ORIGINAL RESEARCH
published: 29 April 2022
doi: 10.3389/fnbot.2022.846219

Avoiding Catastrophe: Active Dendrites Enable Multi-Task Learning in Dynamic Environments

Abhiram Iyer[1,2†], Karan Grewal[1†], Akash Velu[3†], Lucas Oliveira Souza[1], Jeremy Forest[4] and Subutai Ahmad[1†]*

[1] Numenta, Redwood City, CA, United States, [2] Department of Electrical and Computer Engineering, Carnegie Mellon University, Pittsburgh, PA, United States, [3] Department of Computer Science, Stanford University, Stanford, CA, United States, [4] Department of Psychology, Cornell University, Ithaca, NY, United States

A key challenge for AI is to build embodied systems that operate in dynamically changing environments. Such systems must adapt to changing task contexts and learn continuously. Although standard deep learning systems achieve state of the art results on static benchmarks, they often struggle in dynamic scenarios. In these settings, error signals from multiple contexts can interfere with one another, ultimately leading to a phenomenon known as catastrophic forgetting. In this article we investigate biologically inspired architectures as solutions to these problems. Specifically, we show that the biophysical properties of dendrites and local inhibitory systems enable networks to dynamically restrict and route information in a context-specific manner. Our key contributions are as follows: first, we propose a novel artificial neural network architecture that incorporates active dendrites and sparse representations into the standard deep learning framework. Next, we study the performance of this architecture on two separate benchmarks requiring task-based adaptation: Meta-World, a multi-task reinforcement learning environment where a robotic agent must learn to solve a variety of manipulation tasks simultaneously; and a continual learning benchmark in which the model's prediction task changes throughout training. Analysis on both benchmarks demonstrates the emergence of overlapping but distinct and sparse subnetworks, allowing the system to fluidly learn multiple tasks with minimal forgetting. Our neural implementation marks the first time a single architecture has achieved competitive results in both multi-task and continual learning settings. Our research sheds light on how biological properties of neurons can inform deep learning systems to address dynamic scenarios that are typically impossible for traditional ANNs to solve.

Keywords: dendrites, continual learning, reinforcement learning, neuroscience, embodied cognition

Edited by:
*Adam Safron,
Johns Hopkins Medicine,
United States*

Reviewed by:
*Adeel Razi,
Monash University, Australia
Ikuko Smith,
University of California, Santa Barbara,
United States*

Correspondence:
*Subutai Ahmad
sahmad@numenta.com*

†These authors have contributed equally to this work

Received: *30 December 2021*
Accepted: *31 March 2022*
Published: *29 April 2022*

Citation:
Iyer A, Grewal K, Velu A, Souza LO, Forest J and Ahmad S (2022) Avoiding Catastrophe: Active Dendrites Enable Multi-Task Learning in Dynamic Environments. Front. Neurorobot. 16:846219. doi: 10.3389/fnbot.2022.846219

1. INTRODUCTION

Creating embodied systems that thrive in dynamically changing environments is a fundamental challenge for building intelligent systems. Humans handle such environments with ease, but today's deep learning systems struggle with them. Standard Artificial Neural Networks (ANNs) often fail dramatically when learning multiple tasks, a phenomenon known as *catastrophic forgetting* (McCloskey and Cohen, 1989; French, 1999) where the network forgets previously-learned information. ANNs are inherently designed for static environments with batch training, and learning multiple sequential tasks can lead to significant interference between tasks. Embodied systems, where an agent actively behaves in a changing environment, pose additional challenges. In dynamic scenarios, the training dataset itself is not fixed. Sensory inputs are dependent on an agent's actions and as an embodied agent learns, the actions taken for a given context change as well. Thus, a network learning in these situations needs to avoid forgetting relevant information, update only the information that requires fine tuning, and forget the information that is no longer relevant. The network must distinguish between these types of information categories instead of treating all information as equivalent. The optimal algorithms and architectures for learning in dynamic environments are unknown and remain a fundamental research challenge for AI.

We investigate these questions by looking to neuroscience and biological systems for clues to inform ANNs. In particular we hypothesize that biological properties of *pyramidal neurons* in the neocortex can enable targeted context-specific representations that avoid interference. Most ANNs today rely on an idealized (but inaccurate) model of neurons known as the *point neuron model*, consisting of a linear weighted sum of inputs followed by a non-linearity (**Figure 1**, left). Proposed over a hundred years ago (Lapique, 1907), the point neuron model continues to form the basis for current deep learning systems (McClelland et al., 1986; LeCun et al., 2015). In contrast, pyramidal neurons, which comprise most cells in the neocortex, are significantly more sophisticated and demonstrate a wide range of complex non-linear dendrite-specific integrative properties (Spruston, 2008; **Figure 1**, right). Experimental evidence suggests that dendrites are important for learning task-specific patterns (Yang et al., 2014). In this article we incorporate into an ANN two properties of biological neural networks: active dendrites, and sparsity *via* local inhibition.

We explore the impact of these properties in two non-traditional machine learning scenarios: *multi-task reinforcement learning* (multi-task RL) and *continual learning*. In multi-task RL, a robotic agent learns to perform a diverse set of independent tasks (Yu et al., 2019). Even though tasks are interleaved through training, standard ANNs suffer from significant task interference. In continual learning, a network is trained sequentially on a set of tasks and evaluated on all tasks after training (McCloskey and Cohen, 1989; van de Ven and Tolias, 2019). Here, standard ANNs do not perform well due to catastrophic forgetting. Specifically, because ANNs with point neurons overwrite most of their connections during each iteration of learning, tasks learned at the beginning of training are forgotten and receive low accuracy scores during the evaluation phase (French, 1999; Parisi et al., 2019).

The rest of the article is arranged as follows. After discussing background material, we propose a new architecture that incorporates dendrites and sparse representations into deep learning. We then test our architecture on one representative benchmark from each of the two scenarios, multi-task RL and continual learning. We show experimental results on a standard multi-task RL benchmark, Meta-World. We also show results on a standard continual learning benchmark, permutedMNIST. The results in both cases show that an identical architecture with active dendrites performs well in both benchmarks. Finally, we analyze the results and show that active dendrites and sparse representations help with catastrophic forgetting and gradient interference by learning to create task-specific subnetworks where representations are sparse and mostly orthogonal. Overall, our results suggest that detailed biological properties of neurons can be used to address dynamic scenarios that are difficult for traditional ANNs to solve.

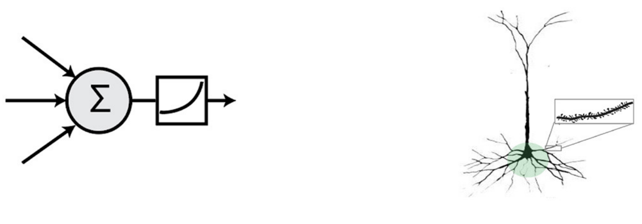

FIGURE 1 | (Left) The point neuron prevalent in most ANNs today computes a simple linear weighted sum of its inputs followed by a non-linearity. **(Right)** Morphology of a representative pyramidal neuron. Pyramidal cells in the brain exhibit a vastly more complex structure and functionality. Inset shows a prototypical basal dendritic segment that acts as an independent computational unit.

2. BACKGROUND

2.1. Multi-Task Learning

The goal of multi-task learning (Caruana, 1997) is to learn a single function that can solve a variety of different learning tasks. The literature in multi-task learning spans many subfields of machine learning, including computer vision (Misra et al., 2016; Kendall et al., 2019; Liu et al., 2019; Purushwalkam et al., 2019), and natural language processing (Dong et al., 2015; McCann et al., 2018). The fields of multi-task RL and continual learning can be seen as subsets of multi-task learning. In the former, tasks are learned in parallel. Conversely, in continual learning, tasks are learned in an ordered sequence.

Compared to single-task machine learning, learning multiple distinct tasks introduces new challenges. When using gradient-based learning algorithms such as backpropagation[1], one challenge is that error gradients and accumulated knowledge from different tasks can interfere with one another. The weight changes necessary to reduce the error for one task may be very different from the changes required for another task. This is a common problem sometimes defined as catastrophic forgetting (French, 1999) or catastrophic interference (McCloskey and Cohen, 1989) in continual learning.

Yu et al. (2020) propose a method to modify conflicting gradients through gradient projection. Several other works demonstrate that using or changing the gradients *via* various normalization, gradient-similarity, and regularization techniques

[1]In this article, the term *backpropagation* refers to the learning method used in deep learning (Rumelhart et al., 1986) and not the phenomenon of back propagating action potentials in dendrites.

can improve learning in multi-task settings (Zhang and Yeung, 2014; Chen et al., 2018; Sener and Koltun, 2018; Du et al., 2020). Novel network architectures are an alternate strategy for avoiding interference in multi-task computer vision settings. Rosenbaum et al. (2018) implement routing networks, learned functions that use task information to determine how to compose a set of function blocks. Liu et al. (2019), Maninis et al. (2019) demonstrate that attention-based architectures could also prevent task interference in multi-task learning scenarios.

2.1.1. Multi-Task Reinforcement Learning

Reinforcement learning (RL) is a branch of machine learning in which an agent acts in an environment and receives rewards for each action taken (Sutton and Barto, 2018). The goal is to train an agent, whose actions are determined by a *policy function*, to maximize the total reward received. One fundamental challenge of RL is that the training set itself is highly dynamic. As the agent learns and updates its policy function, it chooses different actions, which in turn changes the sequence of inputs that are received.

Deep RL uses deep learning networks to represent the policy function (see Arulkumaran et al., 2017 for a review). Recent years have witnessed the promise of deep RL in a variety of different settings. Mnih et al. (2013) demonstrate that an agent trained with their Deep Q-Network can surpass the performance of expert humans in Atari video games. A few years later, Silver et al. (2018) achieve superhuman performance in more challenging games such as Chess and Go. Other algorithms achieve strong performance in continuous environments with continuous action inputs (Lillicrap et al., 2016). Other methods attempt to induce beneficial learning behaviors such as more stable training (Schulman et al., 2017) and improved exploration (Haarnoja et al., 2018b).

Multi-task reinforcement learning combines Deep RL with multi-task learning (Wilson et al., 2007; Yang et al., 2020; Yu et al., 2020). Multi-task RL leads to particularly challenging and interesting scenarios where the system must address both dynamic training regimes and interference from multiple tasks. The idea of separating a neural network into different modules which are composed in a task-dependent manner is proposed in multi-task RL to prevent gradient interference (Andreas et al., 2017; Devin et al., 2017; Sahni et al., 2017; Haarnoja et al., 2018a; Goyal et al., 2020; Yang et al., 2020). Policy distillation, in which information from a "teacher" network is condensed to a smaller "student network," is another popular approach to combine information from different tasks in an effective manner (Rusu et al., 2016).

2.1.2. Continual Learning

While multi-task RL requires the simultaneous acquisition of multiple skills, continual learning requires the sequential acquisition of multiple skills. More generally, continual learning is the ability to acquire new knowledge over time while retaining relevant information from the past. A typical scenario involves training a network on a set of distinct tasks presented in a strict sequence of training phases. Testing the network involves measuring accuracy on all past tasks. van de Ven and Tolias (2019) and Parisi et al. (2019) extensively review the field. Two common approaches to catastrophic forgetting in continual learning involve regularization and subnetworks methods.

Regularization-based methods in continual learning regulate plasticity levels throughout the network during the course of training. In recent years, two of the most prominent examples of regularization are Elastic Weight Consolidation (EWC) (Kirkpatrick et al., 2017) and Synaptic Intelligence (SI) (Zenke et al., 2017). Both methods (EWC and SI) estimate the relevance of each weight of the network in solving each task. Inspired by the complex synapse structures seen in biology, SI uses an additional parameter per weight with internal dynamics that depend on the relevance of each weight to each task.

Subnetwork-based methods reduce task interference by identifying subpopulations of neurons that each learn one of the many tasks in the sequence. Gated Linear Networks (Veness et al., 2021) and Dendritic Gated Networks (Sezener et al., 2021) are examples of this type of approach and work by applying a gating mechanism that selects subnetworks based on the input. Context-dependent Gating (XdG) (Masse et al., 2018) selects predetermined subnetworks of neurons, but exact task information must be provided both at training and test times. Similarly, in Wortsman et al. (2020) each task is designated a sparse subset of neurons in the network.

2.2. Properties of Biological Neurons

Biological neural networks have evolved in ways that make them much more resilient to catastrophic forgetting and are able to perform significantly better in dynamical scenarios than any ANN to date. ANNs and their component point neurons emerged as simplified abstractions of the complex processes occurring in biological networks and neurons respectively. In this section, we explore the complexities of biological neural networks and review a few properties that are relevant to our work.

2.2.1. Neurons and Active Dendrites

The *pyramidal neuron* is the most prevalent neuron type found in the neocortex and hippocampal areas (Spruston, 2008; Ramaswamy and Markram, 2015). In particular they represent the most common excitatory neuron type found in areas associated with advanced cognitive functions (Spruston, 2008). A typical pyramidal neuron has an extensive dendritic tree containing thousands of synapses, each receiving input from another neuron (y Cajal, 1894; Bentivoglio and Swanson, 2001; Kandel, 2012). The *point neuron model* (Lapique, 1907) postulates that all of these synapses have a linear impact on the cell. This simple assumption formed the basis for Rosenblatt's original Perceptron (Rosenblatt, 1958) and continues to form the basis for current deep learning systems and ANNs (McClelland et al., 1986; LeCun et al., 2015).

Today it is well-known that the point neuron assumption is an oversimplified model of biological computations. Proximal synapses (close to the cell body) have a linear impact on the neuron, but the vast majority of synapses are located on distal dendritic segments (far from the cell body) and individually have minimal impact on the cell. These distal segments process groups of synapses locally in a non-linear fashion, and are known as *active dendrites* (Magee, 2000; Antic et al., 2010; Major et al., 2013;

Stuart and Spruston, 2015; Stuart et al., 2016). Empirical evidence (London and Häusser, 2005; Branco and Häusser, 2010) suggests that each distal dendritic segment acts as a separate active subunit performing its own local computation. Modeling studies show that neurons with active dendrites are more powerful and complex than the point neuron model can accommodate (Poirazi et al., 2003; Jadi et al., 2014; Poirazi and Papoutsi, 2020; Beniaguev et al., 2021).

When input to an active dendritic segment reaches a threshold, the segment initiates a *dendritic spike* (Antic et al., 2010). In basal dendritic segments, dendritic spikes travel to the cell body and can depolarize the neuron for an extended period of time, sometimes as long as half a second (Antic et al., 2010; Major et al., 2013; Gao et al., 2021). During this time, the cell is significantly closer to its firing threshold and any new input is more likely to make the cell fire. This suggests that basal active dendrites have a *modulatory*, long-lasting impact on the cell's response, with a very different role than proximal, or feedforward, inputs (Hawkins and Ahmad, 2016; Antic et al., 2018). Active dendritic segments typically receive contextual input that is a different input than received in proximal segments. These context signals can arrive from other neurons in the same layer, neurons in other layers, or in the form of top-down feedback. Recent experimental evidence has shown that the input on active segments can drive context-dependent activity (Takahashi et al., 2020). In our model, we incorporate these ideas and explore the possibility of using context to create task-specific subnetworks.

