How do Active Dendrite Networks Mitigate Catastrophic Forgetting?

Sankarshan Damle[1]* **Satya Lokam**[2] **Navin Goyal**[2]
[1]LIA, EPFL [2]Microsoft Research India
sankarshan.damle@epfl.ch {satya.lokam,navingo}@microsoft.com

Abstract

We investigate the efficacy of Active Dendrite Networks (ADNs) in mitigating catastrophic forgetting in Continual Learning (CL). We consider Sparse Parity and Modular Addition to be our CL-task sequences. ADNs mitigate forgetting for Sparse Parity, but not for Modular Addition. For Sparse Parity, we perform an interpretability analysis to highlight the effectiveness of orthogonal (or uncorrelated) context vectors and task vectors in mitigation. We demonstrate that uncorrelated context vectors facilitate the creation of distinct subnetworks within ADNs, aiding in task separation. We also look at task uncorrelatedness to explain the difference in ADN's performance for Sparse Parity and Modular Addition.

1 Introduction

Continual learning (CL) is a challenging problem in artificial neural networks (ANNs), where models must learn new tasks sequentially without forgetting previously learned tasks [12]. When sequentially trained, ANNs suffer from *catastrophic forgetting* [4], i.e., the model's performance on earlier tasks degrades as it learns new ones. This is in sharp contrast to humans and other animals who appear to learn in a continual fashion [3].

This paper explores the potential of *Active Dendrite Networks* (ADNs) [6] to address this issue, focusing on two specific tasks: `Sparse Parity` [1] and `Modular Addition` [9]. Focusing on these synthetic tasks and simple ADNs enables an interpretability analysis to identify aspects of dendritic architecture and task structures that help (and do not help) avoid catastrophic forgetting. ADNs define a neural network architecture designed to mimic the complex processing and local learning capabilities of biological neurons. ADNs replace point neurons of ANNs with *pyramidal* neurons that have *dendrites* to process non-linearity. Dendrites process *context signals* that encode task information and modulate the feed-forward activity of excitatory neurons [11].

Existing literature [6, 11] hypothesizes that ADNs overcome catastrophic forgetting through task separation, i.e., through distinct (task-specific) subnetworks modulated by dendrites. This paper provides further evidence for continual learning abilities and stability-plasticity trade-offs in ADN's by demonstrating how they learn from context vectors and task similarities. Our takeaway is that uncorrelated context & task vectors and uncorrelated tasks help ADNs to overcome forgetting.

Concretely, **first**, we show that uncorrelated context vectors and uncorrelated task vectors (i.e., task IDs appended to the feature set) help mitigate forgetting for `Sparse Parity` with non-overlapping support. However, when the support for different `Sparse Parity` tasks overlap, we see a 5% drop in ADN's performance even when the context and task vectors are uncorrelated. **Second**, ADNs do not overcome forgetting in `Modular Addition`. This, along with the drop in ADN's performance for overlapping `Sparse Parity`, motivates *task uncorrelatedness* (i.e., the correlation between

*Most of this work was done while the author was at Microsoft Research India.

subsequent CL tasks) as the third criterion for overcoming forgetting. We measure task correlation by the number of epochs Task j takes to generalize when initialized with Task $(j-1)$ weights. For `Modular Addition` and overlapping `Sparse Parity`, Task j generalizes faster with $(j-1)$ weights than with random initialization. In contrast, non-overlapping `Sparse Parity` takes more epochs to generalize, suggesting that task support orthogonality aids task separation.

2 Background and Continual Learning (CL) Setup

Active Dendrite Neural Networks (ADNs). ADNs [6] augment the standard multi-layer perceptron architecture with the following biological properties:

1. The point neuron-architecture is replaced with a *pyramidal* neuron-architecture. Biological neurons are pyramidal and have complex *dendrites* that process inputs non-linearly. In ADNs, these pyramidal neurons process feed-forward inputs using a linear weighted sum and contextual inputs via independent dendrite segments with separate weights.

   Given an input vector $\mathbf{x}$, weights $\mathbf{w}$, and bias $\mathbf{b}$, the pyramidal neuron computes: $\hat{t} = \mathbf{w}^T\mathbf{x} + \mathbf{b}$. Each dendrite segment j with weight $\mathbf{u}_j$ and context vector $\mathbf{c}$, is used to select the segment with the strongest response to the context when computing dendrite activation d, as $d = \langle \mathbf{u}_{j^\star}, \mathbf{c} \rangle$ where $j^\star = \arg\max_j |\langle \mathbf{u}_j, \mathbf{c} \rangle|$.
2. Contextual inputs on active dendrites modulate the neuron's response, increasing its likelihood to fire. Given $\sigma(\cdot)$ as the sigmoid function and $f(\cdot)$ as a *modulation* function, the output of each pyramidal neuron is $\hat{y} = f(\mathbf{w}, \mathbf{b}, \mathbf{u}_{j^\star}; \mathbf{x}, \mathbf{c}) = \big(\mathbf{w}^T\mathbf{x} + \mathbf{b}\big) \times \sigma\left(\langle \mathbf{u}_{j^\star}, \mathbf{c} \rangle\right)$.
3. Neural activity and connectivity are highly *sparse*. In ADNs, this sparsity is enforced using a κ-Winner-Take-All (κWTA) function to ensure sparse activations. Formally, for each pyramidal neuron i, $\kappa\mathsf{WTA}(\hat{y}_i) = \hat{y}_i$ *if* $\hat{y}_i$ is one of the top κ activations over all i, and zero otherwise.

CL Tasks. In this paper, we consider the following tasks.

