kahng, B. (2017). Enhanced storage capacity with errors in scale-free Hopfield neural networks: An analytical study. *PLOS One*, *12*(10), e0184683.

Krotov, D., & Hopfield, J. J. (2016). Dense associative memory for pattern recognition. In D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, & R. Garnett (Eds.), *Advances in neural information processing systems, 29* (pp. 1172–1180). Red Hook, NY: Curran.

Larras, B., Chollet, P., Lahuec, C., Seguin, F., & Arzel, M. (2018). A fully flexible circuit implementation of clique-based neural networks in 65-nm CMOS. *IEEE Transactions on Circuits and Systems I: Regular Papers*, *66*(5), 1704–1715.

Larras, B., & Frappé, A. (2020). On the distribution of clique-based neural networks for edge AI. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, *10*, 469–477.

Marques, M. R. S., Hacene, G. B., Lassance, C. E. R. K., & Horrein, P.-H. (2017). Large-scale memory of sequences using binary sparse neural networks on GPU. In *Proceedings of the 2017 International Conference on High Performance Computing and Simulation* (pp. 553–559). Piscataway, NJ: IEEE.

Maurer, A., Hersch, M., & Billard, A. G. (2005). Extended Hopfield network for sequence learning: Application to gesture recognition. In *Proceedings of the International Conference on Artificial Neural Networks* (pp. 493–498). Berlin: Springer.

Mofrad, A. A., Ferdosi, Z., Parker, M. G., & Tadayon, M. H. (2015). Neural network associative memories with local coding. In *Proceedings of the 2015 IEEE 14th Canadian Workshop on Information Theory* (pp. 178–181). Piscataway, NJ: IEEE.

Mofrad, A. A., & Parker, M. G. (2017). Nested-clique network model of neural associative memory. *Neural Computation*, *29*, 1681–1695.

Mofrad, A. A., Parker, M. G., Ferdosi, Z., & Tadayon, M. H. (2016). Clique based neural associative memories with local coding and pre-coding. *Neural Computation*, *28*, 1–21.

Olshausen, B. A., & Field, D. J. (2004). Sparse coding of sensory inputs. *Current Opinion in Neurobiology*, *14*(4), 481–487.

Rinkus, G. J. (2010). A cortical sparse distributed coding model linking mini- and macrocolumn-scale functionality. *Frontiers in Neuroanatomy*, *4*, 17.

Willshaw, D. J., Buneman, O. P., & Longuet-Higgins, H. C. (1969). Non-holographic associative memory. *Nature*, *222*, 960–962.

Yao, Z., Gripon, V., & Rabbat, M. (2014). A GPU-based associative memory using sparse neural networks. In *Proceedings of the 2014 International Conference on High Performance Computing and Simulation* (pp. 688–692). Piscataway, NJ: IEEE.

Received January 2, 2021; accepted April 6, 2021.