2.2.2. Sparse Representations

Neural circuits in the neocortex are highly sparse. Studies show that relatively few neurons spike in response to a sensory stimulus across multiple sensory modalities (Attwell and Laughlin, 2001; Barth and Poulet, 2012; Liang et al., 2019). Sparsity is also present in neural connectivity; cortical pyramidal neurons show sparse connectivity to each other and receive relatively few excitatory inputs from most surrounding neurons (Holmgren et al., 2003). These two phenomena are significantly different from standard ANNs, where both activations and connectivity are dense.

When modeling sparsity in ANNs, sparse neural representations are translated into vectors where most of the entries are off (i.e., equal to zero; Majani et al., 1989). Just like in dense representations, individual entries in a sparse representation can correspond to the presence of certain features (e.g., the unique position of an edge in an input image). One advantage of sparsity in representations is that vectors for two separate entities have low *overlap*, which means the set of features/entries that are non-zero in both vectors is small. Previous studies show that sparse representations are more resistant to noise than dense representations, and slight perturbations in the input are less likely to hinder a trained pattern recognizer (Ahmad and Hawkins, 2016; Ahmad and Scheinkman, 2019; Paiton et al., 2020). The idea of low representation overlap among unrelated inputs may be particularly useful when an ANN is learning multiple, unrelated tasks. If the representations of two different tasks have near-zero overlap, it is possible to significantly reduce task interference. We explore this question in our simulations below.

3. ACTIVE DENDRITES NETWORK MODEL

Our primary goal is to augment standard ANNs with the biological properties described above. The extensions should be general and applicable to a range of complex scenarios such as multi-task RL and continual learning. The key aspects of our model are summarized as follows, with details noted in the rest of this section:

1. Pyramidal neurons integrate a range of diverse inputs on multiple independent dendritic segments. To model this, we implement neurons that separate out contextual inputs from feedforward inputs. Each neuron processes the feedforward input using a linear weighted sum. A set of independent dendritic segments process the contextual input using a separate set of weights.
2. Contextual inputs on active dendrites can modulate a neuron's response, making it more likely to fire. To model this, we implement a function that can up-modulate or down-modulate the feedforward activation based on dendritic activation.
3. Neural activity and connectivity are highly sparse. To model this, we incorporate a k-Winner-Take-All function (kWTA) that mimics biological inhibitory networks (Cui et al., 2017) and guarantees sparse activations.

The above properties are implemented such that the entire network is differentiable and trainable end-to-end using backpropagation. This makes the architecture suitable for testing on any standard deep learning scenario.

3.1. Active Dendrites Neuron

Building on the original HTM neuron model (Hawkins and Ahmad, 2016), our *Active Dendrites Neuron* [**Figure 2** (right inset)] receives two sources of input, analogous to the proximal and distal inputs in pyramidal neurons. Feedforward activation is computed by a linear weighted sum of the feedforward input vector, identical to the mechanism in a point neuron. Meanwhile, multiple dendritic segments process a context vector, and the subsequent dendritic output modulates the feedforward activation. This computation produces a neuron where the magnitude of the response to a given stimulus is highly context-dependent.

Given input vector $\boldsymbol{x}$, weights $\boldsymbol{w}$, and bias b, our neuron computes the following feedforward activation:

$$\hat{t} = \boldsymbol{w}^\top \boldsymbol{x} + b \tag{1}$$

Similarly, each dendritic segment j computes $\boldsymbol{u}_j^\top \boldsymbol{c}$, given weight $\boldsymbol{u}_j$ and context vector $\boldsymbol{c}$. (The method we use to compute the context vector, $\boldsymbol{c}$, is described in later sections.) We select the segment with the strongest response to the context when computing dendritic activation d, which is used to modulate the neuron:

$$d = \max_j \boldsymbol{u}_j^\top \boldsymbol{c} \tag{2}$$

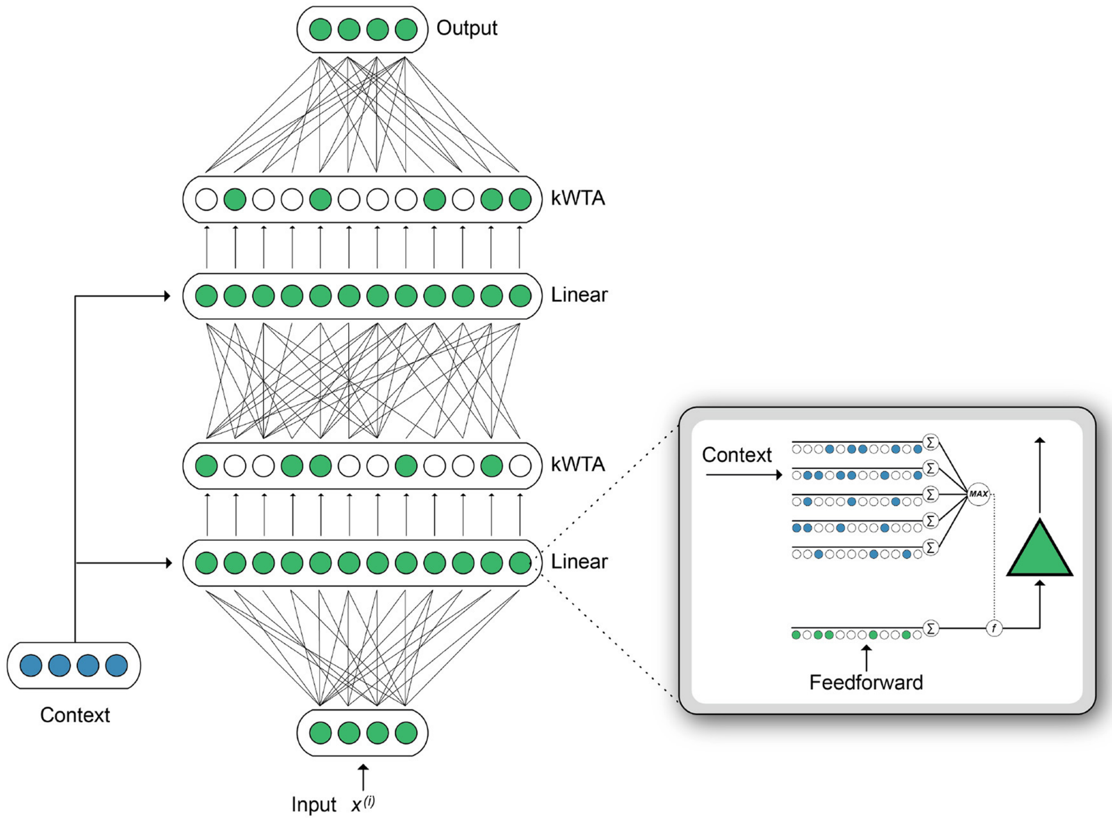

FIGURE 2 | [Right (inset)] Illustration of a single Active Dendrites Neuron. Feedforward weights (green) receive regular feedforward input while dendritic segments (blue) receive a context vector. After all dendritic segments compute an activation value, the highest value modifies the linear weighted sum computed by feedforward weights. **(Left)** An overview of the base network structure used in our experiments. There are two hidden layers. Each layer outputs sparse activations, as determined by a *k*WTA activation function. In addition, the weights between layers can be sparse. A context vector is computed for each input. The dendritic segments in each layer receive this context vector as input.

In order to modulate feedforward activation $\hat{t}$ by the dendritic activation d, we use modulation function $f(\hat{t}, d)$ where $f(m, n) = m \times \sigma(n)$. Here, $\sigma(\cdot)$ is the sigmoid function which takes a real number and maps it into the range $[0, 1]$. Therefore, by combining (1) and (2) with f, we can write the output of a single Active Dendrites Neuron as:

$$\hat{y} = f\left(\hat{t}, d\right) \tag{3}$$

$$= f\left(\boldsymbol{w}^\top \boldsymbol{x} + b, \max_j \boldsymbol{u}_j^\top \boldsymbol{c}\right) \tag{4}$$

$$= \left(\boldsymbol{w}^\top \boldsymbol{x} + b\right) \times \sigma\left(\max_j \boldsymbol{u}_j^\top \boldsymbol{c}\right) \tag{5}$$

Here, a strong positive dendrite response to the context vector will retain the feedforward activation. Conversely, weak or negative responses to the context vector will significantly reduce the activation. We note that there are many variations of (2) that are possible. We found that the network works best when we select the dendrite activation with the largest absolute value and retain the sign in d (Section 6.3).

3.2. Sparse Representations

We apply a *k*WTA activation function (Ahmad and Scheinkman, 2019) as our choice of non-linear activation in each hidden layer of the network:

$$k(\hat{y}_i) = \begin{cases} \hat{y}_i & \text{if } \hat{y}_i \text{ is one of the top } k \text{ activations over all } i \\ 0 & \text{otherwise} \end{cases} \tag{6}$$

where i indexes neurons in the same layer. The effect of *k*WTA is to ensure sparsity by selecting the top k activations and setting all others to zero. Feedforward layers that are modulated by dendritic segments and apply *k*WTA thus produce sparse activity patterns that are highly context-dependent. Additionally, our feedforward layers also use sparse weights as proposed in Ahmad and Scheinkman (2019).

3.3. Active Dendrites Network Architecture

Figure 2 (left) shows our Active Dendrites Network, trained end-to-end with backpropagation, where all neurons in each hidden layer are Active Dendrites Neurons. We make two notes: first, only the neurons that were selected by the *k*WTA function will have non-zero activations (and thus non-zero

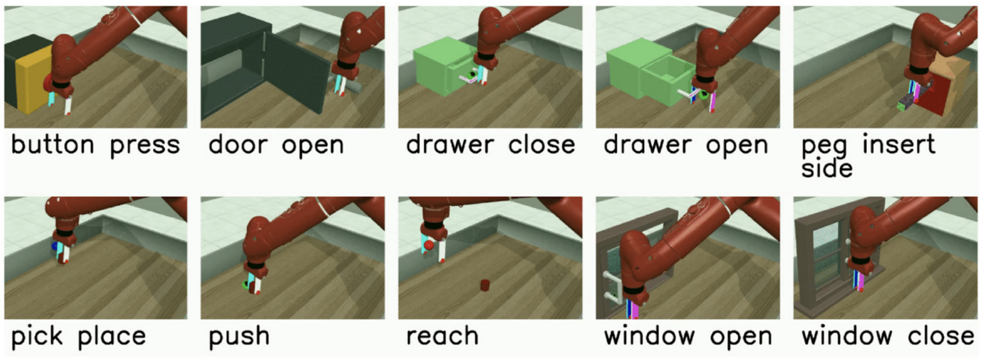

FIGURE 3 | The Meta-World v2 Multi-Task 10 (MT10) environment, where a single robotic arm must learn to solve a variety of tasks ranging in difficulty.

gradients). Therefore, during the backward pass, only the weights corresponding to those winning neurons will be updated. Second, for each of those winner neurons, only the dendritic segment j that was chosen by the max operator is updated; all other segments $\boldsymbol{u}_{j'}$ for $j' \neq j$ remain untouched. Thus a very small sparse subset of the full network is actually updated for each input.

We hypothesize that a functional specialization will emerge where different dendritic segments will each learn to identify specific context vectors. Since most dendritic segments that don't respond to a specific context will not be updated, any context-dependent modulation of the neuron should be preserved from task to task. Ideally, the whole process will preserve any context-dependent modulation of a neuron between tasks, reduce gradient interference, and prevent catastrophic forgetting.

4. RESULTS

4.1. Results With Multi-Task Reinforcement Learning

The multi-task RL problem we investigate uses the Meta-World v2 environment and its associated v2 tasks (Yu et al., 2019). Meta-World contains multiple different object manipulation tasks that a single robotic arm must learn to solve simultaneously. We use the MT10 environment, which contains 10 tasks ranging in complexity as depicted in **Figure 3**. Although the concrete outcome of each task is unique, all tasks share a common structure that enables the agent to leverage some shared information during training. For instance, learning how to grasp an object is a shared concept among many of the tasks.

The algorithm we use to train our robotic agent is multi-task Soft Actor-Critic (MTSAC) as introduced by Yu et al. (2019), an adaptation of the popular Soft Actor-Critic (SAC) framework proposed earlier by Haarnoja et al. (2018b). MTSAC is an actor-critic deep RL algorithm that maximizes an agent's cumulative reward to solve a task while also maximizing entropy to encourage environment exploration. To maintain consistency with the codebase of Yu et al. (2019) which fixes goal states (e.g., position of an object in the environment) through training, we also keep goal locations constant across all our experiments.

A deeper explanation about our multi-task RL setup and the algorithm we use to train the agent can be found in Section 6.1. As in many RL problems, there is no static training and testing dataset. Rather, past experiences from the agent are used to iteratively train the agent. We freeze the network at regular intervals to test accuracy on all tasks.

4.1.1. Network Structure for Multi-Task RL

Figure 4 shows our network architecture for multi-task RL. We use a network with 2 hidden layers—each with 2,800 neurons and followed by a kWTA activation function—and a final output layer. The first hidden layer has standard neurons whereas the second hidden layer contains Active Dendrites Neurons which are modulated by the context vector. The primary feedforward input to the network is a state vector consisting of the agent's position in the world as well as the position and orientation of the target object. The output of the network is an action vector that describes the joint torques and gripper forces of the robotic arm. The structure of the state and output vectors is identical across all tasks. All feedforward weights are sparse.

For our multi-task RL experiments, the context vector $\boldsymbol{c}$ encodes the task ID as a one-hot encoded vector. We considered other options to generate $\boldsymbol{c}$, such as first pre-processing the one-hot encoding by a linear layer, but found that a one-hot encoding was adequate. Each Active Dendrites Neuron in our network has exactly 10 dendritic segments (same as the number of tasks to learn) so that each segment can potentially learn to recognize a unique context vector.

We compare our Active Dendrites Network to baselines reconstructed[2] from Yu et al. (2019) which are multi-layer perceptrons (MLPs) with dense weights and ReLU activations. These MLP baselines are used to model both the policy and the Q function. Additionally, these MLP baselines receive context information $\boldsymbol{c}$ in the form of feedforward input concatenated to the state vector. Thus, both the Active Dendrites Network and the baseline network receive *identical* information at each time step;

[2]We are unable to directly present the published baseline results because their plots contain inconsistencies between success rates per-task and across all tasks. To present a fair comparison, we re-run the baseline networks using their codebase and hyperparameters.