1. **Sparse Parity [8].** Consider the (n,k)-parity function, where we have a dataset X consisting of records $\mathbf{x}_i \in \{-1,+1\}^n$. Based on X, we define two types of parity tasks:
   - `Parity1`. Given $k \ll n$ and a random permutation π of $[n]$, we construct tasks by dividing the vector indices into disjoint segments of size k. Specifically, for each task j, and for each record $\mathbf{x}_i \in X$, the label $y_{i,j}$ is computed as $y_{i,j} = \prod_{m=(kj+1)}^{k(j+1)} \mathbf{x}_{i,\pi(m)}$. This means that the indices $[n]$ are divided into n/k segments, where the j-th segment is associated with Task j.
   - `Parity2`. Given $k \ll n$ and a random permutation π of $[n]$, `Parity2` tasks are defined by overlapping segments. For each task j, and for each record $\mathbf{x}_i \in X$, we compute the label $y_{i,j}$ as $y_{i,j} = \prod_{m=(j+1)}^{k+j} \mathbf{x}_{i,\pi(m)}$. Each task j is defined by k consecutive positions under π, resulting in overlapping support between adjacent tasks.

   To summarize, in `Parity1`, each task's parity computation is performed on disjoint vector segments, ensuring pair-wise orthogonality. In contrast, `Parity2` computes parity over overlapping segments, with adjacent tasks sharing part of their support.

   For both parity tasks, we have the CL setup given by $\{\{(\mathbf{x}_i, \mathbf{j}), y_{i,j}\}_{i\in[|X|]}\}_{j\in[\tau]}$, where τ is the total number of tasks and $(\mathbf{x}_i, \mathbf{j})$ the feature-tuple comprising the *task vector* $\mathbf{j}$ for each task j.
2. **Modular Addition [2, 9].** Consider the cyclic group C_p for any prime p. For all pair of elements $a, b \in C_p$ let $\mathbf{x}_i = \mathbf{a}||\mathbf{b}$, where $\mathbf{a}$ and $\mathbf{b}$ are one-hot vectors of a and b, respectively. For each such $\mathbf{x}_i$, we set the corresponding label as $y_i = (a+b) \bmod p$.

   For modular addition, the CL setup includes sequentially training a model on different groups C_{p_j}, where p_j, $\forall j \in [\tau]$ is a prime. We also append the feature set with a unique *task vector* $\mathbf{j}$ corresponding to each set, i.e., $(\mathbf{x}_i, \mathbf{j})$ is the feature tuple for the task j.

Grokking [10]. Grokking is a phenomenon where models suddenly exhibit a significant performance improvement after a prolonged training period, often without any changes in the training process. Recent literature has observed grokking for both tasks: sparse parity [1, 8] and modular addition [2, 9].

Average Accuracy (AA) [12]. Let $a_{i,j} \in [0,1]$ be the accuracy of a model on the test-set of the jth task after sequential training for the ith task (where $j \leq i$). Then, AA is the model's overall performance at the ith task: $\text{AA}_i = \frac{1}{i}\sum_{j=1}^{i} a_{i,j}$.

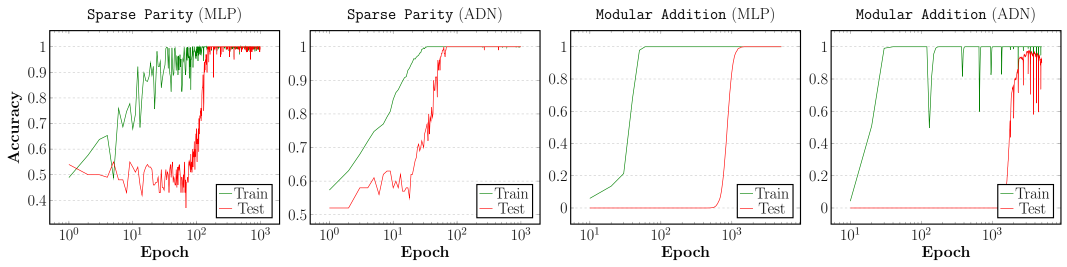


Figure 1: Grokking: Train and test accuracies for `Sparse Parity` and `Modular Addition`.

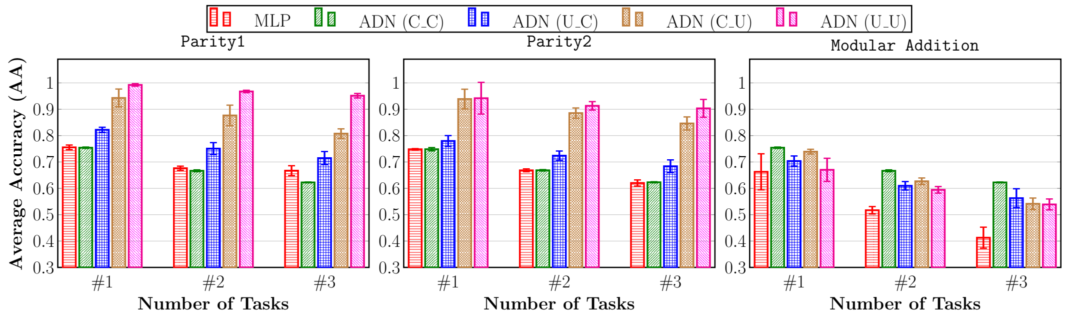


Figure 2: Average Accuracy (AA) scores for `Sparse Parity` and `Modular Addition`. ADN (x_y), $x, y \in \{U, C\}$, represents different setups with x corresponding to uncorrelated/correlated context vectors and y task operators. For Task #0, all 5 models generalize, i.e., AA $= 1.0$.

