3RD WORKSHOP ON MACHINE LEARNING IN NETWORKING (MaLeNe) PROCEEDINGS

SEPTEMBER 1, 2025

CO-LOCATED WITH
THE 6TH INTERNATIONAL CONFERENCE ON NETWORKED SYSTEMS (NETSYS 2025)
ILMENAU, GERMANY

State Cloning with the GraphLearner

1st Timothy Harrison
Lehrgebiet Kommunikationsnetze
University of Hagen
Hagen, Germany
timothy.harrison@fernuni-hagen.de

2nd Herwig Unger
Lehrgebiet Kommunikationsnetze
University of Hagen
Hagen, Germany
0000-0002-8818-3600

***Abstract*—This paper introduces the Cloned GraphLearner, a neuromorphic sequence generation model that mitigates state aliasing in high-order Markov chains through a lightweight, iterative state cloning procedure. Starting from the original GraphLearner, which stores variable length histories in a Graph Structured Bloom Filter, the algorithm successively creates layers of cloned states whose identity and inter-clone edges index increasingly long context windows while an oblivion rule bounds growth. When trained with action-observation sequences the resulting Cloned GraphLearner acts as a topographic schema with individual clones firing in context specific patterns that resemble hippocampal place cell activity.**

***Index Terms*—Bloom Filters, Neuromorphic Computing, Place Cells, Markov Models, Transfer Learning**

I. Introduction

The GraphLearner is a neuromorphic machine learning algorithm for sequence generation inspired by Hawkins and Mountcastle's studies of the neocortex [1] [12]. Its behavior is readily human-explainable and it can be trained in an online manner. The GraphLearner works by estimating high order Markov Chains, using Counting Bloom Filters to dynamically weight first order edges of a sequence graph with higher order sequences [5] [6]. However it suffers from issues of state aliasing when observed states are ambiguous. To alleviate this a process of state cloning is introduced, where nodes of the GraphLearner are cloned based on context provided by the state sequences stored on their edges. The resulting Cloned GraphLearner is capable of learning spacial and network topologies, with individual clones mapping to consistent spacial ranges even in novel environments.

This behavior is notably neuromorphic, with clone behavior resembling that of place cells in the brain. To demonstrate this the 1996 rodent place cell experiments of O'Keefe and Burgess [14] are simulated with the Cloned GraphLearner. These simulations demonstrate the topographic learning capabilities of the Cloned GraphLearner.

II. Related Work

A. Bloom Filters

Bloom Filters are designed to store sets which are too large to store in memory, such as usernames on social media websites [3] [4]. They can test for set occupancy with a guarantee of no false negatives and a controllable false positive rate. A Bloom Filter consists of a bit array of m bits, initially all set to 0, and k unique hash functions, which map to locations on the array [3] [4]. This array can be stored on the harddrive and still accessed in O(1) time. When an element is added to the filter it is passed through the k hash functions mapping it to k locations on the array all of which are set to 1. To test if an element exists in a set this process is reversed with an element existing in the set only if all k of its hash mappings return a 1. It is possible for the k hash mappings of an element to be assigned a 1 from other elements. In such a case a false positive occurs where the element is incorrectly considered part of the set. The chance of a false positive occurring depends on m, k, and the number of elements stored into the filter n.

If the Bloom Filter stores integer values instead of bits and iterates hashmappings by 1 every time an element is added then it becomes a Counting Bloom Filter [5]. Such filters provide an estimated count of each item added to the filter with a risk of false counts similar to the false positive risk.

B. The GraphLearner

The GraphLearner can be characterized as a Graph Structured Counting Bloom Filter where the first order edges of a sequence graph structure the filter. In other words each edge of the sequence graph has its own associated filter. This is depicted in Figure 1. These edge filters store training histories which then provide a dynamic edge weights based on the stored count of a given sequence. This means a high order Markov chain probability can be calculated by searching only the first order edges of the last element of an input sequence. In other words the probability of choosing an edge e for an input sequence S is:

$$P(e) = \frac{B_c^e(S)}{\sum_{e' \in E} B_c^{e'}(S)} \tag{1}$$

Where $B_c^e(S)$ is the count returned by the Bloom Counter associated with edge e and E is the set of all edges of the last element of S.