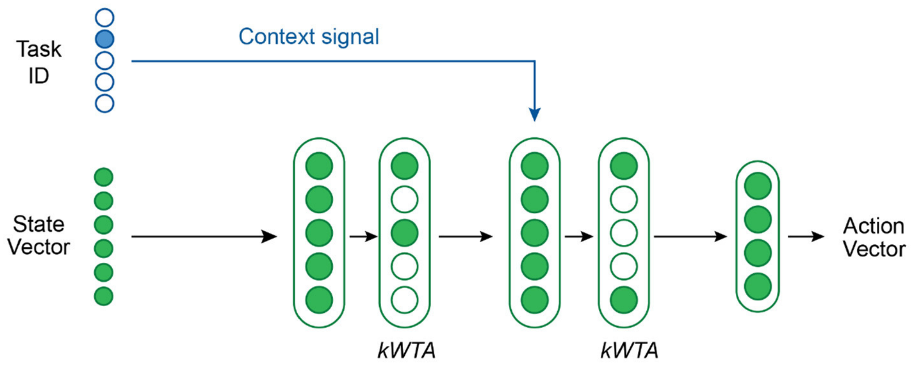

FIGURE 4 | An overview of the network structure used in our multi-task RL experiments. A *k*WTA activation is applied to both hidden layers. The context vector is the task ID. The dendritic segments in the second hidden layer receive this context vector as input.

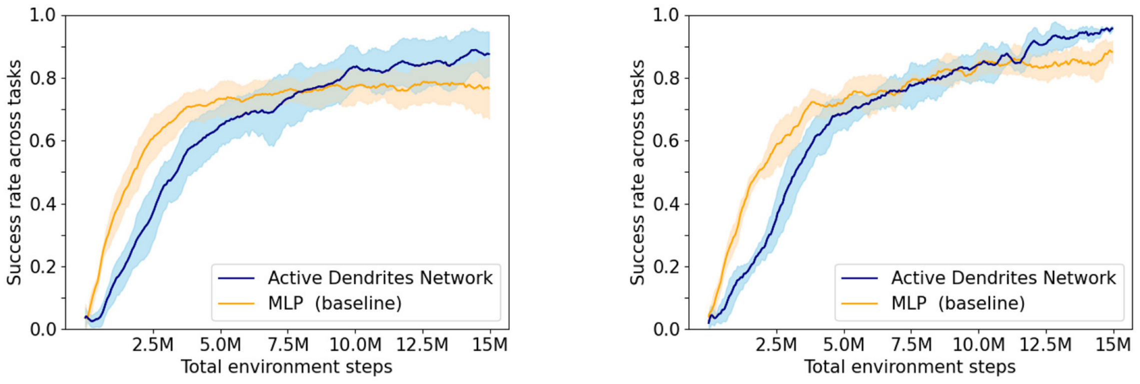

FIGURE 5 | The success rate of our network when learning 10 tasks compared to the MLP baseline with context. **(Left)** Experiment 1—the average of 10 Active Dendrites Network runs and 10 MLP baseline network runs that all share the same training hyperparameters. **(Right)** Experiment 2—the average of the five best Active Dendrites Network experiments and the five best MLP baseline experiments. The shaded region in each plot represents the standard deviation of the success rate from the average.

the primary difference between the two architectures is how the context vector is handled.

Table 2 in Section 6.1.4 shows the networks we ran, the number of *non-zero parameters* in each network, and the hyperparameters used to train each network. Although we control the hidden sizes to yield approximately the same number of total non-zero parameters across our experiments, we note that the MLP baseline network contains nearly 500,000 more non-zero parameters than our Active Dendrites Network. We chose a network with two hidden layers to draw fair comparisons with the MLP baselines presented in Yu et al. (2019). **Supplementary Materials** (Section 1) includes the results of additional experiments detailing the impact of some of our architectural choices.

4.1.2. Dendrites Improve Multi-Task RL Accuracy

We show results from two different experiments that compare our Active Dendrites Network to the MLP baseline network.

Experiment 1: In this experiment, we assess the overall performance of each architecture. We ran both an Active Dendrites Network and a MLP baseline network with identical training hyperparameters. **Figure 5** (left) shows the mean overall success rate for each architecture during the course of training over 10 independent trials. To identify which architecture performs the best for each task, we compute the mean success rate per task for the last 500,000 environment steps of training and list these values in **Table 1**. Additionally, we show the per-task training statistics during this same segment of training, as seen in **Figure 6**.

We see in **Figure 5** (left) that although the Active Dendrites Network has lower success rates early in training, it overtakes the baseline architecture and is about 10% better by the end. **Table 1** shows that the average end success rate for the Active Dendrites Network (across the last 500,000 steps of training) is **87.5%**. In comparison, the average success rate for the MLP baseline is **76.6%**. We also note that the *push*, *peg-insert-side*, and *pick-place* tasks were the hardest to solve because they are the most unlike the other tasks. Specifically, these three tasks require that a robot grasp and move a small object to a specified location. As evident in **Figure 6** for these three tasks, the median success rate of an

TABLE 1 | The mean per-task success rate produced by each network in Experiment 1.

Tasks	Model	
	Active Dendrites Network (%)	MLP baseline (%)
Drawer-close	**100.0**	**100.0**
Window-close	**99.7**	95.3
Button-press-topdown	95.7	**97.3**
Reach	**99.7**	86.3
Window-open	**99.3**	93.7
Drawer-open	**99.7**	86.0
Door-open	**94.7**	84.3
Push	**67.3**	59.0
Peg-insert-side	**71.7**	47.0
Pick-place	**47.7**	17.3
Overall	**87.5**	76.6

The success rates are averaged over the last 500,000 steps of training. The best success rate between the Active Dendrites Network and MLP baseline is highlighted in bold.

Active Dendrites Network is far greater than that of the MLP baseline network. We hypothesize that these tasks are hard to learn because of significant gradient interference with the other tasks, and that the context-specific sparsity imposed by the Active Dendrites Network helps remove this interference.

Experiment 2: The high variance in **Figure 5** (left) is inherent in many RL scenarios (Irpan, 2018; Ibarz et al., 2021). This is in large part due to the highly stochastic and dynamic training process. For instance, small variations in the trained policy can result in large variations in the agent's behavior which significantly impacts the data collected during training. Additionally, a policy can generate different behaviors during training when sampling from its predicted action distribution.

To control for some of this variation in training, for each network initialization we select the best result across different training runs. For each of five different Active Dendrites Network initializations, we ran five training runs and picked the run with the highest end success rate across the last 500,000 environment steps of training. We then compute the mean overall success rate across these five best runs. We follow the same procedure for finding the five best MLP baseline networks and compare the results in **Figure 5** (right).

We find that this process significantly reduces the variance and that the best Active Dendrites Networks still outperform the best MLP baseline networks. Across the five best Active Dendrites Network runs, the average overall end success rate is **95.6%**. In comparison, across the five best MLP baseline runs, the average overall end success rate is **88.2%**.

4.2. Results With Continual Learning

A typical continual learning problem consists of training a neural network on a discrete number of tasks in sequence. Once a network is trained on a particular task, it does not encounter that task during training again. The goal is to learn all the tasks in sequence without forgetting previously-learned tasks.

We use the permutedMNIST dataset (Goodfellow et al., 2014), a common benchmark in continual learning where each task requires classifying images of handwritten digits (0–9) after a unique pixel-wise permutation has been applied. Since the data distribution of each task changes and because neural networks are generally not permutation-invariant, forgetting occurs.

We use the original MNIST training dataset of $60,000$ images to construct the dataset for a single task. Since we train on $\mathcal{T}$ consecutive tasks, the network is trained on a total of $\mathcal{T} \times 60,000$ images. Once training is complete, the network accuracy is calculated using a test set consisting of all $\mathcal{T}$ permutations applied to the MNIST test dataset of $10,000$ images.

We train our model to learn up to 100 tasks in sequence. The network is tested at the end of training by computing accuracy on the test set for all tasks. When attempting to learn $\mathcal{T}$ consecutive tasks, the hidden neurons are equipped with $\mathcal{T}$ dendritic segments each to give it sufficient capacity to recognize a unique context vector for each task. We report accuracy numbers by averaging over 8 independent runs each with a randomly-picked seed. See Section 6.2 for additional details.

4.2.1. Computing the Context Vector

As with multi-task RL, we need to compute an appropriate context vector. For continual learning, we use a simple prototype method (Rosch, 1975; Snell et al., 2017) to select the context vector where a single vector represents each task [**Figure 7** (left)]. We implement two different variations of the prototype method depending on the knowledge available to the system during training.

4.2.1.1. Training Method 1 (Task Information Provided)

In the first method, we assume that the system receives task information during training, when all training samples for a particular task are assigned a single prototype context vector. We compute the prototype vector for task τ by taking the element-wise mean over all the training samples across all features:

$$\boldsymbol{p}_\tau = \frac{1}{|V_\tau|} \sum_{\boldsymbol{x} \in V_\tau} \boldsymbol{x}$$

where V_τ denotes the set of all data samples $\boldsymbol{x}$ that the model observes to train on task τ. The dimensionality of the context vector is thus identical to the dimensionality of the input vectors. This context vector is specific to each task and agnostic to the target label.

4.2.1.2. Training Method 2 (Task Information Not Provided)

In the second method, we relax the constraint that the identity of the task is given during training and instead implement prototypes that are automatically selected during training. To achieve this, we use a statistical clustering approach that builds context prototypes on the fly. When the system receives a new batch of training samples from a task, we use an unpaired multivariate t-test to compare the current samples to previously-observed training samples. If the new batch of samples is similar to earlier training samples, they are assigned to an existing prototype. If not, the new batch of samples is assumed to

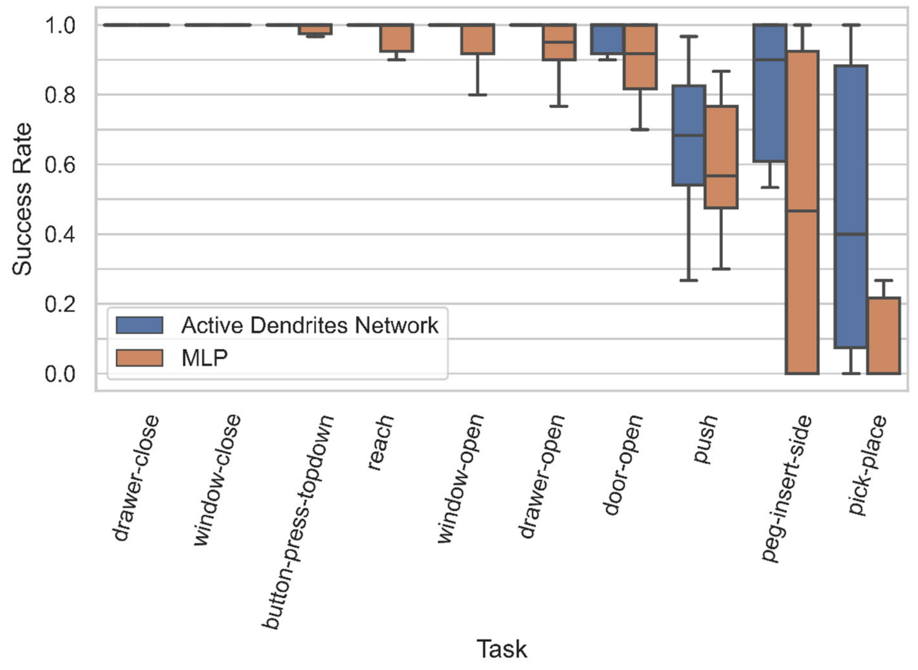

FIGURE 6 | Box plots of the accuracies for each MT10 task for our Active Dendrites Networks and MLP baseline networks in Experiment 1. We discard outliers for all runs for clarity.

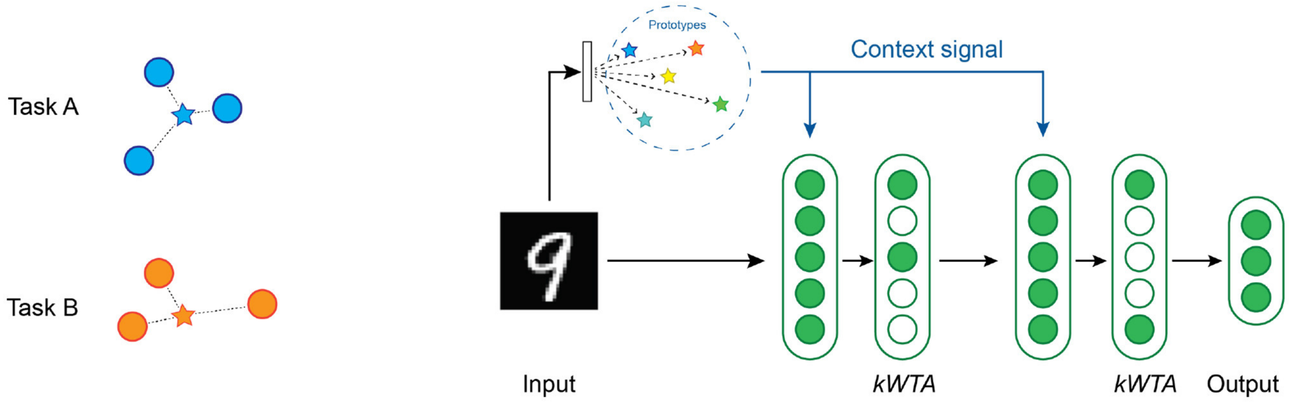

FIGURE 7 | (Left) An illustration of the prototype method for computing the context vectors. The blue circles are training samples in input space for task A, while the orange circles are training samples for task B. The blue star is a vector that represents the prototype for task A, and the orange star represents the prototype for task B. **(Right)** An overview of the network structure used in our continual learning experiments. There are two layers of hidden units, each with a *k*WTA activation function. A context vector is computed from each image by locating the nearest prototype vector.

correspond to a new task, and a novel prototype is instantiated. In this case, there isn't necessarily a one-to-one mapping between tasks and prototype context vectors. More details on this method are described in Section 6.2.3.

4.2.1.3. Selecting Prototypes During Inference

For both methods above, we do not provide any task information to the system during evaluation. Instead it must dynamically select the correct context vector and provide that to the network. We enable this dynamic approach by selecting the closest prototype vector to each test example using Euclidean distance. That is, for a test example $\boldsymbol{x}'$, the chosen prototype is:

$$\arg\min_{\boldsymbol{p}_\tau} ||\boldsymbol{x}' - \boldsymbol{p}_\tau||_2$$

computed over all prototypes $\boldsymbol{p}_\tau$ stored in memory.