3 Results

Setup. We train an ADN model with 2 hidden layers of 2048 pyramidal neurons each with a classification head. Unlike prior work [6, 11], *we train each task till generalization*. We also train a 2-layer MLP with 2048 neurons in each hidden layer for our baseline. For both, we use the Adam optimizer [7] with constant weight decay and learning rate across all CL tasks. We refer the reader to Appendix A for details on other hyperparameters.

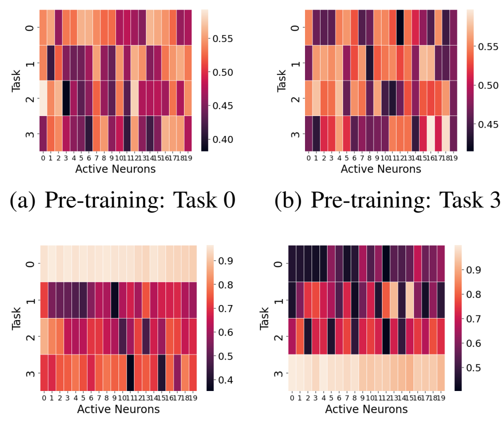
Context Vectors $(\mathbf{c})$ *& Task Vectors* $(\mathbf{j})$. Iyer et al. [6] use the mean of the task-set of permutedMNIST [5] as its choice for context vectors. Unlike permutedMNIST, features for `Sparse Parity` and `Modular Addition` are not well-separated in the feature space, so we randomly sample task-wise (i) **uncorrelated** (or orthogonal) and (ii) **correlated** context vectors. We also append task vectors to each feature. We randomly append task-wise (i) **uncorrelated** (or orthogonal) and (ii) **correlated** task vectors. We use ADN (x_y), where $x, y \in \{U, C\}$, to denote these setups with x corresponding to context vectors and y task operators. Lastly, the input to ADN is $((\mathbf{x}_i, \mathbf{j}), \mathbf{c})$, where the neurons receive the features $(\mathbf{x}_i, \mathbf{j})$ as input, and the dendrites use the context $\mathbf{c}$ for modulation.

`Sparse Parity`. From [8], we set $n = 40$, $k = 3$ and $\tau = 4$. We minimize the *hinge* loss, i.e., $l(y, \hat{y}) = \max(0, 1 - y \cdot \hat{y})$, where y and $\hat{y}$ is the ground truth and the model output, respectively.

`Modular Addition`. Motivated from [9], we consider a CL setup for $\tau = 4$ with primes $p_1 = 113, p_2 = 107, p_3 = 103$ and $p_4 = 101$. We treat these as a classification task with the dimension of the model's output head fixed at 113. We minimize the *cross-entropy* loss [2].

ADNs Exhibit Grokking. We first study grokking in ADNs (refer to Figure 1). For these, we mimic the training details and other hyperparameters for both MLP and ADN from Merrill et al. [8] for `Sparse Parity` and Chughtai et al. [2] for `Modular Addition`. The generalization accuracy of ADNs also sees a sudden increase post memorization, i.e., ADNs also grok.

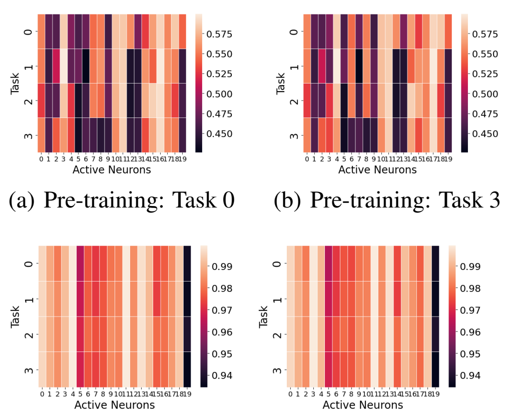
ADN-CL: Results. Figure 2 presents the average and standard deviation of average accuracy (AA), computed across 3 different runs. Our vanilla-MLP baseline shows catastrophic forgetting. Unsurprisingly, we see that ADNs overcome forgetting compared to the baseline. The AA at the end of the 4th task for ADN (U_U) is **0.951**$\pm$**0.0084** (+0.284 over MLP) for `Parity1` and **0.903**$\pm$**0.0338**



(a) Pre-training: Task 0 (b) Pre-training: Task 3

(c) Post-training: Task 0 (d) Post-training: Task 3

Figure 3: ADN (U_U): Plotting $\sigma\left(\mathbf{u}_{j^\star}^T \mathbf{c}\right)$



(a) Pre-training: Task 0 (b) Pre-training: Task 3

(c) Post-training: Task 0 (d) Post-training: Task 3

Figure 4: ADN (C_C): Plotting $\sigma\left(\mathbf{u}_{j^\star}^T \mathbf{c}\right)$

(+0.283 over MLP) for `Parity2`. For `Modular Addition`, AA for ADN (C_C) at the end of the 4th task is **0.622** $\pm$ **0.0014** (+0.21 over MLP). Appendix B presents results for (i) $(n = 40, k = 3)$ with $\tau = 10$ and (ii) $(n = 80, k = 3)$ with $\tau = 4$. Further observations are available in Appendix B.1.

4 Interpreting ADNs for `Parity1`

Similar to prior works [6, 11], we hypothesize that ADNs create *task-specific distinct subnets* that help mitigate catastrophic forgetting. This section presents interpretability measures that show that uncorrelated context vectors and task vectors lead to task-specific distinct subnets. We begin by isolating these subnetworks.

Isolating Subnetworks for `Parity1`. After training of the 4th task, we take the mean of the task-wise activations of the 2048 pyramidal neurons from the two layers separately on the test dataset for ADN (U_U) and ADN (C_C). We sort the neurons based on the mean activations and pick the top 1% of neurons for each task as its 'active' set of neurons[2].