During training sequences of length h, the maximum history length, are stored in appropriate edge filter. For added flexibility padded or shortened sequences are also stored. Thus the GraphLearner can estimate a Markov Chain distribution for any input sequence of h or shorter, with a sequence of length 1 defaulting to a first order probability. When generating new elements for an input sequence the GraphLearner initially searches with the h most recent elements of the input and

iteratively shortens the search until a non-zero $P(e)$ can be calculated, i.e. when at least one edge matches the search.

If stored with traditional transition matrices these High Order Markov Chains would require $O(n^{h+1})$ worst case space complexity, where n is the number of unique states. This becomes prohibitively large for higher values of h. By contrast the GraphLearner can estimate the same Markov Chains with just $O(n)$ references in RAM.

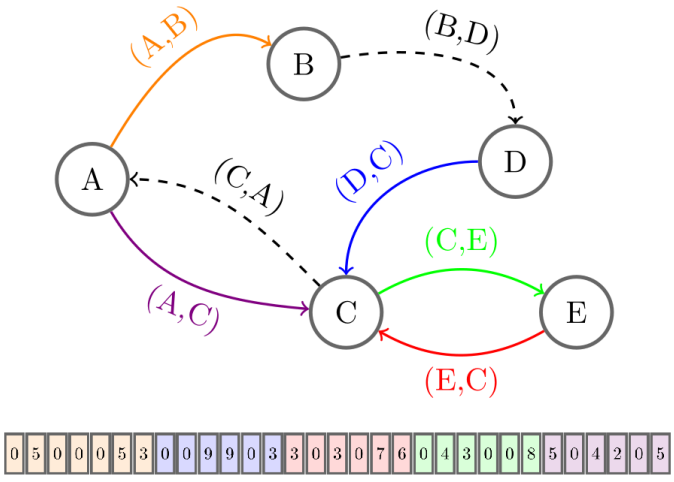

Fig. 1. The original GraphLearner: a Graph Structured Bloom Filter. Edge colors correspond to sections of the structured filter, equivalent to having a filter on each edge. Padded histories of max length h are stored in these filters. The counts returned by the filters for a given h or shorter input then provide dynamic weights which can estimate an h-order Markovian $P(e)$.

C. State Aliasing and Dynamic Markov Modeling

Markov Models like the GraphLearner suffer from problems of state aliasing where a single observable state results from multiple distinct states. For example, in natural language homonyms like ”mouse” or ”bark” have multiple meanings which can only be distinguished with further context.

Dynamic Markov Modeling [11] resolves this issue by creating clones of states from first order information. However the greedy, first order nature of Dynamic Markov Modeling often results in an unsustainable explosion of clones. To resolve this Cloned Hidden Markov Models (CHMM) [18] dynamically merge clones of a given state. In the context of action-state sequence modeling CHMMs are know as Clone Structured Cognitive Graphs (CSCG) [10] due to their similarity to schema networks [17]. The clones of CSCGs are accessed or fired in patterns resembling those of place cells in the brain. However to achieve efficiency CHMMs and CSCGs must fix an upper bound on the number of possible clones of each state.

D. Place Cells and Cognitive Mapping

To perform complex tasks the brain forms cognitive maps [16]. One of the key components of these maps are place cells [13]. Place cells are hippocampal cells which trigger at specific locations in an environment given appropriate observations. As such the firing patterns of place cells encode for spacial topography. Importantly they can fire ahead of time in anticipation of future observations, enabling hippocampal replay. As such, they are important in the planning of actions and complex behaviors.

Place cells specifically occur in the hippocampus but similar structures have been discovered across the brain, including in the neocortex [15].

O’Keefe and Burgess’ expanding box experiment [14] demonstrated that place cell firing patterns corresponded to environmental topographic features, even in novel environments. In this experiment rats were placed in a small square box and given time to learn this environment. The sides of the box were then extended to form three additional environments, a vertical and horizontal rectangle, and a larger square. Place cells which fired at given locations in the small square continued to fire at similar locations in the morphed environments, however in some cases their firing fields were stretched or distorted. However even with these distortions they continue to provide meaningful spacial information, enabling generalization across common environments.