4.2.2. Network Structure for Continual Learning

Figure 7 (right) shows the network that we used for our continual learning experiments. Each of the two hidden layers contain

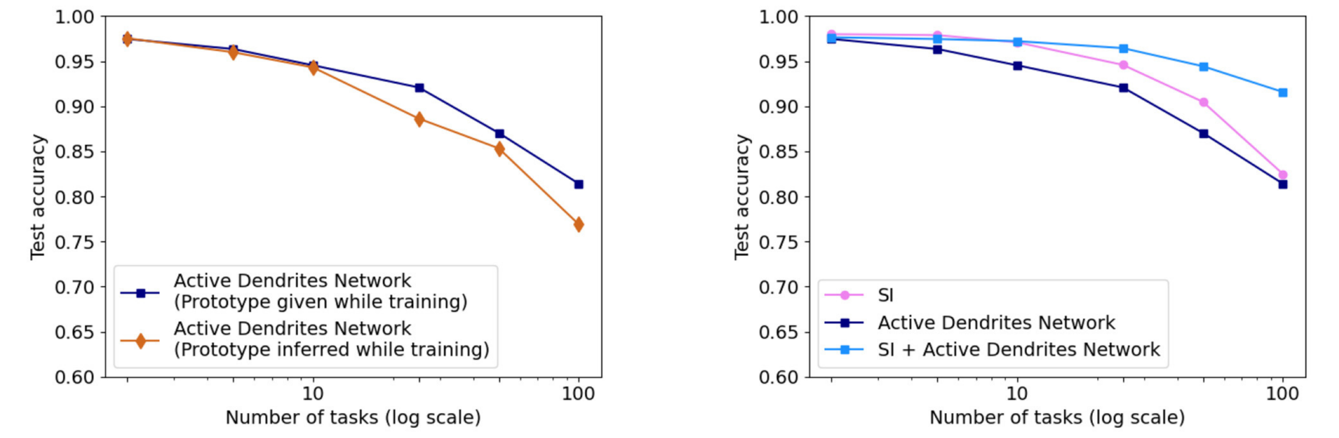

FIGURE 8 | (Left) The accuracy of our Active Dendrites Networks when learning 2, 5, 10, 25, 50, and 100 permutedMNIST tasks in sequence. We show results using both prototype methods while training: when the model is provided with a prototype, and when it must select the vector in an online manner. **(Right)** The accuracy of the Active Dendrites Network and SI. The accuracy when combining SI + active dendrites is greater than either one on its own.

2,048 Active Dendrites Neurons followed by a *k*WTA activation function. The output of the network is a standard layer with 10 neurons. We choose our network layer sizes to be similar to previous studies that report results on this dataset (Kirkpatrick et al., 2017; Zenke et al., 2017; Masse et al., 2018). (Section 6.2 details the hyperparameters used for each experiment).

4.2.3. Dendrites Mitigate Catastrophic Forgetting in Continual Learning

As shown in **Figure 8** (left), we achieve accuracies of **94.6** and **81.4%** on 10 and 100 consecutive permutedMNIST tasks, respectively, when context is provided during training, and accuracies of **94.3** and **76.9%** when context is dynamically chosen during training. Since there are always 10 categories, chance accuracy is 10% independent of the number of tasks. This demonstrates that the network successfully retains the majority of the knowledge from previous tasks. Note that a standard feedforward network performs poorly on this benchmark (Kirkpatrick et al., 2017; Zenke et al., 2017; van de Ven and Tolias, 2019; see also Section 4.3.3 for more direct comparisons).

We also compare the results with SI (Zenke et al., 2017; see Section 2.1.2). SI is inspired by the complex structure of biological synapses and known to do well on this benchmark. SI operates solely at the level of synapses: it maintains an additional parameter per weight that controls the speed of weights adapting to specific tasks. In SI, the weight updates are sprinkled throughout the network and not grouped according to units or dendrites. On the other hand, the dendrites in our network impact a small subset of the neurons, and only the weights on these neurons and dendrites are modified. As such, our two approaches seem to be complementary. **Figure 8** (right) shows the benefits of combining these two techniques. The accuracy of Active Dendrites Networks combined with SI improves to **97.2** and **91.6%** accuracy on 10 and 100 consecutive tasks, respectively. Combining the two leads to higher accuracy than either method on its own. This suggests that biological mechanisms at the synapse, neuron, and network levels can operate together to handle continual learning. Note that SI as described in Zenke et al. (2017) requires knowledge of the task during training; therefore we only combine it with our first prototype method. It may be possible to remove this restriction, which is a direction for future research.

4.2.4. Comparison With Context Dependent Gating

The idea of leveraging sparse representations and subnetworks within an ANN to combat catastrophic forgetting is not entirely novel. The implementation closest to ours is XdG (Masse et al., 2018) that uses a hard-coded distinct subnetwork for each task. When training on a task, the implementation invokes the task-specific subset of the hidden layer of the ANN; other neurons are forced to have an activation value of zero. The XdG implementation requires a task ID that determines exactly which neurons to turn on or off. Training Active Dendrites Networks in a continual learning scenario also yields subnetworks and sparse representations. However, we emphasize two major distinctions between our model and XdG:

1. Task information is inferred in our system (*via* prototyping) whereas XdG provides the system with a task ID during training and testing. As such, our system is solving a problem that is known to be significantly more challenging (van de Ven and Tolias, 2019).
2. Subnetworks automatically emerge *via* the use of dendritic segments for each new task whereas XdG pre-allocates a different subnetwork for each task, which also indicates our system is solving a more challenging problem.

We compare Active Dendrites Networks to XdG in **Figure 9**. Just as we augment Active Dendrites Networks with SI, so too does XdG. Our results with a large number of tasks are significantly better than XdG, and slightly worse than XdG combined with SI, but without their limitations.

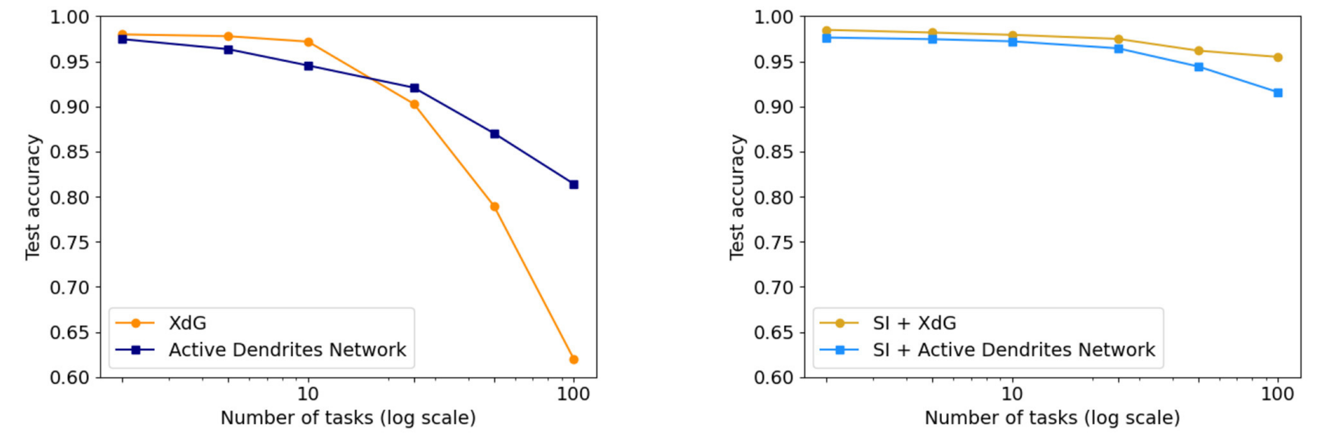

FIGURE 9 | (Left) Final accuracy of the Active Dendrites Network in comparison to XdG when learning 2, 5, 10, 25, 50, and 100 permutedMNIST tasks. The more tasks learned by the system, the greater the accuracy of the Active Dendrites Network. **(Right)** Final accuracy of each method when augmented with SI, and SI itself. XdG results are taken from Masse et al. (2018).

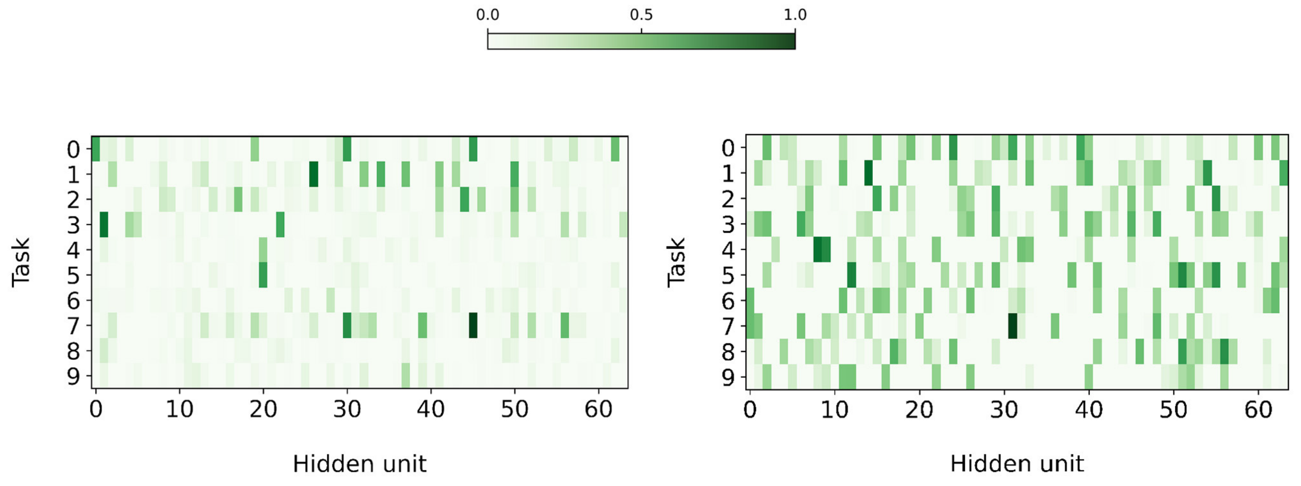

FIGURE 10 | The fraction of instances for which each of the first 64 hidden units in the hidden layer became active (after applying *k*WTA), when training an Active Dendrites Network on MT10 tasks **(Left)** and 10 permutedMNIST continual learning tasks **(Right)**. Both figures separate instances by task. For MT10, the figure tests the trained RL policy on each task three times during evaluation. For permutedMNIST, the figure uses 5,000 randomly-chosen test examples across all tasks. Note that each hidden layer contains more than 2,000 hidden units, but we show just 64 for ease of visualization.

Learning is more challenging in our system as dendritic segments must learn the mapping between context vectors and different subnetworks. In effect, sparse representations and minimally overlapping subnetworks emerge organically in our model. We note that perhaps this makes learning more effective as dendritic segments can choose subnetworks that overlap more for tasks that are more semantically related, thus requiring less network capacity.

4.3. Analysis

4.3.1. Are Dendrites Invoking Subnetworks?

The hypotheses of our work are two-fold. First, Active Dendrites Networks modulate an *individual* neuron's activations for each task. Second, *k*WTA activations use this modulation to activate subnetworks that correspond to each task. To test these hypotheses, we train and analyze an Active Dendrites Network for 10 tasks in multi-task RL and continual learning scenarios and investigate the representations of a layer of neurons modulated by dendritic segments.

Figure 10 shows the average activation frequency per task (after applying *k*WTA) for the first 64 neurons in the second hidden layer for both multi-task RL and continual learning. Looking horizontally across the rows, each task appears to select a different sparse subset of neurons. Looking vertically across the columns, each neuron appears to activate frequently only for a small fraction of tasks. According to this measure, it appears that the network has indeed learned to invoke minimally overlapping subnetworks for different tasks.

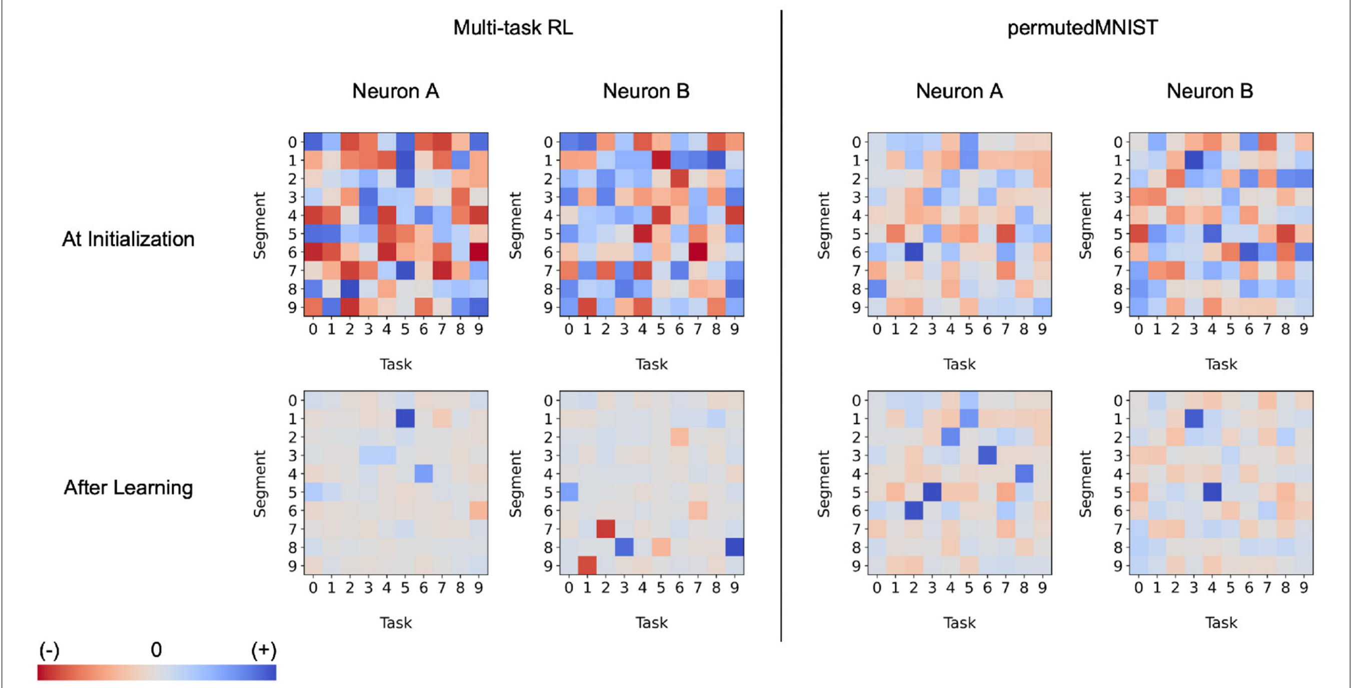

FIGURE 11 | The behavior of the dendritic segments of two separate neurons in a hidden layer of an Active Dendrites Network during three random evaluations of each MT10 task and 5,000 random evaluations of each permutedMNIST task. These charts show the activation computed by each dendritic segment given the context vector corresponding to each task, before **(Top)** and after **(Bottom)** training. Note that the dendritic segments for a particular neuron are completely separate of the segments of another in both multi-task RL and continual learning scenarios (e.g., Neuron A's first segment is unrelated to Neuron B's first segment).