Distinct Subnetworks for ADN (U_U). Here, we look at the overlap between active subnets for each task (refer to Table C.1). For ADN (U_U), the maximum fractional overlap for Layer 0 is 0.0 and 0.05 for Layer 1. However, for ADN (C_C), the maximum fractional overlap for Layer 0 is 1.0 and 0.95 for Layer 1. That is, uncorrelated dendrite segments result in distinct subnetwork formation.

Q: How do Uncorrelated Context Vectors Enforce Subnets?

We now explore how uncorrelated context vectors create these subnets. Figures 3 and 4 plot $\sigma\left(\mathbf{u}_{j^\star}^T \mathbf{c}\right)$ pre and post-training for ADN (U_U) and ADN (C_C), respectively. We see that ADN (C_C) results in higher $\sigma\left(\mathbf{u}_{j^\star}^T \mathbf{c}\right)$ values for *all* tasks. In contrast, ADN (U_U) has higher $\sigma\left(\mathbf{u}_{j^\star}^T \mathbf{c}\right)$ for distinct tasks. For ADN (U_U), the dendrite segments selected through κWTA create distinct subnets. Given a task, distinct neurons are more likely to fire and hence, more likely that κWTA selects them – leading to distinct subnets. For ADN (C_C), any neuron can fire and consequently pass the κWTA filter leading to overlapping active subnets.

We further visualize the dendrites "on-off" mechanism to create distinct subnets in Figures C.1 and C.2 (Appendix C.1) by plotting $\sigma\left(\mathbf{u}_j^T \mathbf{c}\right)$, $\forall j \in [4]$. For any task t and ADN (U_U), any dendrite $\mathbf{u}_j$ "switches-off" its pyramidal neuron by aligning orthogonally to task context vector $\mathbf{c}$ resulting in a post-sigmoid value of ≈ 0.5. To switch the neuron on, the context and dendrites align parallelly.

Q: Why do Uncorrelated Context Vectors Work?

We now define a couple of *progress measures* [9] to understand how uncorrelated context vectors assist in distinct subnet formation.

[2]This active subnet is also functional as it realizes the full model's predictions (Appendix C.1).

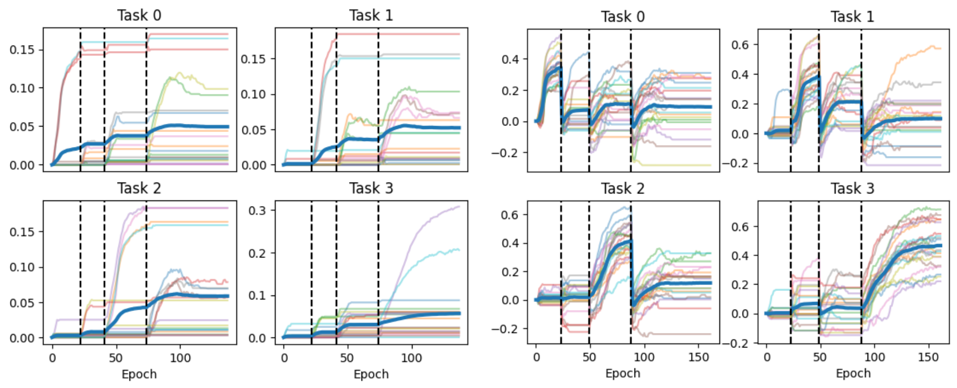


Figure 5: Dendrite Weight Movement (left) and Dendrite Activation Movement (right) for ADN (U_U) and `Parity1`. The dashed vertical lines are task boundaries and the thick blue horizontal lines denote the mean value.

Dendrite Weight Movement (DWM). For each active neuron i for a task j, we plot $||\mathbf{u}_{i,t} - \mathbf{u}_{i,0}||_\infty$ as epoch t increases. Figure 5 (left) depicts the results. We see a pattern: for the active neurons of the j^{th} task, their segments see an increase in the weight movement value as the j^{th} task starts. The weight movement tends to saturate once the j^{th} task ends.

Dendrite Activation Movement (DAM). For each active neuron i for a task j, we plot $|\langle \mathbf{u}_{i,t}, \mathbf{c}_j \rangle - \langle \mathbf{u}_{i,0}, \mathbf{c}_j \rangle|$ as epoch t increases. Figure 5 (right) depicts the results. While the task-specific pattern is obvious, we also see that the uncorrelated context vectors actively bring the dendrite activation to initialization levels for tasks for which the neuron is not active.

Note: When the fraction overlap between active subnets is high (e.g., for ADN (C_C)), we (trivially) get similar DWM and DAM plots across all four tasks (refer to Figure C.3).

5 Task Uncorrelatedness: The Final Piece!

Figure 2 shows that pair-wise orthogonality of context vectors and task vectors in ADNs overcomes catastrophic forgetting. The difference in performance for `Parity1` and `Parity2` suggests another criterion: *task uncorrelatedness*. For `Parity2` (with overlapping supports) the average accuracy at the end of 4th task is $\approx 5\%$ less than `Parity1` (with non-overlapping supports). By isolating task-specific subnets for `Parity2`, we see an increase in active neuron overlap, highlighting the effect of the lack of task uncorrelatedness (Appendix C.2).