III. The Cloned GraphLearner

The GraphLearner was originally developed for natural language, generating new words and characters from inputs. However it can also be used to control agents when used with action-observation sequences. In this case the GraphLearner is trained on an alternating series of actions and their corresponding observations from a chosen environment. Observations and actions are both treated as states of this sequence. As a result the trained GraphLearner can take an action-observation input ending in an observation and chose an appropriate next action for the agent it controls. Similarly if fed a sequence ending in an action the GraphLearner can predict the next observation, providing a feedback measure of how well it understands its environment.

If trained correctly the GraphLearner forms a simple, transferable schema describing its environment. Unfortunately if that environment has multiple unaliased states the GraphLearner will treat them all identically. In theory a large enough h value can distinguish these states but this method fails when exploring modified environments. Even when those modifications are minor they force the GraphLearner to match shortened sequence histories. In other words the GraphLearner struggles to generalize across common environments because it can no longer rely on precise training histories in these cases. This problem inspired the creation of the Cloned GraphLearner, one layer of which is depicted in Figure 2.

Ideally clones are formed for each distinct state of the training environment, however even when they are not so precise the path taken through successive clones preserves contextual information which might otherwise be lost, such as when the exploration environment is not a precise copy of the training environment.

Creating these clones follows a simple process:

1) Train the GraphLearner as before with a given training sequence T and history length h_0.

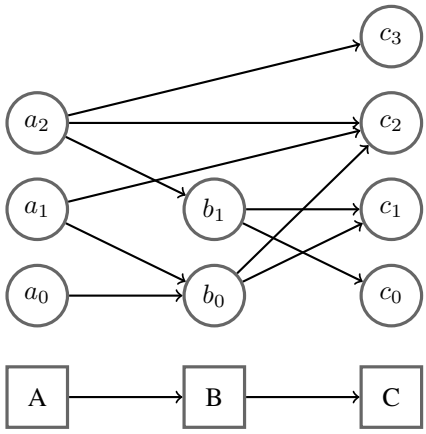

Fig. 2. Part of a Cloned GraphLearner with 1 layer of clones. In (Squares) the original GraphLearner nodes. In (Circles) their clones. Intuitively the paths through these clones can represent more complex historical context than the first order edges of the original nodes. Note: edge (A, C) of the original GraphLearner is not depicted but is implied by the edges connecting clones of A to clones of C. The number of clones implies the existence of other edges.

2) Iterate through T a second time, creating clones with identity edges linking them to original node storing h_0 length sequences.
3) Optionally remove clones which fall under some oblivion threshold.
4) Iterate through T once more connecting clones to clones, creating clone edges which store longer padded histories of max length h_1.
5) Repeat steps 2 through 4 with increasing history lengths, h_n and h_{n+1}, creating new layers of clones from previous layers.

Alternatively it is possible for steps 2 and 4 to be merged, with clones identified and connected to their successors during the same pass of T, followed by oblivion thresholding. However this complicates the oblivion process as all clone to clone edges connecting a deleted clone must also be removed.

Once clones are created and connected the new Cloned GraphLearner generates new sequence elements by following clone to clone edges in the highest possible layer. The cloning and generation processes are described in detail in the following sections.

A. The First Layer of Clones

The following section details the first round of cloning and connecting where clones are created from the nodes of the original GraphLearner and then connected. These layer 1 clones will be indexed with h_0-length sequences, stored on identity edges, and the edges between these clones will store h_1 max length padded histories. Since this process is repeated at each round of cloning Figure 4 refers to h_n and h_{n-1} the history and indexing lengths of the n-th round of cloning.

1) Creating Clones: During cloning each h_0-length subsequence S from the training sequence T is matched to an appropriate clone. For a given S ending in state A the subset of A's edges which match S defines the appropriate clone of A: a_S. This is depicted in Figure 3. An identity edge, with associated Bloom Filter, links A to a_S. Sequence S is stored on this identity edge (A, a_S), along with all other sequences which match a_S's edge subset. Subsequences which match on all edges of A will also have a clone.

In theory T can be a different sequence from the original training sequence so long as it comes from the same underlying distribution, e.g. a separate random walk through the same environment.