What is the effect of dendrites on a single neuron? In **Figure 11**, we analyze a few Active Dendrites Neurons and their responses to different context vectors before and after learning 10 multi-task RL and permutedMNIST tasks in sequence. At the beginning of training, the responses are random with scattered positive, negative, and near-zero responses. After training, most responses are weak and only a few are either strongly positive or negative. Notably, across the neurons, dendrites only have strong responses to a few contexts as different neurons participate in different subnetworks. We note that in the multi-task RL scenario, we observe both strong positive and negative responses while the continual learning scenario only shows strong positive activity. We are unclear as to why this particular behavior emerges in continual learning but not multi-task RL.

4.3.2. Impact of Sparsity Level and the Number of Dendrites

We show that an Active Dendrites Network is competitive with benchmarks in both multi-task RL and continual learning. However, to what extent are active dendrites and sparse representations both contributing factors toward alleviating catastrophic forgetting?

We investigate this question in the context of continual learning. We find that both active dendrites without sparse representations and standard point neurons with sparse representations are better than chance in a continual learning scenario. However, the combination of both active dendrites and sparse representations yield significantly better results than either one on its own. As **Figure 12** (left) shows, the accuracy of both methods evaluated independently and evaluated together on 10 and 100 permutedMNIST tasks demonstrates the importance of implementing both active dendrites and sparse representations.

To further test the impact of dendrites and sparsity, we run two additional tests in the continual learning scenario. First, we fix the level of sparsity in our hidden representations and vary the number of dendritic segments per hidden neuron. Second, we fix the number of dendritic segments per hidden neuron and vary the sparsity in our hidden representations (i.e., vary *k* in *k*WTA). As seen in **Figure 13** (left), increasing the number of dendritic segments leads to a small monotonic increase in accuracy. **Figure 13** (right) shows that reducing sparsity translates to a sharp drop in accuracy, further highlighting the need for sparse representations.

4.3.3. Are Networks With Dendrites Equivalent to Larger Networks?

Over the last couple of decades, multiple studies have suggested that dendritic computations performed by pyramidal neurons can be approximated by ANNs that have one or more hidden layers. For example, Poirazi et al. (2003) shows that a larger two-layer neural network can well-approximate the post-synaptic responses of a pyramidal neuron with active dendrites. Various follow-up studies also make similar claims (Jadi et al., 2014; Beniaguev et al., 2021), with Beniaguev et al. (2021) suggesting

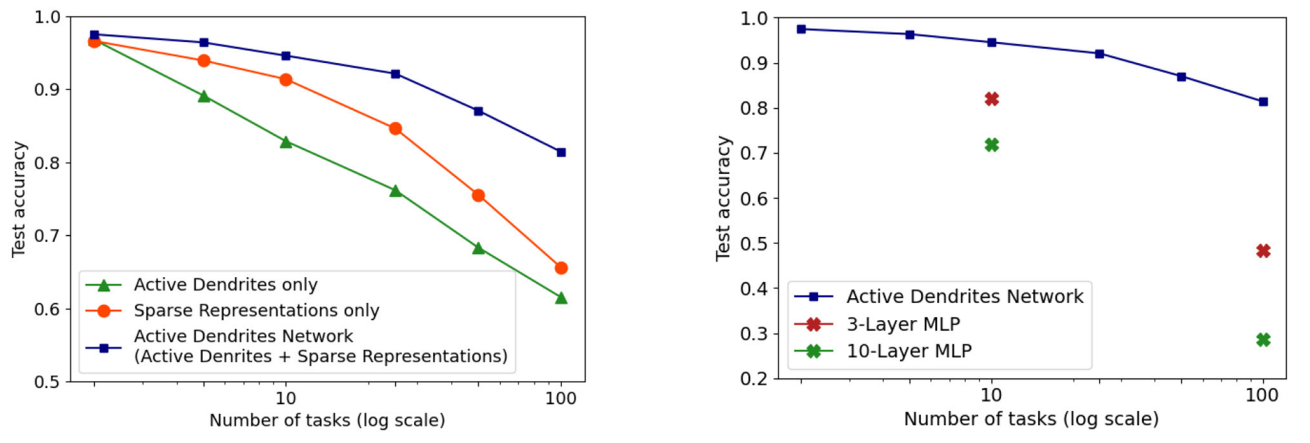

FIGURE 12 | (Left) Continual learning test accuracy on permutedMNIST using active dendrites and dense representations (green), regular ANNs with sparse representations (orange), and Active Dendrites Networks (blue) which use both active dendrites and sparse representations. **(Right)** Continual learning test accuracy for our Active Dendrites Network compared to regular feedforward networks with more layers. Our Active Dendrites Network has three layers; the two hidden layers contain neurons modulated by dendritic segments. In all experiments (left and right subfigures), we average results over 8 independent runs, each with a randomly initialized seed, and omit standard error bars as they highlight a very small range.

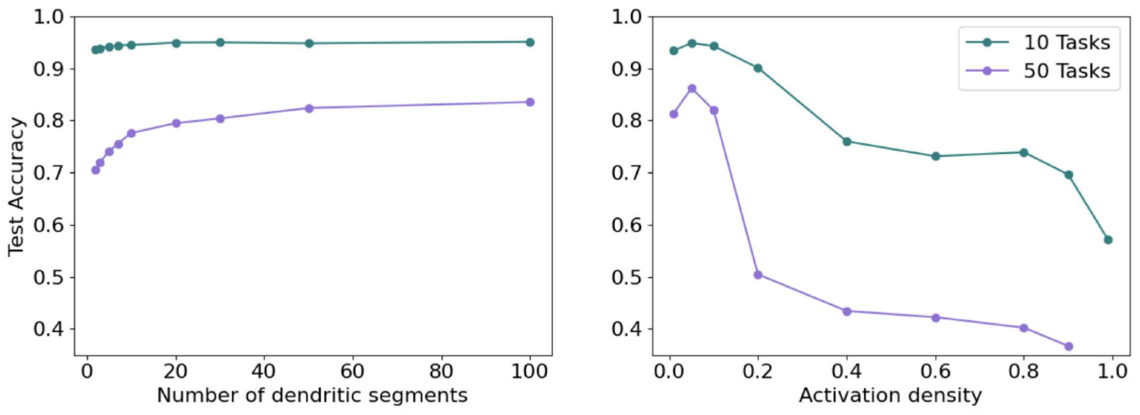

FIGURE 13 | (Left) Final accuracy on test examples across all tasks when varying the number of dendritic segments per neuron and keeping activation sparsity constant when learning 10 (top) and 50 (bottom) permutedMNIST tasks. **(Right)** Final accuracy on test examples across all tasks for a fixed number of dendritic segments per neuron and varying activation density level on 10 (top) and 50 (bottom) permutedMNIST tasks.

that a pyramidal neuron is equivalent to a larger ANN with seven hidden layers. In this section we show that in the dynamic scenarios considered here, an Active Dendrites Network is not equivalent to larger or deeper ANNs.

In the case of multi-task RL, a pyramidal neuron's activity *cannot* be approximated by a neural network with more parameters. For instance, classical deep networks that are trained on a variety of tasks are incapable of performing well due to gradient interference, an issue that cannot be solved with simply more hidden neurons. When comparing a three-layer Active Dendrites Network and a three-layer MLP with 500,000 more learnable, non-zero parameters, **Figure 5** shows that networks with dendrites and sparse representations far outperform the MLP baseline. We also experiment with larger 3-layer MLPs that have 1,700,000 more non-zero parameters than our Active Dendrites Network (hyperparameters found in **Table 2** of Section 6.1.4). In this case, we find that the MLP produces a success rate of **73.1%** across 10 tasks (averaged over the last 500,000 environment steps of training), which underperforms our Active Dendrites Network yielding an average success rate of **87.5%**.

In addition, in the continual learning setting, our network with dendrites cannot be approximated by a neural network with multiple layers. When considering continual learning, classical deep networks are incapable of performing well due to catastrophic forgetting, regardless of network depth. This specific trend can be observed in **Figure 12** where our Active Dendrites Network outperforms standard MLPs that have (a) the same number of layers but no dendrites (for 10 and 100 permutedMNIST tasks), and (b) many more layers and roughly the same number of learnable parameters (for 10 permutedMNIST tasks). (Other ablation studies, not shown in **Figure 12**, are described in the **Supplementary Materials** section.)

These results for both multi-task RL and continual learning suggest that standard ANNs that are wider or deeper are still prone to gradient interference and catastrophic forgetting while active dendrites can help retain knowledge from previous tasks. In these dynamic settings, our experiments show that a standard feedforward network with more hidden units or additional layers is not as powerful as a network with active dendrites.

5. DISCUSSION

The exact mechanistic details of how a biological neuron converts incoming signals into action potentials (i.e., spikes) remain unclear. Ever since Rosenblatt (1958), models of biological neurons favor a single linear weighted sum (the point neuron) as a tractable abstraction. This idea continues to serve as the prevalent paradigm in machine learning today for the individual computational unit. One shortcoming is that standard ANNs with point neurons can suffer from catastrophic forgetting. They overwrite many of their connections for each learning iteration, and thus quickly lose previously-acquired knowledge (French, 1999; Parisi et al., 2019).

In this article we show that augmenting point neurons with biological properties such as active dendrites and sparse representations significantly improves a network's ability to learn multiple tasks at once. In the multi-task RL setting, a three-layer Active Dendrites Network can achieve an average accuracy of about **88%** when learning 10 Meta-World tasks together. In the continual learning setting, an almost identical network can achieve greater than **90%** accuracy when learning 100 permutedMNIST tasks in sequence. These results, on two very different scenarios, suggest that Active Dendrites Networks may represent a general purpose architecture for avoiding interference and forgetting in complex settings. In the rest of this Discussion we elaborate on this idea and describe some relationships to other research.

5.1. Dendrites Enable Dynamic Context Integration and Routing

In this section, we attempt to elucidate how active dendrites help in dynamic scenarios such as multi-task and continual learning, and discuss our theory of their underlying role in the neocortex. Following experimental evidence (Section 2.2.1), our model suggests that dendritic segments in each neuron identify specific contexts and then modulate neuronal activity based on this identification. Combined with subsequent local inhibition (kWTA function), the modulation can impact whether the neuron activates.

We propose that the consequence of this behavior is to invoke sparse context-specific subsets of the network. Two different context vectors can lead to different winners and different sparse activation patterns (illustrated in **Figure 14**). As suggested by the figure, the same feedforward input can activate completely different neurons based on the specific context. Note that the subnetworks are distributed and that two different subnetworks may share some neurons. In Section 4.3.1, we showed that task-specific representations do indeed emerge in our experiments (**Figure 10**).

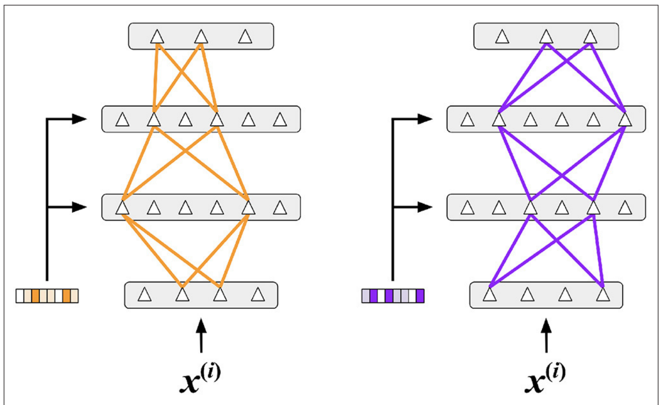

FIGURE 14 | A representation of subnetworks within an Active Dendrites Network. By receiving different context vectors as input, dendritic segments invoke different subnetworks for a fixed feedforward input. The subnetworks are distributed, i.e., they may share some of the same neurons.

Why do subnetworks help? In dynamic conditions, the system must react and learn in constantly changing situations. Subnetworks restrict the flow of information to be highly context-dependent and relevant to each specific situation. In addition, errors will only propagate through the active subnetwork. Only the active neurons will update their feedforward weights and only the winning segment within those active neurons will update their dendritic weights. Thus, by utilizing context the brain can isolate information flow, and direct learning itself in a highly localized and task specific manner. The last two decades have seen significant experimental support for highly localized task specific learning in the dendrites of pyramidal neurons (Losonczy et al., 2008; Yang et al., 2014; Kerlin et al., 2019; Limbacher and Legenstein, 2020).

What is the role of context? In this article we have used a context vector that represents the current task. Prior experimental and modeling work shows the utility of various other types of context. In recurrent networks, it is possible to use the previous activity of the network as context for dendrites. In this case a layer of neurons becomes a powerful sequence memory system (Hawkins and Ahmad, 2016). For sensorimotor inference, if the coordinates of an external reference frame is used as context, neurons can perform object recognition with actively moving sensors and by integrating information over time (Hawkins et al., 2017). Schmidt-Hieber et al. (2017) and Heald et al. (2021) also provide experimental evidence for the role of dendrites in separating out information in continuous sensorimotor streams. In the neocortex, if the inference results of neighboring cortical areas are used as context, dendrites can be used to disambiguate uncertain information and perform voting (Hawkins et al., 2017).

In each of the above scenarios, although the nature of the context greatly impacts emerging behavior, the fundamental operations remain the same. Dendrites recognize patterns that best match their synapses, and up-modulate their neurons such that they are more likely to win. This in turn invokes context specific subnetworks that route information flow and gate learning in order to effectively learn and perform the task at hand. Contextual routing mediated by dendrites may thus be a general-purpose and powerful capability that underlies much of cognitive function (Phillips, 2015; Phillips et al., 2015). Indeed, the ability to generate context-dependent output based on a common set of operations could be a crucial building block of cognitive maps able to cover any domain (Whittington et al., 2022). Flesch et al. (2022) provide experimental evidence for contextual gating in a study of human continual learning and memory.

In our implementation we have focused primarily on feed-forward information flow and basal dendrites, and have ignored recurrent and feedback connections and apical dendrites (Larkum et al., 1999; Larkum, 2022). Interestingly, lateral connections and feedback connections seem to segregate onto different dendritic integration zones (Guest et al., 2021; Lafourcade et al., 2022). Recent experimental evidence suggests that apical dendrites also process feedback context and have a modulatory impact on the cell leading to task specific functionality (Kerlin et al., 2019; Takahashi et al., 2020; Schoenfeld et al., 2022). From a modeling perspective, there is additional complexity related to generating top-down context (Siegel et al., 2000) and simultaneously processing three separate input streams (Phillips, 2015; Larkum, 2022), an interesting area for future research.

In this article, we have focused on modeling the dendritic properties of pyramidal neurons, but we note that dendritic modulation and gating may occur with other neuron types. For example, thalamocortical neurons may exhibit analogous dendrite initiated gating properties (Errington and Connelly, 2011). As such, dendrite mediated contextual integration and gating may be a more general phenomenon of biological neural systems. Modeling other neuron types is an interesting area for future work.