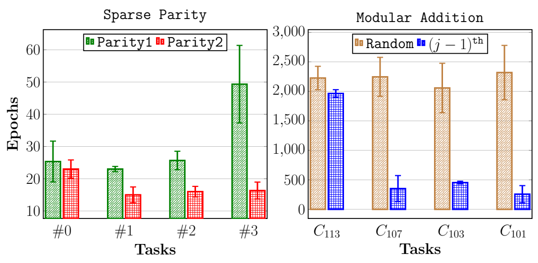


Figure 6: Number of epochs for Task j to generalize when continually trained up to Task $(j-1)$. For `Modular Addition`, the baseline (Random) represents the epochs needed for ADN to generalize with random weight initialization. For `Sparse Parity`, Task $\#0$ is the baseline as the supports are random.

This criterion explains ADN's ineffectiveness in overcoming forgetting for `Modular Addition`. Specifically, we empirically show that `Modular Addition` (i.e., $C_{113}, C_{107}, C_{103}$ and C_{101}) is also pair-wise correlated. To quantify adjacent task uncorrelatedness, in Figure 6, we plot the mean and standard deviation (across 3 runs) of the number of epochs ADN takes to generalize for Task j when continually trained till Task $(j-1)$. When the tasks are pair-wise uncorrelated (`Parity1`), we see an increase in the epochs taken for generalization. That is, Task $(j-1)^{\text{th}}$ weights are used as 'initialization' for Task j, and an increase in epochs taken for Task j represents the difference in the two tasks. Whereas, for `Parity2` and `Modular Addition`, we see a significant drop in the epochs taken. This shows that tasks are correlated as Task $(j-1)^{\text{th}}$ weights used for initialization aid Task j in generalizing faster.

References

[1] Boaz Barak, Benjamin Edelman, Surbhi Goel, Sham Kakade, Eran Malach, and Cyril Zhang. Hidden progress in deep learning: Sgd learns parities near the computational limit. *Advances in Neural Information Processing Systems (NeurIPS)*, 35:21750–21764, 2022.

[2] Bilal Chughtai, Lawrence Chan, and Neel Nanda. A toy model of universality: reverse engineering how networks learn group operations. In *Proceedings of the 40th International Conference on Machine Learning (ICML)*, 2023.

[3] Joseph Cichon and Wen-Biao Gan. Branch-specific dendritic ca2+ spikes cause persistent synaptic plasticity. *Nature*, 520(7546):180–185, 2015.

[4] Robert M French. Catastrophic forgetting in connectionist networks. *Trends in cognitive sciences*, 3(4):128–135, 1999.

[5] Ian J Goodfellow, Mehdi Mirza, Da Xiao, Aaron Courville, and Yoshua Bengio. An empirical investigation of catastrophic forgetting in gradient-based neural networks. *arXiv preprint arXiv:1312.6211*, 2013.

[6] Abhiram Iyer, Karan Grewal, Akash Velu, Lucas Oliveira Souza, Jeremy Forest, and Subutai Ahmad. Avoiding catastrophe: Active dendrites enable multi-task learning in dynamic environments. *Frontiers in neurorobotics*, 16:846219, 2022.

[7] Diederik P Kingma. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[8] William Merrill, Nikolaos Tsilivis, and Aman Shukla. A tale of two circuits: Grokking as competition of sparse and dense subnetworks. *arXiv preprint arXiv:2303.11873*, 2023.

[9] Neel Nanda, Lawrence Chan, Tom Lieberum, Jess Smith, and Jacob Steinhardt. Progress measures for grokking via mechanistic interpretability. In *The Eleventh International Conference on Learning Representations (ICLR)*, 2023.

[10] Alethea Power, Yuri Burda, Harri Edwards, Igor Babuschkin, and Vedant Misra. Grokking: Generalization beyond overfitting on small algorithmic datasets. *arXiv preprint arXiv:2201.02177*, 2022.

[11] Fahad Sarfraz, Elahe Arani, and Bahram Zonooz. A study of biologically plausible neural network: The role and interactions of brain-inspired mechanisms in continual learning. *Transactions on Machine Learning Research*, 2023.

[12] Liyuan Wang, Xingxing Zhang, Hang Su, and Jun Zhu. A comprehensive survey of continual learning: theory, method and application. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2024.

In this technical appendix, we provide additional training details (Appendix A), results for different `Sparse Parity` setups (Appendix B), and further interpretability analysis for `Parity1` and `Parity2` (Appendix C).

A Training Details

Table A.1 provides the complete list of hyperparameters used for `Sparse Parity` and `Modular Addition`. Furthermore, similar to [8], for Spare Parity we perform training with batch size 32. For Modular Addition, similar to [9], we do full batch training.

Sparse Parity

Hyperparameter	**Value**
Batch (=32) Training with Adam Optimizer	
Learning Rate	5e-4
Weight Decay	1e-6
(β_1, β_2)	$(0.9, 0.98)$
Number of Samples	10000
Train-test Split	60%-40%
MLP	
Input Feature Size	440
Output Size	1
Hidden Size	$(2048, 2048)$
Activation	ReLU
Dropout	0%
ADN Parameters	
Input Feature Size	440
Output Size	1
Hidden Size	$(2048, 2048)$
Context Size ($\mathbf{c}$)	400
Number of Dendrites	τ
κWTA	1%
Context Weight Sparsity ($\mathbf{c}$)	5%
Feedforward Weight Sparsity ($\mathbf{w}$)	50%
Dendrite Weight Sparsity ($\mathbf{u}$)	90%