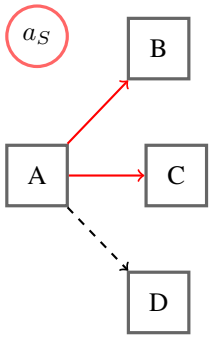

Fig. 3. Clone a_S (Red) is created when a given input sequence only occurs on a subset of A's edges (Red), in this case (A, B) and (A, C). A will have an identity edge from it to a_S which stores all h_0-length sequences matching this edge subset.

2) Connecting Clones: Once clones have been created they must be connected. This process requires two history lengths, h_0 the length of sequences stored on identity edges, and h_1 the maximum length of sequences to be stored on clone to clone edges. The cloning sequence T is traversed again, connected clones are identified from successive h_0-length sequences and h_1-length histories are trained onto their edges. As with the original GraphLearner these h_1 histories are padded so appropriate sequences of length $h_1 - 1$, $h_1 - 2$, ... ,1 are also stored. Figure 4 depicts the clone connecting process for a section of T.

B. Clones of Clones

Once one layer of clones has been created and connected a new layer can be created. This process can be repeated as desired, following the same cloning rules outlined the previous sections, now using subsets of clone to clone edges to define new clones of clones. The edges between these clones of clones then store new longer sequence histories with each successive layer increasing the stored history length. Importantly identity edges will always link from the original, observed states of the GraphLearner to their respective clone.

If clone layer n was accessed with identity edges storing sequences of length h_{n-1} and contained clone to clone edges storing max histories of length h_n then layer $n + 1$ will be indexed with identity edges containing sequences of length h_n and the edges in layer $n+1$ will store max histories of length h_{n+1}, and so on.

The only major difference from the first round of clone creation is that clones of layer $n - 1$ must first be identified

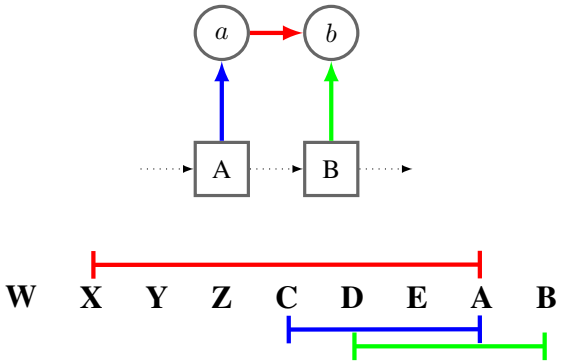

Fig. 4. The connecting process for two layer n clones of A and B: a and b, depicting edges and their corresponding state subsequences for a section of the training sequence. In (Blue) the h_{n-1}-length sequence which matches on the identity edge (A, a) also in (Blue). In (Green) the h_{n-1}-length sequence which matches on the identity edge (B, b) again in (Green). In (Red) the h_n-length sequence which is trained onto edge (a, b) in (Red). Here the clones a and b are identified from their respective subsequences $CDEA$ and $DEAB$ and the longer history $XYZCDEA$ is stored on the edge between them, along with padded or shortened histories $YZCDEA$, $ZCDEA$, etc.

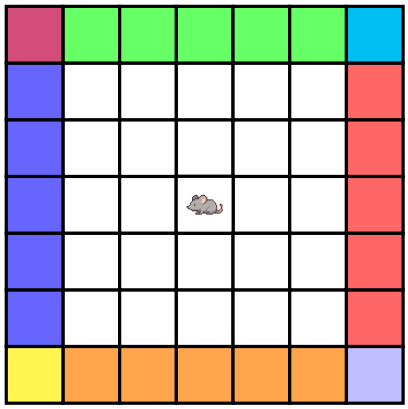

Fig. 5. The 9 observable states of the small square training environment obscure 49 aliased states. This coloration process expedites the formation of meaningful clones by increasing the number of initial edges. The other environments are colored in a similar manner. The cartoon mouse denotes the starting point of the agent. Note that the clones in Figures 6, 7, and 8 are all clones of the white internal state.

by searching the appropriate identity edges before they can be used to create the clones of layer n.

To avoid a potential explosion in the number of clones an appropriate oblivion threshold must be used when creating each new layer.