5.2. Comparison to Other Multi-Task RL Systems

Many techniques in multi-task RL make manual changes to the network structure or learning scheme in order to account for the learning of new tasks. In multi-task scenarios, optimizers struggle to learn different tasks that vary in gradient magnitude and have conflicting gradient direction. In these cases, tasks with larger magnitudes are usually preferred during optimization over others. To rectify this issue, Yu et al. (2020) minimizes gradient interference by orthogonally projecting the gradients of tasks that conflict with each other. Additionally, in most scenarios, a policy trained on a specific task with a specific agent cannot be adapted to similar problem settings. Devin et al. (2017) proposes a framework to learn separate policy modules corresponding to a particular task or robotic agent. Ultimately, they show how these modules can be mixed to perform new task–agent combinations or serve as a starting point for good initializations when learning complex behaviors. Many multi-task problems also highlight the issue of parameter sharing between distinct tasks. To that end, Yang et al. (2020) introduces a base policy network composed of multiple modules and a separate routing network. The routing network uses a task embedding and the current state of the agent to reconfigure the base network's modules with a learned routing strategy.

In contrast, our Active Dendrites Network activates sparse subnetworks by introducing control over individual neurons in a network. By dynamically integrating a context vector to modulate these neurons, the network automatically creates distinct subnetworks to learn each task. Unlike prior approaches, our network does not require modified learning rules, separate modules, or dedicated routing networks to train new tasks. Rather, a single architecture is capable of reducing gradient interference, learning a diverse range of tasks, and can be applied to scenarios beyond multi-task RL.

5.3. Comparison to Other Continual Learning Systems

There are a few papers on continual learning that are very related to the core ideas in this paper. Our networks create representations composed of different sparse subnetworks of neurons. Abbasi et al. (2022) use *k*WTA in conjunction with a modified gradient update method to avoid task interference. XdG (Masse et al., 2018) and Supermasks (Wortsman et al., 2020) also explicitly utilize sparse subnetworks per task. XdG, discussed extensively in Section 4.2.4, hard-codes a sparse subnetwork for each task. This extra supervision step removes the need to dynamically gate activations but requires knowledge of the task identity during inference. In addition, as seen in **Figure 9**, XdG does not scale as well as our networks. In contrast, Supermasks uses a randomly initialized network and focuses on locating the best subnetwork for each task and forgoes any further training. The technique shows impressive scaling behavior, but it's unclear whether complex tasks can be solved without any network training.

Our Active Dendrites Neurons dynamically determine a representation for each feedforward input based on auxiliary contextual inputs. In the case where the modulation function f involves multiplication, our Active Dendrites Networks are an instance of multiplicative networks. Jayakumar et al. (2020) demonstrated that multiplicative networks can excel in multi-task scenarios by learning dynamic representations in a task-specific manner.

Several ANN-based techniques leverage the idea of auxiliary contextual inputs. For instance, Gated Linear Networks (Veness et al., 2021) and Dendritic Gated Networks (Sezener et al., 2021) gate activation values for each neuron based on contextual information. Although inspired by dendrites these models (1) don't activate sparse subnetworks, (2) have fixed random dendritic weights (to model cerebellar dendritic branches), and (3) are binary classifiers (i.e., 10 Dendritic Gated Networks are required to classify MNIST digits). Furthermore, because Sezener et al. (2021) test Dendritic Gated Networks only up to 10 permutedMNIST tasks using a very different metric, we cannot provide a direct comparison with our model.

5.4. Future Work

Our initial results show that active dendrites and sparse representations can mitigate catastrophic forgetting and interference in multi-task RL and continual learning settings. One crucial next step is to test this framework on more real-world scenarios with greater complexity than MT10 or permutedMNIST. The majority of existing work in MTRL considers tasks with shared input and action spaces. Dendrites may be beneficial in scenarios where this assumption does not hold. Extending to tasks with very different input and output spaces is an interesting area for future research. Another interesting area is to combine our two scenarios and explore continual multi-task RL. While testing on more diverse benchmarks, it will also be important to explore additional methods for generating context vectors for a given task. Another important direction for future research is to investigate sparse dendritic segments, following neuroscience evidence suggesting that each segment relies on just a handful of synapses (Branco and Häusser, 2011).

6. METHODS

6.1. Multi-Task Reinforcement Learning Experiments

In this section, we provide the details of our multi-task RL experiments[3]. We use the Multi-Task Soft Actor-Critic algorithm (MTSAC) originally discussed in Yu et al. (2019), which is described as an adaptation of the Soft Actor-Critic algorithm (SAC; Haarnoja et al., 2018b). We adapt the code in the original Meta-World GitHub repository[4] to fit our experiments.

6.1.1. Basics of Reinforcement Learning

To formalize our specific RL problem, we define some fundamental concepts. The state of the RL agent and the action it will take at a specific time t are denoted as s_t and a_t, respectively. The RL algorithm trains a policy π to take a_t given s_t in order to maximize total return $G = \sum_t \gamma^t r(a_t, s_t)$ across all time-steps t, where $r(a_t, s_t)$ is the reward given by the environment and γ is a discount factor to strongly consider immediate rewards.

To optimize this policy, our RL formulation uses Markov Decision Processes (MDPs) to model decision making in stochastic environments. Following the notation introduced in Sutton and Barto (2018), we consider a finite-horizon MDP defined by the tuple (S, A, P, r, T) that operates in a state space S and action space A. The MDP also uses the transition probability P between any two states s_t and s_{t+1} by taking action a_t, which is explicitly defined across all states and actions as $P(s_{t+1}|s_t, a_t) : S \times A \to \mathbb{R}$. Agents in this setting receive a reward $r : S \times A \to \mathbb{R}$ that is also defined across all states and actions. Additionally, agents must make decisions within a fixed number of steps, denoted by the finite-time horizon T.

The RL algorithm we consider computes a value function that estimates the total return accrued at a specific state. More precisely, the value function describes the significance of starting at some state s_t and following some policy π. The value function for policy π can be defined below:

$$V_\pi(s_t) = \mathbb{E}_{a_t \sim \pi}\left[r(s_t, a_t) + \mathbb{E}_{s_{t+1} \sim P}\left[\gamma V_\pi(s_{t+1})\right]\right] \tag{7}$$

$$= \mathbb{E}_{a_t \sim \pi}\left[r(s_t, a_t) + \sum_{s_{t+1} \in S} P(s_{t+1}|s_t, a_t)\left(\gamma V_\pi(s_{t+1})\right)\right] \tag{8}$$

$$= \sum_{a_t \in A} \pi(a_t|s_t)\left[r(s_t, a_t) + \sum_{s_{t+1} \in S} P(s_{t+1}|s_t, a_t)\left(\gamma V_\pi(s_{t+1})\right)\right] \tag{9}$$

Note that Equation (7) establishes a recursive relation with respect to the function V_π. To estimate the value at a given state s_t, an agent must take an action a_t sampled from policy π to calculate the expected value at the next state s_{t+1}. By repeating this process until a terminal state is reached, the agent can use the value function to choose actions that lead to highly valued states.

The RL algorithm we consider also estimates an action-value function Q_π. While value functions estimate the value of starting at s_t and following π, action-value functions estimate the value of starting at s_t, taking action a_t, and *then* following π until a terminal state is reached. This is known as the Q function, which can be described explicitly below:

$$Q_\pi(s_t, a_t) = r(s_t, a_t) + \mathbb{E}_{s_{t+1} \sim P}\left[\gamma \mathbb{E}_{a_{t+1} \sim \pi}\left[Q_\pi(s_{t+1}, a_{t+1})\right]\right] \tag{10}$$

Fundamentally, value functions and Q functions can be related by the following two expressions:

$$Q_\pi(s_t, a_t) = r(s_t, a_t) + \mathbb{E}_{s_{t+1} \sim P}\left[\gamma V_\pi(s_{t+1})\right] \tag{11}$$

$$V_\pi(s_t) = \mathbb{E}_{a_t \sim \pi}\left[Q_\pi(s_t, a_t)\right] \tag{12}$$

Throughout the training process, explored state, action, reward, and next state transitions–namely (s_t, a_t, r_t, s_{t+1})–are used to train π. In some algorithms, including ours, these transitions are stored in a replay buffer $\mathcal{D}$ and are sampled by batch during each step of training to dynamically compute either the value or Q function. After a suitable period of exploration, the agent will take actions that yield the maximum $V_\pi(s_t)$ or $Q_\pi(s_t, a_t)$ value. Note that while V, Q, and π are expressed in discrete state and action spaces above, they can be easily extended to work in continuous state and action spaces using function approximations such as neural networks.

6.1.2. Basics of Multi-Task Reinforcement Learning

We can extend the ideas in Section 6.1.1 to our multi-task RL experiments in Meta-World. Specifically, the problem framework uses a separate MDP to model each task τ. In the context of the Meta-World multi-task environment, each task shares

[3]PyTorch source code for our experiments is available at https://github.com/numenta/htmpapers.

[4]Meta-World source code is available at https://github.com/rlworkgroup/metaworld.

TABLE 2 | The hyperparameters for each multi-task RL model.

	Active dendrites network	MLP baseline	Large MLP
Network hyperparameters			
Feedforward input size	39	49	49
Hidden sizes	2 × [2,800]	2 × [2,800]	2 × [3,000]
Output size	4	4	4
Feedforward weight sparsity	10%	0%	0%
Activation function	*k*WTA	ReLU	ReLU
Activation sparsity	25%	∼ 50%	∼ 50%
Num. dendritic segments per neuron	10	0	0
Num. weights per dendritic segment	10	—	—
Num. hidden layers modulated	1	—	—
Dendritic segment weight sparsity	0%	—	—
Non-zero feedforward parameters	7,169,964	7,994,004	9,165,004
Non-zero dendritic parameters	280,000	0	0
Non-zero parameters (total)	7,449,964	7,994,004	9,165,004
Training hyperparameters			
Number of epochs	3,000	3,000	3,000
Number of timesteps	15,000,000	15,000,000	15,000,000
Number of gradient steps per epoch	250	250	250
Buffer sampling batch size	2,560	2,560	2,560
Replay buffer size	1,000,000	1,000,000	1,000,000
Policy learning rate	3×10^{-4}	3×10^{-4}	3×10^{-4}
Q-function learning rate	3×10^{-4}	3×10^{-4}	3×10^{-4}
Target Q-function update rate	5×10^{-3}	5×10^{-3}	5×10^{-3}
Policy (Min, Max) Std	e^{-20}, e^{2}	e^{-20}, e^{2}	e^{-20}, e^{2}
Action sampling distribution type	Tanh normal	Tanh normal	Tanh normal

identical state and actions spaces and defines common transition probabilities and time horizons. However, each task defines separate reward functions, although all functions share a similar scale and structure to allow a single agent to uniformly learn all tasks. We assume a uniform distribution of tasks $p(\tau)$ and train a task-conditioned, stochastic policy $\pi(a|s, \boldsymbol{c})$ to solve all $\mathcal{T}$ tasks, where $\boldsymbol{c}$ is a context vector that provides information about a specific task. Explicitly, the policy is trained to maximize the total return from the task distribution $p(\tau)$ as expressed by $\mathbb{E}_{\tau\sim p(\tau)}\left[E_{\pi}\left[\Sigma_{t=0}^{T}\gamma^{t}r_{t}(s_t, a_t)\right]\right]$.

6.1.3. The Multi-Task Soft-Actor Critic Algorithm

The MTSAC algorithm we use in our experiments is based on the SAC algorithm and slightly modified to solve τ various tasks simultaneously. In SAC, an RL algorithm uses a V or Q network (known as the critic) to train a policy π (known as the actor) to take better actions. SAC modifies the original value function definition to also consider the entropy of the policy π. By maximizing *both* expected return and entropy, an agent is motivated to explore new states while computing an optimal policy. More details about the SAC algorithm can be found in Haarnoja et al. (2018b).

In MTSAC, both π and Q are conditioned by context vector $\boldsymbol{c}$ and are thus denoted as $\pi(a_t|s_t, \boldsymbol{c})$ and $Q(s_t, a_t|\boldsymbol{c})$, respectively. MTSAC also uses τ different entropy coefficients α_τ to control the exploration per task. More details about the MTSAC algorithm can be found in Yu et al. (2019).

The Meta-World environment we use is MT10, which contains 10 different tasks that a single robotic arm must solve. All tasks share an identical state space $s_t \in \mathbb{R}^{39}$ and action space $a_t \in \mathbb{R}^{4}$. Because there are 10 different tasks, $\boldsymbol{c}$ is a 10-dimensional one-hot encoded vector that describes the task ID.

6.1.4. Experiment Settings

The training hyperparameters are identical for the Active Dendrites Network and the MLP baseline. For every run, the model is trained for 3,000 epochs. Each epoch comprises of one episode of 500 timesteps for each of the 10 tasks. In total, this amounts to 5,000 timesteps per epoch and 15,000,000 timesteps for the entire run. Our implementation parallels the baseline implementation. In our experiments, we use one Active Dendrites Network to model the policy π and another to model the Q function.

The model is also used to collect new data to be stored in a replay buffer. At the end of each epoch, the model is then trained for 250 gradient steps. For each gradient step, the algorithm randomly samples a batch of 2,560 experiences from the replay buffer. The replay buffer is a queue of limited size, capped at 1 million, with newer experiences replacing older experiences.

To allow a better comparison between the models, we set the learning rates, target Q function update rate, and policy minimum and maximum standard deviations to be the same for all runs. The difference between the models is the network architecture of the policy and Q functions. The Active Dendrites Network modulates each neuron in the second hidden layer with 10 dendritic segments, where each segment is a vector of size 10. In total, this dendritic layer adds an additional 280,000 parameters to the overall network. We apply a fixed sparsity mask of 10% to the weights of the feedforward layers of the Active Dendrites Network to reduce the number of free parameters and keep it of comparable size to the MLP baseline. The Active Dendrites Network also uses a kWTA activation function instead of ReLU, which effectively selects the top 25% of units and zeroes out the remaining during every forward step.

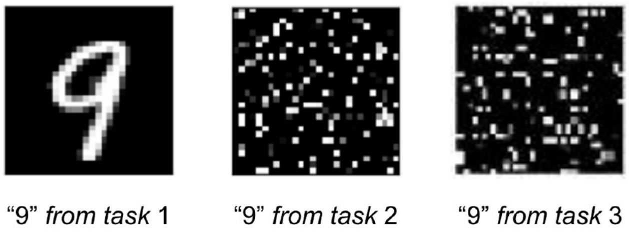

FIGURE 15 | A visual illustration of permutedMNIST. Each task applies a unique pixel-wise permutation to the same original image (leftmost image) while preserving the target label. A model's task is to identify the digit in each case regardless of permutation.