Modular Addition

Hyperparameter	**Value**
Full Batch Training with Adam Optimizer	
Learning Rate	5e-3
Weight Decay	5e-1
(β_1, β_2)	$(0.9, 0.98)$
Number of Samples	$p_j \cdot p_j, \forall j \in [4]$
Train-test Split	60%-40%
MLP	
Input Feature Size	626
Output Size	113
Hidden Size	$(2048, 2048)$
Activation	ReLU
Dropout	0%
ADN Parameters	
Input Feature Size	626
Output Size	113
Hidden Size	$(2048, 2048)$
Context Size ($\mathbf{c}$)	400
Number of Dendrites	τ
κWTA	5%
Context Weight Sparsity ($\mathbf{c}$)	5%
Feedforward Weight Sparsity ($\mathbf{w}$)	50%
Dendrite Weight Sparsity ($\mathbf{u}$)	90%

Table A.1: Hyperparameters for Sparse Parity and Modular Addition

B Additional Results and Observations

Sparse Parity $(n = 40, k = 3)$ **with** $\tau = 10$**.** Figure B.7 depicts the results. We see that as the number of tasks increases, both the baseline MLP and ADN (C_C) quickly tend towards a random classifier (accuracy: 0.5). Whereas ADN (U_U)'s accuracy is $\approx 0.85 \pm 0.0205$ for `Parity1` and $\approx 0.8 \pm 0.01465$ for `Parity2`. The standard deviation (denoted with error bars in Figure B.7) is also low showing the stability of these results.

Sparse Parity $(n = 80, k = 3)$ **with** $\tau = 4$**.** Figure B.8 depicts the results. We see that $(n = 80, k = 3)$ is a harder CL task for ADNs compared to $(n = 40, k = 3)$. This is seen in the drop in performance as τ increases. For `Parity1`, at the end of the 4th task, ADN (U_U)'s average accuracy drops from 0.95 ± 0.0084 for $(n = 40, k = 3)$ to 0.81 ± 0.0204 for $(n = 80, k = 3)$. Likewise, for `Parity2`, the average accuracy drops from 0.90 ± 0.0338 for $(n = 40, k = 3)$ to 0.80 ± 0.0075 for $(n = 80, k = 3)$.

B.1 Observations

Weight Decay, Grokking, and Average Accuracy. Our experiments show that higher weight decay hurts the average accuracy. E.g., ADN (U_U)'s performance drops from ≈ 0.95 with weight decay 1e-6 to almost random (≈ 0.5) with weight decay 1. Interestingly, prior works show grokking on `Sparse Parity` [8] and `Modular Addition` [9] with larger weight decay, 1e-2/1e-3 and 1,

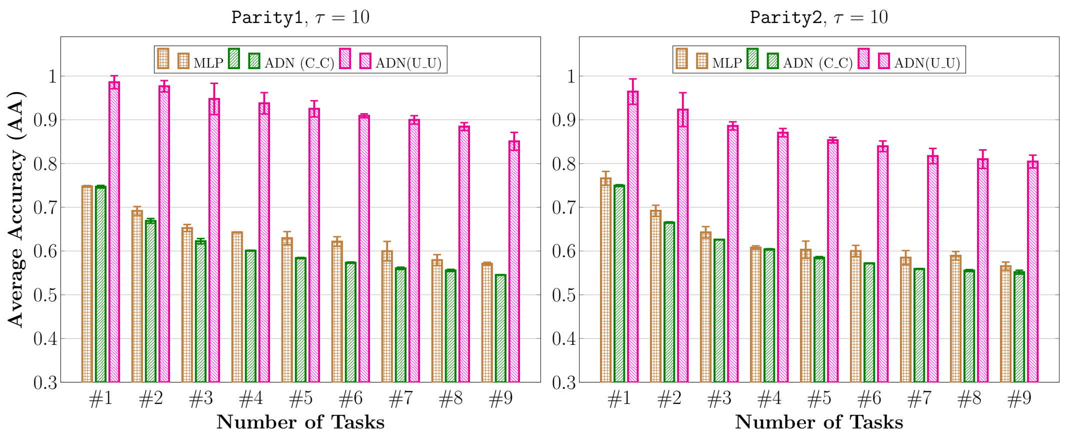


Figure B.7: Average Accuracy (AA) scores for Sparse Parity $(n = 40, k = 3)$ and $\tau = 10$. We use ADN (x_y), where $x, y \in \{U, C\}$, denotes different setups with x corresponding to correlated/uncorrelated context vectors and y task operators. For Task #0, all 3 models generalize, i.e., AA $= 1.0$. We keep κWTA at 2.5% for these experiments, while the other hyperparameters remain the same as in Table A.1.