Once completed the Cloned GraphLearner will have n layers of clones. The clones of each layer are indexed by identity edges containing h_{n-1} length sequences with the edges between clones in that layer storing padded histories of max length h_n. These identity h values are stored to allow faster indexing of clones. In this framework the original GraphLearner effectively forms a zeroth layer, $n = 0$, which is indexed by identity sequences of length 1, i.e. the unique first order states of the training sequence.

C. Oblivion

With each successive round of cloning the number of clones grows. To avoid explosive growth in the number of clones of clones a simple oblivion function is introduced [8]. Once the clone creation process is completed, clones that were indexed less than some threshold value are deleted. This is done before clone connecting occurs to avoid creating edges with deleted clones. The impact of the process can be seen in Figure 9 where a threshold of 100 keeps the number of clones manageable.

D. Generation

At generation time a sequence, S, is input into the Cloned GraphLearner and an appropriate new element is generated for this sequence. If a clone is currently being tracked then the edges of this clone are searched to create a $P(e)$ in the same manner as the original GraphLearner: by recursively searching shortened or padded slices of S until at least one edge match is found. A new clone is selected from this $P(e)$, its associated observation is returned, i.e. the original node-state associated with that clone, and the clone tracker updates to track said clone.

If no clone is currently tracked the last element of S is used to identify a zeroth layer node and the identity edges of this node are searched with h-length slices of the most recent elements of S, starting with the largest possible identity length h_{n-1}. If a matching identity edge is found then its clone is used for generating $P(e)$.

If the input is an action-observation sequence, where new actions are being generated from inputs, then it is possible the most recent observed state, i.e. the last element of S, will not match with the currently tracked clone. In these cases the tracked clone is discarded and a new one must be found.

Given the similarity to place cells, a clone being accessed to generate new sequence elements is referred to as clone firing.

IV. Experiments and Results

A. Experimental Setup

O'Keefe's place cell experiment is replicated with the Cloned GraphLearner. The small square room (SS) is represented by a 7x7 grid, the horizontal rectangle room (HR) with a 14x7 grid, the verticle rectangle room (VR) with 7x14, and the large square room (LS) with a 14x14 grid. All of which are surrounded by impenetrable walls. These rooms are depicted in Figures 6 7 and 8. A simulated rodent agent performs a random walk within the SS environment recording a sequence of actions and observations, T. The GraphLearner is then trained on this sequence and successive rounds of cloning occur, repeatedly using T.

Next the agent performs a new random walk in each of the four environments with the Cloned GraphLearner generating new actions and states from the recorded walk. Each time a clone is accessed during this process the agent's location in the environment is recorded, along with the corresponding clone.

Finally these access locations are used to create a heatmap of where each clone is utilized in the four environments.

The agent's action space is limited to the four cardinal directions. Walls and corners of the environment are treated as distinct states based on the restricted set of actions in each. This results in 9 unique observational states as depicted in Figure 5. The Cloned GraphLearner is trained with initial history length h_0 = 6, and contains 4 layers of clones, not including the zeroth layer nodes of the original GraphLearner. Layer 1 is indexed with a sequences of length h_0 = 6 and stores padded histories of maximum length h_1 = 9. History values continue growing by size 3 until layer 4 which is indexed with h_3 = 15 length sequences and stories padded histories up to length h_4 = 18. An oblivion threshold of 100 is used at each round of cloning. The training sequence, T, is a 150,000 step random walk in the small square environment, see Figure 5.

To measure a process known as place cell remapping the original O'Keefe experiments also considered the directions in which walls were expanded when creating the 3 addition environments. That is not considered in this simulation.

B. Results

The following heatmaps depict the firing locations of selected clones across each of the four environments. In Figure 6 a clone behaves like the place cells from O'Keefe. It has become attached to the left vertical wall and when that wall is extended its firing field also expands.