6.2. Continual Learning Experiments

We discuss the setup of the continual learning experiments. Our model is trained on $\mathcal{T}$ discrete tasks in sequence. More specifically, our model first trains on task $\tau = 1$. Once learning task τ is complete, the model then starts training on task $\tau + 1$. After training on task $\tau = \mathcal{T}$, all learning is complete. Each task τ consists of standard batch learning with i.i.d. training data. While training on task τ where $1 \leq \tau \leq \mathcal{T}$, our model only receives training data corresponding to task τ. Once the model finishes learning task τ, it never again receives information about any task $\tau' \leq \tau$ for training purposes. The model is, however, evaluated on the test data for each task to determine how well it performs.

6.2.1. PermutedMNIST

We train our model on the permutedMNIST dataset, a benchmark dataset for continual learning (Goodfellow et al., 2014), which is derived from MNIST. MNIST comprises approximately 60,000 black and white images of handwritten digits 0–9 where each such image has dimensions 28×28 pixels and the associated target digit as the label. During training, roughly 50,000 images are used for training and the remaining 10,000 for testing.

In permutedMNIST with $\mathcal{T}$ tasks, MNIST is replicated $\mathcal{T}$ times, but each time with a unique pixel-wise permutation applied to all 60,000 images. That is, each task randomly re-arranges the pixels of all images exactly the same way while preserving the associated target label. The first task ($\tau = 1$) corresponds to the identity permutation (i.e., regular MNIST) and every subsequent task generates a random pixel-wise permutation. As permutedMNIST is synthesized from regular MNIST, there can be an arbitrary number of tasks, $\mathcal{T}$. **Figure 15** illustrates a single image taken from different tasks.

Our model, and all comparisons we made, uses a single output head. Each model has 10 output units in the final layer of the network representing the 10 categories. These output units are re-used for each task, i.e., the model is trained to predict the first output unit for label "0" regardless of which task the input data corresponds to. In this setup chance accuracy is 10%.

6.2.2. Experiment Settings

When employing the prototype method described in Section 4.2.1 to select context signals at test time only, we train an Active Dendrites Network with two hidden layers that comprise Active Dendrites Neurons. We find that having just a single hidden layer reduced accuracy by a few percentage points while 3 hidden layers provided a minimal performance boost. For 100 tasks, a single layer reduced accuracy by 3% and three layers improved accuracy by 0.5%. For all training, we use the Adam optimizer (Kingma and Ba, 2015) and a batch size of 256 samples. **Table 3** gives the exact hyperparameters and model architecture for each model we train and evaluate on permutedMNIST. Note that hyperparameters were optimized individually for each setting.

To combine Active Dendrites Network with SI, and to compare against XdG, we reduce the number of units in each hidden layer from 2,048 to 2,000 as to exactly match the architectures (with the exception of dendritic segments) used in the SI and XdG papers. (See **Supplementary Materials** for a discussion on the number of parameters.) In addition, the SI-and-Active-Dendrites network is trained for 20 epochs per task instead of just three as this significantly improves results. We fix the learning rate to be 5×10^{-4} for all numbers of tasks, and we use SI regularization strength $c = 0.1$ and damping coefficient $\xi = 0.1$. Both (a) training for 20 epochs per task and (b) the c, ξ values that we use here align with the training setups of Zenke et al. (2017) and Masse et al. (2018).

6.2.3. Constructing Prototypes During Training Without Task Information

When task information is not given during training nor testing, the task corresponding to each input example must be inferred. This section describes the online clustering method we implemented to infer task information during training. One inductive bias in our procedure is that all training examples in a batch correspond to the same task, since continual learning scenarios usually only observe examples from a single task within a given batch.

Formally, let $X = \{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(n)}\}$ be a batch of n training examples (in the case of permutedMNIST, each $\boldsymbol{x}^{(i)}$ is a 784-dimensional vector for $1 \leq i \leq n$). Suppose M individual prototypes are designated thus far: $\boldsymbol{p}_1, \ldots, \boldsymbol{p}_M$. For each $\boldsymbol{p}_j$ (where $1 \leq j \leq M$), the individual examples used to construct that prototype are also stored in memory: $Y_j = \{\boldsymbol{y}^{(1)}, \ldots, \boldsymbol{y}^{(m_j)}\}$, where m_j gives the number of examples for cluster j. These

TABLE 3 | The hyperparameters used to train each model on permutedMNIST.

	Active dendrites network (task info. provided)	Active dendrites network (task info. not provided)	3-Layer MLP	10-Layer MLP
Network hyperparameters				
Feedforward input size	784	784	784	784
Hidden sizes	2 × [2,048]	2 × [2,048]	2 × [2,048]	10 × [2,048]
Output size	10	10	10	10
Feedforward weight sparsity	50%	50%	0%	0%
Activation function	*k*WTA	*k*WTA	ReLU	ReLU
Activation sparsity	5%	5%	∼ 50%	∼ 50%
Num. dendritic segments per neuron	$\mathcal{T}$	$\mathcal{T}$	0	0
Num. weights per dendritic segment	784	784	—	—
Dendritic segment weight sparsity	0%	0%	—	—
Non-zero feedforward parameters	2,914,314	2,914,314	5,824,522	35,198,986
Non-zero dendritic parameters	$\mathcal{T} \times 3,211,264$	$\mathcal{T} \times 3,211,264$	0	0
Non-zero parameters (total)	See **Supplementary Materials**		5,824,522	35,198,986
Training hyperparameters				
($\mathcal{T} = 2$) Learning rate	5×10^{-4}	10^{-3}	—	—
($\mathcal{T} = 2$) Number of epochs	1	5	—	—
($\mathcal{T} = 5$) Learning rate	5×10^{-4}	10^{-3}	—	—
($\mathcal{T} = 5$) Number of epochs	1	5	—	—
($\mathcal{T} = 10$) Learning rate	5×10^{-4}	10^{-3}	3×10^{-6}	3×10^{-6}
($\mathcal{T} = 10$) Number of epochs	3	3	5	3
($\mathcal{T} = 25$) Learning rate	3×10^{-4}	3×10^{-4}	—	—
($\mathcal{T} = 25$) Number of epochs	5	1	—	—
($\mathcal{T} = 50$) Learning rate	3×10^{-4}	10^{-4}	—	—
($\mathcal{T} = 50$) Number of epochs	3	3	—	—
($\mathcal{T} = 100$) Learning rate	10^{-4}	10^{-4}	10^{-6}	3×10^{-7}
($\mathcal{T} = 100$) Number of epochs	3	3	3	3

previous training examples are observed by the learner during previous batches of learning and stored in memory. We identify if the new batch X is similar enough to any cluster of training examples Y_j such that the corresponding prototype $\boldsymbol{p}_j$ should be used as the context signal. If a cluster j is found such that X is "similar" to Y_j, then Y_j is expanded to include X. Subsequently, $\boldsymbol{p}_j$ is updated to incorporate samples from X. Otherwise, if X is deemed significantly different from Y_j for all j, then a new cluster is formed: $Y_{M+1} \leftarrow X$ and its prototype is the element-wise mean of all $\boldsymbol{x} \in X$. **Algorithm 1** describes the procedure for clustering during training when task information is not provided.

In the pseudocode, how do we determine when X is similar enough to some Y_j? If we have univariate data (i.e., if each $\boldsymbol{x} \in X$ and $\boldsymbol{y} \in Y_j$ is a scalar quantity), we could use an unpaired t-test do this. Instead, we use a generalized version of an unpaired t-test that applies to multivariate data. In our hypothesis testing setup, the null hypothesis is that for any given j, the same underlying process generates samples from both X and Y_j. When we accept the null hypothesis, we assume each $\boldsymbol{x} \in X$ and each $\boldsymbol{y} \in Y_j$ are training examples from the same permutedMNIST task—and therefore $\boldsymbol{p}_j$ can be used as the context signal when training an Active Dendrites Network on examples in X (albeit $\boldsymbol{p}_j$ is first updated to account for X).

Algorithm 1 : Clustering algorithm by which a new batch of inputs X either gets assigned to one of M existing clusters or initiates cluster $M + 1$. This procedure is greedy since it assigns X to the first cluster j that it suitably matches.

```
1:  procedure CLUSTER($X$, $Y$)
2:      $M \leftarrow 0$                          ▷ Number of existing clusters
3:      while not done learning do
4:          $X \leftarrow$ new batch
5:          assigned ← False
6:          for $j = 1$ to $M$ do
7:              if ¬ assigned and IS_MATCH($X$, $Y_j$) then
8:                  assigned ← True
9:                  $Y_j \leftarrow Y_j \cup X$
10:                 update $\boldsymbol{p}_j$ to include each $\boldsymbol{x} \in X$
```

Hotelling (1931) proposed Hotelling's t-squared statistic (t^2) as a generalization of the t-statistic used to perform single-variable t-tests; it is computed as

$$t^2 = \frac{|X||Y_j|}{|X| + |Y_j|} \left(\bar{\boldsymbol{x}} - \bar{\boldsymbol{y}}\right)^\top \boldsymbol{\Sigma}^{-1} \left(\bar{\boldsymbol{x}} - \bar{\boldsymbol{y}}\right)$$

where $\bar{\boldsymbol{x}}$ and $\bar{\boldsymbol{y}}$ are simply the element-wise means of all $\boldsymbol{x} \in X$ and $\boldsymbol{y} \in Y_j$, respectively, and $\boldsymbol{\Sigma}$ is the pooled, sample-adjusted covariance matrix of samples in X and Y_j. The test statistic t^2 can be compared to a chosen p-value to accept or reject the null hypothesis by first transforming it to a value drawn from an F-distribution (whose cumulative density function is more well-studied than that of the t-squared distribution) as follows:

$$f = \frac{|X| + |Y_j| - d - 1}{d\left(|X| + |Y_j| - 2\right)} t^2$$

where d is the dimensionality of the samples.

We fix a p-value and derive a value for f based on t^2 as give above. If $f > p$, then we reject the null hypothesis since the probability that the same generative process explains both X and Y_j is extremely low, and thus create a new cluster. Since we perform pairwise multivariate t-tests between X and Y_j for all existing prototypes j, a new cluster and prototype emerge if and only if we reject the null hypothesis for all M t-tests. **Algorithm 2** describes the procedure for performing the multivariate t-test *via* the t-squared statistic given two sets of multivariate samples.

Algorithm 2 : Unpaired multivariate t-test using Hotelling's t-squared statistic. Here, we use a slight abuse of notation when computing covariance matrices by assuming sets of d-dimensional vectors can also be treated as matrices whose rows correspond to their d-dimensional elements. We assume a p-value is fixed a priori. In our implementation, we replace all standard matrix inversions with the Moore-Penrose pseudo-inversion.

```
1:  procedure IS_MATCH($X$, $Y$)
2:      $\bar{\boldsymbol{x}} \leftarrow \frac{1}{|X|}\sum_{\boldsymbol{x}\in X}\boldsymbol{x}$    ▷ Compute $X$ mean
3:      $\bar{\boldsymbol{y}} \leftarrow \frac{1}{|Y|}\sum_{\boldsymbol{y}\in Y}\boldsymbol{y}$    ▷ Compute $Y$ mean
4:      $\boldsymbol{\Sigma}_X \leftarrow \frac{1}{|X|-1}(X-\bar{\boldsymbol{x}})(X-\bar{\boldsymbol{x}})^\top$    ▷ Compute $X$ covariance
5:      $\boldsymbol{\Sigma}_Y \leftarrow \frac{1}{|Y|-1}(X-\bar{\boldsymbol{y}})(Y-\bar{\boldsymbol{y}})^\top$    ▷ Compute $Y$ covariance
6:      $\boldsymbol{\Sigma} \leftarrow \frac{(|X|-1)\boldsymbol{\Sigma}_X+(|Y|-1)\boldsymbol{\Sigma}_Y}{|X|+|Y|-2}$    ▷ Compute pooled covariance
7:      $t^2 \leftarrow \frac{|X||Y_j|}{|X|+|Y_j|}\left(\bar{\boldsymbol{x}}-\bar{\boldsymbol{y}}\right)^\top \boldsymbol{\Sigma}^{-1}\left(\bar{\boldsymbol{x}}-\bar{\boldsymbol{y}}\right)$    ▷ Compute $t^2$
8:      $f = \frac{|X|+|Y_j|-d-1}{d(|X|+|Y_j|-2)}t^2$    ▷ Convert $t^2$ to $f$
9:      if $f > p$ then:
10:         return False    ▷ Reject null hypothesis
11:     else
12:         return True    ▷ Accept null hypothesis
```

6.3. Absolute Max Gating

We outline how we implement gating in Active Dendrites Networks. In Section 3, we present gating as modifying the value of the weighted linear sum computed by the point neuron based on the maximum activation, i.e., $\sigma(\max_j \boldsymbol{u}^\top \boldsymbol{c})$. One problem with this formulation is that it becomes difficult to turn a neuron off (i.e., force it's activation value to be zero) due to the max operator. That is, if dendritic segment j learns to turn off the unit, then based on sigmoidal gating, we should expect that $\boldsymbol{u}_j^\top \boldsymbol{c}$ is a small number with large absolute value (very negative). However, it's likely that for some other segment j' ($j \neq j'$), $\boldsymbol{u}_{j'}^\top \boldsymbol{c} > 0 > \boldsymbol{u}_j^\top \boldsymbol{c}$ which means that segment j' will be selected by the max operator instead of segment j, hence increasing the chance that the neuron will be selected by the kWTA process.

This motivates absolute max gating in which the activation with the largest magnitude is selected and its sign is kept. More formally, a point neuron augmented with absolute max gating computes its output as

$$j^* = \arg\max_j \left|\boldsymbol{u}_j^\top \boldsymbol{c}\right|,$$
$$\hat{y} = \left(\boldsymbol{w}^\top \boldsymbol{x} + b\right)\sigma\left(\boldsymbol{u}_{j^*}^\top \boldsymbol{c}\right).$$

DATA AVAILABILITY STATEMENT

The original contributions presented in the study are included in the article/**Supplementary Material**, further inquiries can be directed to the corresponding author.

AUTHOR CONTRIBUTIONS

SA conceived of the overall theory and the mapping to neuroscience. KG and SA implemented the core dendrites code. KG and JF implemented the continual learning experiments and their analysis. AV originally conceived of the mapping to multi-task RL and designed the original RL experiments. AI, LS, and AV implemented and ran the multi-task RL experiments. SA, AI, AV, and KG contributed to the article design and wrote the text. All authors contributed to the article and approved the submitted version.

ACKNOWLEDGMENTS

We thank Jeff Hawkins, Ben Cohen, Greg Maltz, Luiz Scheinkman, and the rest of the Numenta team for helpful feedback throughout the editing process.