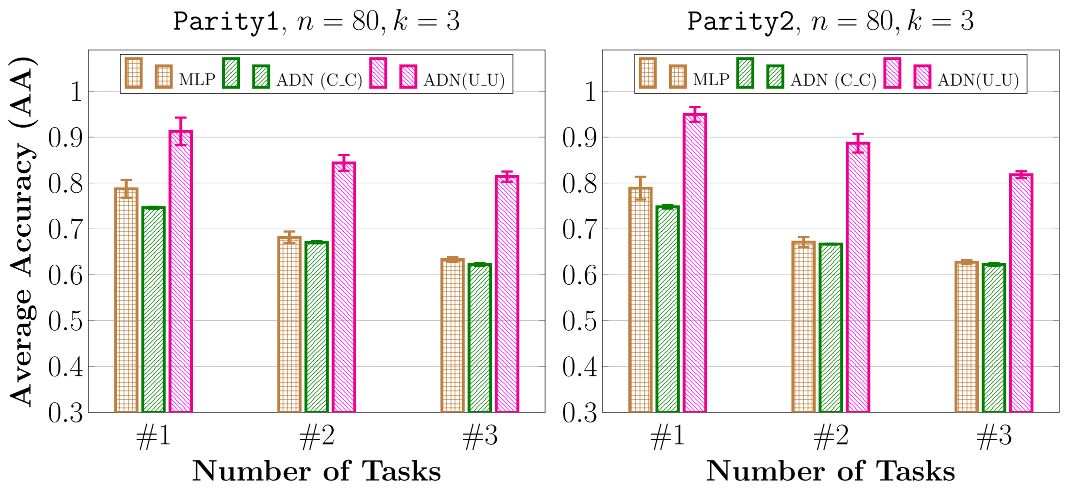


Figure B.8: Average Accuracy (AA) scores for Sparse Parity $(n = 80, k = 3)$ and $\tau = 4$. We use ADN (x_y), where $x, y \in \{U, C\}$, denotes different setups with x corresponding to correlated/uncorrelated context vectors and y task operators. For Task #0, all 3 models generalize, i.e., AA $= 1.0$. We keep κWTA at 2.5% for these experiments, while the other hyperparameters remain the same as in Table A.1.

Model	Tasks i	j	`Parity1` Layer 0	`Parity1` Layer 1	`Parity2` Layer 0	`Parity2` Layer 1
ADN (U_U)	0	1	0.0	0.05	0.35	0.15
	0	2	0.0	0.0	0.15	0.05
	0	3	0.0	0.0	0.0	0.05
	1	2	0.0	0.0	0.35	0.2
	1	3	0.0	0.0	0.1	0.1
	2	3	0.0	0.0	0.55	0.3
ADN (U_C)	0	1	0.4	0.15	0.85	0.45
	0	2	0.35	0.2	0.85	0.55
	0	3	0.6	0.05	0.85	0.25
	1	2	0.2	0.1	0.8	0.65
	1	3	0.5	0.0	0.8	0.5
	2	3	0.25	0.0	0.8	0.4
ADN (C_U)	0	1	0.0	0.0	0.35	0.4
	0	2	0.0	0.1	0.2	0.25
	0	3	0.0	0.2	0.0	0.15
	1	2	0.0	0.2	0.45	0.45
	1	3	0.0	0.1	0.2	0.25
	2	3	0.1	0.3	0.3	0.35
ADN (C_C)	0	1	1.0	0.9	1.0	1.0
	0	2	1.0	0.95	1.0	1.0
	0	3	1.0	0.95	1.0	1.0
	1	2	1.0	0.9	1.0	1.0
	1	3	1.0	0.95	1.0	1.0
	2	3	1.0	0.95	1.0	1.0

Table C.1: Fraction Overlap of Active Subnetworks: Top-1% of the neurons ranked by mean activations on the test set.

respectively. While ADNs grok for these weight decays (Figure 1), we observe that for smaller weight decays (e.g., 5e-3 for `Modular Addition`), they do not.

Sparsity and Average Accuracy. Sparsity is crucial in aiding task separation. In ADNs, sparsity is enforced using κWTA. We observe that a decrease in sparsity – enforced by increasing κWTA – reduces the average accuracy.

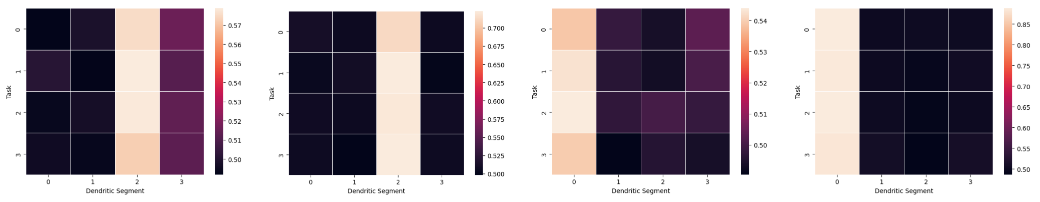
Comparing MLP's Performance with ADN (C_C). We see that the comparative performance of MLP and ADN (C_C) is dependent on the task correlation. For `Parity1` – when tasks are orthogonal – correlated context and task vectors result in a 0.0447 drop in ADN (C_C)'s performance compared to MLP. However, increased task correlation improves ADN (C_C)'s performance. For instance, for `Parity2`, ADN (C_C)'s performance compared to MLP is marginally higher (i.e., a relative accuracy increase of 0.002). For `Modular Addition` – where the tasks are highly correlated – ADN (C_C) comfortably outperforms MLP with a 0.2096 relative increase in average accuracy.

C Interpretability for Spare Parity

Appendix C.1 first provides further evidence of task separation with distinct subnets in `Parity1`. Next, we show that high task correlatedness in `Parity2` results in non-distinct subnetworks (Appendix C.2).

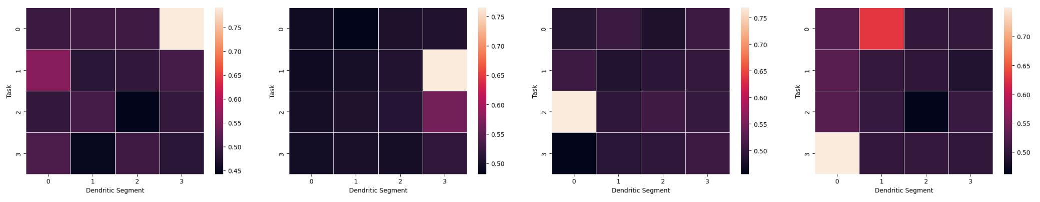
C.1 `Parity1`: Additional Experiments

Fraction Overlap of Active Subnetworks. In Section 4, we show that using mean activations computed over the test set allows us to isolate the subnets for each task. For ADN (U_U) we get distinct subnets, while for ADN (C_C) we get overlapping subnetworks. Table C.1 presents the task-wise numbers.