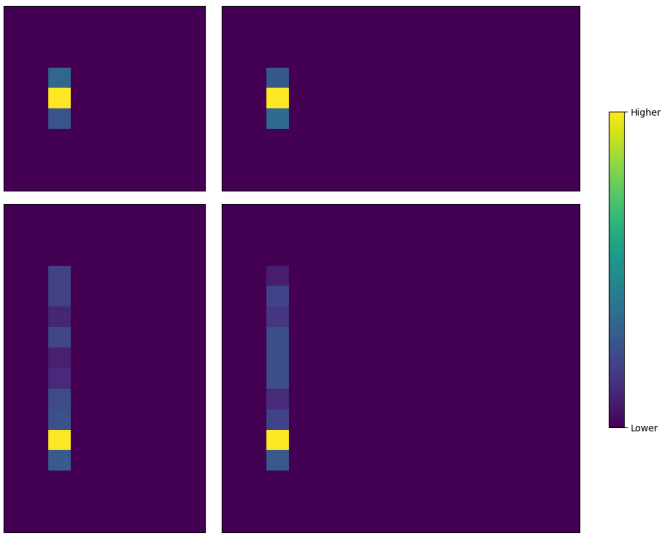

Fig. 6. A stretched clone. Firing like a place cell. In the top left, the original small square room (SS) in which the agent was trained. In the top right: the horizontal rectangle room (HR), bottom left: vertical rectangle room (VR), and bottom right: the large square room (LS). Grid locations are colored according to how often this clone fires in that position. Brighter, yellower coloration representing more frequent firing at a given location. The clone's firing pattern in HR matches its pattern in the original SS environment, yet in the VR and LS environments it becomes extended. This tracks with the observations of O'Keefe. Note: This heatmap includes the barrier wall surrounding each environment.

The ideal case is depicted in Figure 7. Here the clone has fixed onto a single location in the small square environment (SS) and continues to fire at that location even in the other environments.

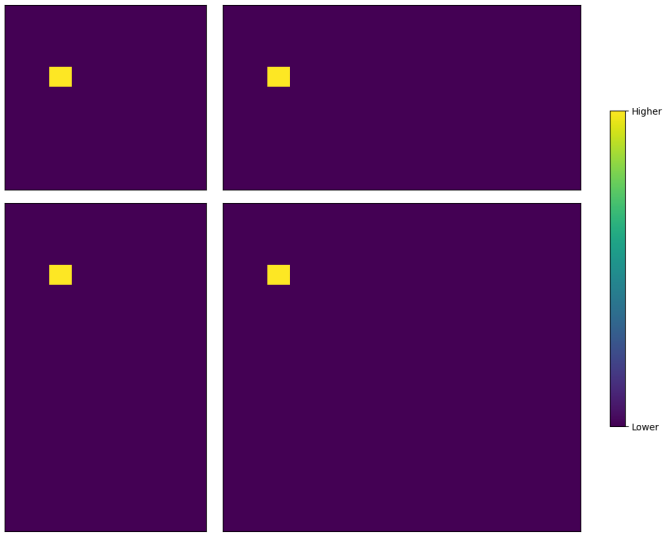

Fig. 7. A clone which has fixed to a specific location in the environment. This clone transfers precise topographic knowledge across environments. Rooms arranged and displayed in the same manner as Figure 6.

A less informative clone is depicted in Figure 8. The clone fires through much of the interior of each environment. However it prefers firing at the bottom of its range and strongly avoids the lower and left most walls.

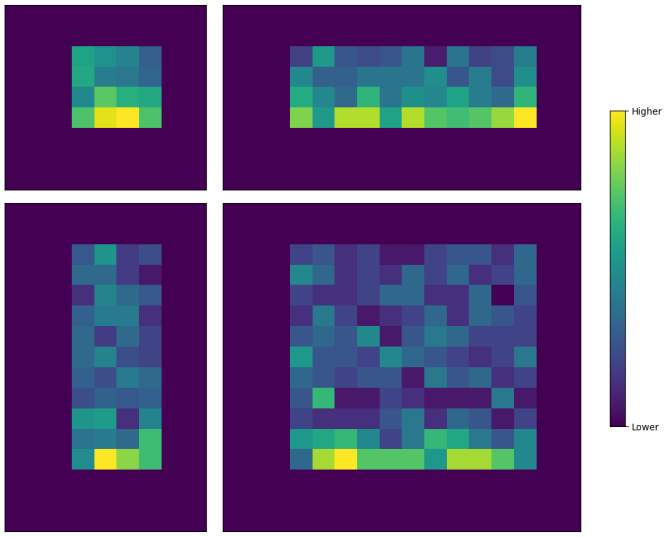

Fig. 8. An emerging clone. Rooms are again arranged in the same manner as Figure 6. A few more rounds of cloning and this clone may encode a unique location. However it already provides partial topographic information by preferring to fire at the bottom center of its range and away from barrier walls. This firing pattern is repeated in the novel environments.

To demonstrate the impact of oblivion thresholding the same 4-layer cloning process from the previous section is repeated without thresholding and compared to the thresholded process in Figure 9.

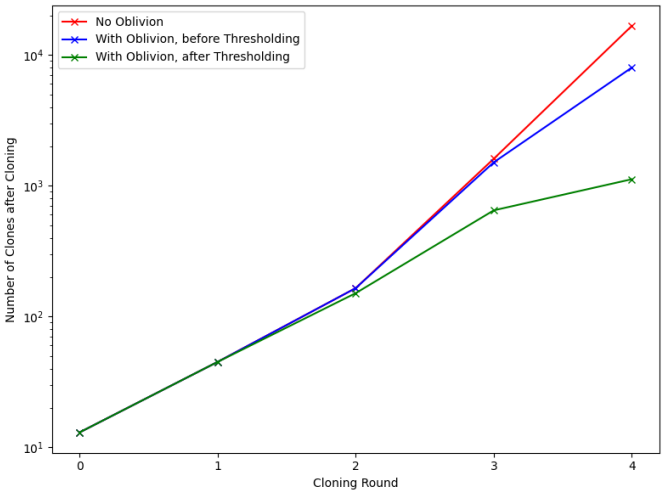

Fig. 9. The impact of oblivion. The average clones at each round of cloning before (Blue) and after (Green) an oblivion threshold is applied. Compared to the number of clones if no thresholding occurs (Red). From the same training and cloning process as described in Section 4A. Note the logarithmic scale.

V. Discussion

State cloning allows the GraphLearner to learn spacial topographies of explored environments. In other words the Cloned GraphLearner builds a schema representation of its learned environment. The topographic knowledge from this schema can be readily transferred to similar environments as demonstrated by replicating the O'Keefe experiments. This place cell like behavior is best seen in Figure 6, which closely follows O'Keefe's results.

The clones of the Cloned GraphLearner all reflect meaningful spacial information. Some like the clone in Figure 7 correspond to an exact location. Any complex behavior linked to this clone can be safely expected to always happen at this location, even in novel environments which resemble the original learned environment. In other words these location specific behaviors can be transferred across common environments.

While the clone in Figure 8 does not provide exact location its bias is clear. In fact this clone represents a snapshot of the knowledge gained from the cloning process. Clones of this clone will have more restricted firing fields, providing increasingly precise spacial information.

Other algorithmic techniques have already achieved similar results, such as Clone Structured Cognitive Graphs. However to be efficiently calculated CSCGs require a predetermined limit on the number of clones of each observed state. By contrast the Cloned GraphLearner maintains efficiency by limiting the depth of its sequence history h. This value can always be increased with further rounds of cloning. The Cloned GraphLearner is highly modifiable. Layers of clones can be deleted and rebuilt as desired, without impacting lower layers.

While the experiments in Section 4 reused the same training sequence at all stages of the cloning process this is not strictly necessary. It is possible for the Cloned GraphLearner to perform cloning in an online manner, with each successive round of cloning relying on a new sequence provided that sequence comes from the same underlying distribution.

As previously noted, the processes of clone creation and clone connection could be merged to occur during the same pass of the training sequence. The oblivion thresholding process would then need to delete not just clones but also all edges containing those clones. This is particularly problematic as the oblivion thresholding used in this paper is only a simple, greedy method of minimizing clone numbers. More advanced methods, such as the clone merging technique employed by CSCGs, must be tested. Keeping clone creation and connection separate leaves open more developmental paths.