(a) Task 0 (b) Task 1 (c) Task 2 (d) Task 3

Figure C.1: ADN (C_C): Plotting $\sigma\left(\mathbf{u}_j^T\mathbf{c}\right), \forall j \in [4]$ for Parity1



(a) Task 0 (b) Task 1 (c) Task 2 (d) Task 3

Figure C.2: ADN (U_U): Plotting $\sigma\left(\mathbf{u}_j^T\mathbf{c}\right), \forall j \in [4]$ for Parity1

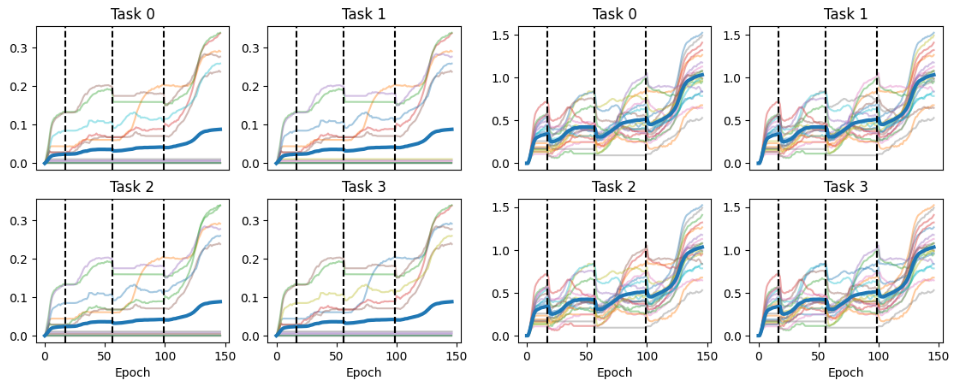
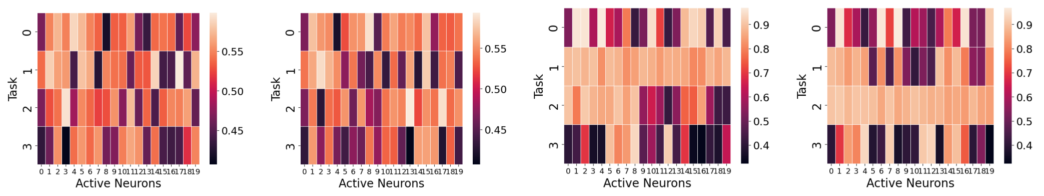


Figure C.3: Dendrite Weight Movement (left) and Dendrite Activation Movement (right) for ADN (C_C) and Parity1. The dashed vertical lines are task boundaries and the thick blue horizontal lines denote the mean value.

Functional Subnetwork. Here, we show that the task-specific subnetworks isolated in Section 4 are also functional. That is, they realize the full model's performance. To test the subnetwork's performance, we add a masked layer after each hidden layer that only keeps the weights of the active neurons and zero out the weights for the others. We see that the top-1% of the neurons realize 90% of the full model's performance and the top-5% realize 100%.

Plotting $\sigma\left(\mathbf{u}_j^T\mathbf{c}\right), \forall j \in [4]$**.** We further visualize the dendrites "on-off" mechanism to create distinct subnets in Figures C.1 and C.2 by plotting $\sigma\left(\mathbf{u}_j^T\mathbf{c}\right), \forall j \in [4]$. We see that for ADN (U_U) the dendrites help create distinct subnets by modulating the pyramidal neuron to fire only for a single task. In sharp contrast, for ADN (C_C), dendrites do not perform task-specific modulation and fire for all neurons leading to overlapping subnets.

Dendrite Weight Movement (DWM) and Dendrite Activation Movement (DAM). Similar to Figure 5, in Figure C.3, we also plot DWM and DAM for ADN (C, C). Both our progress measures show that the dendrites fire for all four tasks, thus explaining the large overlap between task-wise subnets.



(a) Pre-training: Task 1 (b) Pre-training: Task 2 (c) Post-training: Task 1 (d) Post-training: Task 2

Figure C.4: ADN (U_U): Plotting pre and post-training values of $\sigma\left(\max_j \mathbf{u}_j^T \mathbf{c}\right)$ for `Parity2`

C.2 `Parity2`

We explain the drop in ADN's performance for `Parity2`, compared to `Parity1`, because `Parity2` has correlated adjacent tasks. The lack of task uncorrelatedness results in overlapping subnetworks compromising task separation.

Increase in Subnetwork Overlap for ADN (U_U). For `Parity2`, we repeat the same process of isolating subnetworks as described for `Parity1` (refer to Section 4).

We again pick the top-1% of the 2048 neurons, from each layer, for each task. Table C.1 presents the task-wise numbers. Unlike for `Parity1`, we see an increase in the active set of neurons. Concretely, for `Parity2`, the maximum fractional overlap increases from 0.05 for `Parity1` to 0.55 for `Parity2`. This increase in the overlap is due to increased correlation in tasks for `Parity2` compared to `Parity1`.

ADN (U_U): Plotting $\sigma\left(\mathbf{u}_{j^\star}^T \mathbf{c}\right)$. Similar to Figure 3, we again plot $\sigma\left(\mathbf{u}_{j^\star}^T \mathbf{c}\right)$ for `Parity2`. For `Parity1`, we saw that dendrites modulated the activations so that a single pyramidal neuron only fires for a single task, creating task-specific subnets.

From the increased fractional overlap of active subnets for `Parity2`, we know that task-specific subnet formation is relatively less disjoint in this case. Figure C.4 depicts the numbers. We see that for several neurons, the dendrites fire for multiple tasks increasing active subnets' fractional overlap, thus explaining the performance drop.