VI. Conclusion

The Cloned GraphLearner builds a topographic schema which can be transferred across environments. It is neuromorphic with the firing patterns of clones resembling those of place cells in the hippocampus. This is demonstrated by replicating the rodent place cell experiments of O'Keefe and Burgess. Importantly it does all this while retaining the online-learning, memory efficient, and explainable characteristics of the original GraphLearner. It has untapped potential in real world applications, where topological structures may be too complex to sample and learn in offline manners or store in memory. Hopefully the Cloned GraphLearner can aid in advances both in Neuroscience and in Artificial Intelligence.

References

[1] J. Hawkins and S. Blakeslee, *On Intelligence*. New York, NY, USA: Macmillan, 2004.

[2] V. B. Mountcastle, "The columnar organization of the neocortex," *Brain*, vol. 120, no. 4, pp. 701–722, Apr. 1997, doi: 10.1093/brain/120.4.701.

[3] B. H. Bloom, "Space/time tradeoffs in hash coding with allowable errors," *Commun. ACM*, vol. 13, no. 7, pp. 422–426, Jul. 1970.

[4] J. Blustein and A. El-Maazawi, "Bloom filters: a tutorial, analysis, and survey," Dalhousie Univ., Halifax, NS, Canada, Dec. 2002, pp. 1–31.

[5] L. Fan, P. Cao, J. Almeida, and A. Z. Broder, "Summary cache: a scalable wide-area Web cache sharing protocol," *IEEE/ACM Trans. Netw.*, vol. 8, no. 3, pp. 281–293, Jun. 2000, doi: 10.1109/90.851975.

[6] T. Harrison and H. Unger, "The GraphLearner as a high order Markov chain simulator," in *Proc. 16th Int. Conf. Autonomous Syst.*, 2024.

[7] A. A. Markov, *Extension of the Law of Large Numbers to Quantities Depending on One Another*. St. Petersburg, Russia: Imperial Acad. Sci., 1906 (in Russian).

[8] R. D. Fields, "Making memories stick," *Sci. Am.*, vol. 292, no. 2, pp. 74–81, 2005.

[9] T. Harrison and H. Unger, "GraphLearner: An approach to sequence recognition and generation," in *Proc. Mallorca Workshop Autonomous Syst.*, 2023.

[10] D. George *et al.*, "Clone-structured graph representations enable flexible learning and vicarious evaluation of cognitive maps," *Nat. Commun.*, vol. 12, Art. no. 2392, 2021, doi: 10.1038/s41467-021-22559-5.

[11] G. V. Cormack and R. N. S. Horspool, "Data compression using dynamic Markov modelling," *Comput. J.*, vol. 30, no. 6, pp. 541–550, Dec. 1987, doi: 10.1093/comjnl/30.6.541.

[12] J. Hawkins, D. George, and J. Niemasik, "Sequence memory for prediction, inference and behaviour," *Philos. Trans. R. Soc. B*, vol. 364, no. 1521, pp. 1203–1209, May 2009, doi: 10.1098/rstb.2008.0322.

[13] R. U. Muller and J. L. Kubie, "The effects of changes in the environment on the spatial firing of hippocampal complex-spike cells," *J. Neurosci.*, vol. 7, no. 7, pp. 1951–1968, Jul. 1987, doi: 10.1523/JNEUROSCI.07-07-01951.1987.

[14] J. O'Keefe and N. Burgess, "Geometric determinants of the place fields of hippocampal neurons," *Nature*, vol. 381, no. 6581, pp. 425–428, May 1996, doi: 10.1038/381425a0.
[15] A. Bubic, D. Y. von Cramon, and R. I. Schubotz, "Prediction, cognition and the brain," *Front. Hum. Neurosci.*, vol. 4, 2010, doi: 10.3389/fnhum.2010.00025.
[16] J. O'Keefe and L. Nadel, *The Hippocampus as a Cognitive Map*. Oxford, U.K.: Clarendon Press, 1978.
[17] K. Kansky *et al.*, "Schema networks: zero-shot transfer with a generative causal model of intuitive physics," in *Int. Conf. Mach. Learn.* (ICML), 2017, pp. 1809–1818.
[18] A. Dedieu *et al.*, "Learning higher-order sequential structure with cloned HMMs," *arXiv* preprint arXiv:1905.00507, 2019. [Online]. Available: https://arxiv.org/abs/1905.00507