SUPPLEMENTARY MATERIAL

The Supplementary Material for this article can be found online at: https://www.frontiersin.org/articles/10.3389/fnbot.2022.846219/full#supplementary-material

REFERENCES

Abbasi, A., Nooralinejad, P., Braverman, V., Pirsiavash, H., and Kolouri, S. (2022). Sparsity and heterogeneous dropout for continual learning in the null space of neural activations. *arXiv:2203.06514*. doi: 10.48550/arXiv.2203.06514

Ahmad, S., and Hawkins, J. (2016). How do neurons operate on sparse distributed representations? A mathematical theory of sparsity, neurons and active dendrites. *arXiv:1601.00720*. doi: 10.48550/arXiv.1601.00720

Ahmad, S., and Scheinkman, L. (2019). How can we be so dense? The benefits of using highly sparse representations. *arXiv:1903.11257*. doi: 10.48550/arXiv.1903.11257

Andreas, J., Klein, D., and Levine, S. (2017). "Modular multitask reinforcement learning with policy sketches," in *Proceedings of the 34th International Conference on Machine Learning* (Sydney, NSW).

Antic, S. D., Hines, M., and Lytton, W. W. (2018). Embedded ensemble encoding hypothesis: The role of the "prepared" cell. *J. Neurosci. Res*. 96, 1543–1559. doi: 10.1002/jnr.24240

Antic, S. D., Zhou, W.-L., Moore, A. R., Short, S. M., and Ikonomu, K. D. (2010). The decade of the dendritic NMDA spike. *J. Neurosci. Res*. 88, 2991–3001. doi: 10.1002/jnr.22444

Arulkumaran, K., Deisenroth, M. P., Brundage, M., and Bharath, A. A. (2017). A brief survey of deep reinforcement learning. *IEEE Signal Process. Mag*. 34, 26–38. doi: 10.1109/MSP.2017.2743240

Attwell, D., and Laughlin, S. B. (2001). An energy budget for signaling in the grey matter of the brain. *J. Cereb. Blood Flow Metab*. 21, 1133–1145. doi: 10.1097/00004647-200110000-00001

Barth, A. L., and Poulet, J. F. a. (2012). Experimental evidence for sparse firing in the neocortex. *Trends Neurosci*. 35, 345–355. doi: 10.1016/j.tins.2012.03.008

Beniaguev, D., Segev, I., and London, M. (2021). Single cortical neurons as deep artificial neural networks. *Neuron* 109, 2727–2739. doi: 10.1016/j.neuron.2021.07.002

Bentivoglio, M., and Swanson, L. W. (2001). On the fine structure of the PES hippocampi major. *Brain Res. Bull*. 54, 461–483. doi: 10.1016/S0361-9230(01)00430-0

Branco, T., and Häusser, M. (2010). The single dendritic branch as a fundamental functional unit in the nervous system. *Curr. Opin. Neurobiol*. 20, 494–502. doi: 10.1016/j.conb.2010.07.009

Branco, T., and Häusser, M. (2011). Synaptic integration gradients in single cortical pyramidal cell dendrites. *Neuron* 69, 885–892. doi: 10.1016/j.neuron.2011.02.006

Caruana, R. (1997). Multitask learning. *Mach. Learn*. 28, 41–75. doi: 10.1023/A:1007379606734

Chen, Z., Badrinarayanan, V., Lee, C.-Y., and Rabinovich, A. (2018). "Gradnorm: gradient normalization for adaptive loss balancing in deep multitask networks," in *Proceedings of the 35th International Conference on Machine Learning* (Stockholm).

Cui, Y., Ahmad, S., and Hawkins, J. (2017). The HTM Spatial Pooler–a neocortical algorithm for online sparse distributed coding. *Front. Comput. Neurosci*. 11:111. doi: 10.3389/fncom.2017.00111

Devin, C., Gupta, A., Darrell, T., Abbeel, P., and Levine, S. (2017). "Learning modular neural network policies for multi-task and multi-robot transfer," in *Proceedings of the IEEE International Conference on Robotics and Automation* (Singapore). doi: 10.1109/ICRA.2017.7989250

Dong, D., Wu, H., He, W., Yu, D., and Wang, H. (2015). "Multi-task learning for multiple language translation," in *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing* (Beijing: Association for Computational Linguistics), 1723–1732. doi: 10.3115/v1/P15-1166

Du, Y., Czarnecki, W. M., Jayakumar, S. M., Farajtabar, M., Pascanu, R., and Lakshminarayanan, B. (2020). Adapting auxiliary losses using gradient similarity. *arXiv:1806.08730*. doi: 10.48550/arXiv.1812.02224

Errington, A. C., and Connelly, W. M. (2011). Dendritic T-type Ca2+ channels: giving a boost to thalamic reticular neurons. *J. Neurosci*. 31, 5551–5553. doi: 10.1523/JNEUROSCI.0067-11.2011

Flesch, T., Nagy, D. G., Saxe, A., and Summerfield, C. (2022). Modelling continual learning in humans with Hebbian context gating and exponentially decaying task signals. *arXiv [Preprint]. arXiv: 2203.11560*. Available online at: https://arxiv.org/ftp/arxiv/papers/2203/2203.11560.pdf

French, R. M. (1999). Catastrophic forgetting in connectionist networks. *Trends Cogn. Sci*. 3, 128–135. doi: 10.1016/S1364-6613(99)01294-2

Gao, P. P., Graham, J. W., Zhou, W. L., Jang, J., Angulo, S., Dura-Bernal, S., et al. (2021). Local glutamate-mediated dendritic plateau potentials change the state of the cortical pyramidal neuron. *J. Neurophysiol*. 125, 23–42. doi: 10.1152/jn.00734.2019

Goodfellow, I. J., Mirza, M., Xiao, D., Courville, A., and Bengio, Y. (2014). "An empirical investigation of catastrophic forgetting in gradient-based neural networks," in *Proceedings of the 2nd International Conference on Learning Representations* (Banff, AB).

Goyal, A., Sodhani, S., Binas, J., Peng, X. B., Levine, S., and Bengio, Y. (2020). "Reinforcement learning with competitive ensembles of information-constrained primitives," in *Proceedings of the 8th International Conference on Learning Representations, Digital*.

Guest, J. M., Bast, A., Narayanan, R. T., and Oberlaender, M. (2021). Thalamus gates active dendritic computations in cortex during sensory processing. *bioRxiv*. doi: 10.1101/2021.10.21.465325

Haarnoja, T., Pong, V., Zhou, A., Dalal, M., Abbeel, P., and Levine, S. (2018a). "Composable deep reinforcement learning for robotic manipulation," in *Proceedings of the IEEE International Conference on Robotics and Automation* (Brisbane, QLD). doi: 10.1109/ICRA.2018.8460756

Haarnoja, T., Zhou, A., Abbeel, P., and Levine, S. (2018b). "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *Proceedings of the 35th International Conference on Machine Learning* (Stockholm).

Hawkins, J., and Ahmad, S. (2016). Why neurons have thousands of synapses, a theory of sequence memory in neocortex. *Front. Neural Circuits* 10:23. doi: 10.3389/fncir.2016.00023

Hawkins, J., Ahmad, S., and Cui, Y. (2017). A theory of how columns in the neocortex enable learning the structure of the world. *Front. Neural Circuits* 11:81. doi: 10.3389/fncir.2017.00081

Heald, J. B., Lengyel, M., and Wolpert, D. M. (2021). Contextual inference underlies the learning of sensorimotor repertoires. *Nature* 600, 489–493. doi: 10.1038/s41586-021-04129-3

Holmgren, C., Harkany, T., Svennenfors, B., and Zilberter, Y. (2003). Pyramidal cell communication within local networks in layer 2/3 of rat neocortex. *J. Physiol*. 551, 139–153. doi: 10.1113/jphysiol.2003.044784

Hotelling, H. (1931). The generalization of Student's ratio. *Ann. Math. Stat*. 2, 360–378. doi: 10.1214/aoms/1177732979

Ibarz, J., Tan, J., Finn, C., Kalakrishnan, M., Pastor, P., and Levine, S. (2021). How to train your robot with deep reinforcement learning: lessons we have learned. *Int. J. Robot. Res*. 40, 698–721. doi: 10.1177/0278364920987859

Irpan, A. (2018). *Deep Reinforcement Learning Doesn't Work Yet*. Available online at: https://www.alexirpan.com/2018/02/14/rl-hard.html (accessed November 1, 2021).

Jadi, M. P., Behabadi, B. F., Poleg-Polsky, A., Schiller, J., and Mel, B. W. (2014). An augmented two-layer model captures nonlinear analog spatial integration effects in pyramidal neuron dendrites. *Proc. IEEE* 102, 782–798. doi: 10.1109/JPROC.2014.2312671

Jayakumar, S. M., Czarnecki, W. M., Menick, J., Schwarz, J., Rae, J., Osindero, S., et al. (2020). "Multiplicative interactions and where to find them," in *Proceedings of the 8th International Conference on Learning Representations, Digital*.

Kandel, E. (2012). *Principles of Neural Science, 5th Edn*. New York, NY: McGraw-Hill.

Kendall, A., Gal, Y., and Cipolla, R. (2019). "Multi-task learning using uncertainty to weigh losses for scene geometry and semantics," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (Long Beach, CA).

Kerlin, A., Boaz, M., Flickinger, D., Maclennan, B. J., Dean, M. B., Davis, C., et al. (2019). Functional clustering of dendritic activity during decision-making. *eLife* 8:e46966. doi: 10.7554/eLife.46966

Kingma, D. P., and Ba, J. L. (2015). "Adam: a method for stochastic optimization," in *Proceedings of the 3rd International Conference on Learning Representations* (San Diego, CA).

Kirkpatrick, J., Pascanu, R., Rabinowitz, N., Veness, J., Desjardins, G., Rusu, A. A., et al. (2017). Overcoming catastrophic forgetting in neural networks. *Proc. Natl. Acad. Sci. U.S.A*. 114, 3521–3526. doi: 10.1073/pnas.1611835114

Lafourcade, M., van der Goes, M.-S. H., Vardalaki, D., Brown, N. J., Voigts, J., Yun, D. H., et al. (2022). Differential dendritic integration of long-range inputs in association cortex *via* subcellular changes in synaptic AMPA-to-NMDA receptor ratio. *Neuron*. doi: 10.1016/j.neuron.2022.01.025

Lapique, L. (1907). Recherches quantitatives sur l'excitation électrique des nerfs traitée comme une polarisation. *J. Physiol. Pathol*. 9, 620–635.

Larkum, M. (2022). Are dendrites conceptually useful? *Neurosci*. S0306-4522(22)00120-8. doi: 10.1016/j.neuroscience.2022.03.008. [Epub ahead of print].

Larkum, M. E., Zhu, J. J., and Sakmann, B. (1999). A new cellular mechanism for coupling inputs arriving at different cortical layers. *Nature* 398, 338–341. doi: 10.1038/18686

LeCun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. *Nature* 521, 436–444. doi: 10.1038/nature14539

Liang, F., Li, H., Chou, X.-l., Zhou, M., Zhang, N. K., Xiao, Z., et al. (2019). Sparse representation in awake auditory cortex: cell-type dependence, synaptic mechanisms, developmental emergence, and modulation. *Cereb. Cortex* 29, 3796–3812. doi: 10.1093/cercor/bhy260

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., et al. (2016). "Continuous control with deep reinforcement learning," in *Proceedings of the 4th International Conference on Learning Representations* (San Juan).

Limbacher, T., and Legenstein, R. (2020). Emergence of stable synaptic clusters on dendrites through synaptic rewiring. *Front. Comput. Neurosci*. 14:57. doi: 10.3389/fncom.2020.00057

Liu, S., Johns, E., and Davison, A. J. (2019). "End-to-end multi-task learning with attention," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (Long Beach, CA). doi: 10.1109/CVPR.2019.00197

London, M., and Häusser, M. (2005). Dendritic computation. *Annu. Rev. Neurosci*. 28, 503–532. doi: 10.1146/annurev.neuro.28.061604.135703

Losonczy, A., Makara, J. K., and Magee, J. C. (2008). Compartmentalized dendritic plasticity and input feature storage in neurons. *Nature* 452, 436–441. doi: 10.1038/nature06725

Magee, J. C. (2000). Dendritic integration of excitatory synaptic input. *Nat. Rev. Neurosci*. 1, 181–190. doi: 10.1038/35044552

Majani, E., Erlanson, R., and Abu-Mostafa, Y. (1989). "On the k-winners-take-all network," in *Advances in Neural Information Processing Systems* (Denver, CO).

Major, G., Larkum, M. E., and Schiller, J. (2013). Active properties of neocortical pyramidal neuron dendrites. *Annu. Rev. Neurosci*. 36, 1–24. doi: 10.1146/annurev-neuro-062111-150343

Maninis, K.-K., Radosavovic, I., and Kokkinos, I. (2019). "Attentive single-tasking of multiple tasks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (Long Beach, CA). doi: 10.1109/CVPR.2019.00195

Masse, N. Y., Grant, G. D., and Freedman, D. J. (2018). Alleviating catastrophic forgetting using context-dependent gating and synaptic stabilization. *Proc. Natl. Acad. Sci. U.S.A*. 115, E10467–E10475. doi: 10.1073/pnas.1803839115

McCann, B., Keskar, N. S., Xiong, C., and Socher, R. (2018). The natural language decathlon: Multitask learning as question answering. *arXiv:1806.08730*. doi: 10.48550/arXiv.1806.08730

McClelland, J. L., Rumelhart, D. E., and the PDP Research Group (1986). *Parallel Distributed Processing, 2nd Edn*. Cambridge, MA: MIT Press.

McCloskey, M. and Cohen, N. J. (1989). Catastrophic interference in connectionist networks: the sequential learning problem. *Psychol. Learn. Motiv*. 24, 109–165. doi: 10.1016/S0079-7421(08)60536-8

Misra, I., Shrivastava, A., Gupta, A., and Hebert, M. (2016). "Cross-stitch networks for multi-task learning," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (Las Vegas, NV). doi: 10.1109/CVPR.2016.433

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., et al. (2013). "Playing atari with deep reinforcement learning," in *Advances in Neural Information Processing Systems* (Lake Tahoe).

Paiton, D. M., Frye, C. G., Lundquist, S. Y., Bowen, J. D., Zarcone, R., and Olshausen, B. A. (2020). Selectivity and robustness of sparse coding networks. *J. Vis*. 20:10. doi: 10.1167/jov.20.12.10

Parisi, G. I., Kemker, R., Part, J. L., Kanan, C., and Wermter, S. (2019). Continual lifelong learning with neural networks: a review. *Neural Netw*. 113, 54–71. doi: 10.1016/j.neunet.2019.01.012

Phillips, W. A. (2015). Cognitive functions of intracellular mechanisms for contextual amplification. *Brain Cogn*. 112, 39–53. doi: 10.1016/j.bandc.2015.09.005

Phillips, W. A., Clark, A., and Silverstein, S. M. (2015). On the functions, mechanisms, and malfunctions of intracortical contextual modulation. *Neurosci. Biobehav. Rev*. 52, 1–20. doi: 10.1016/j.neubiorev.2015.02.010

Poirazi, P., Brannon, T., and Mel, B. W. (2003). Pyramidal neuron as two-layer neural network. *Neuron* 37, 989–999. doi: 10.1016/S0896-6273(03)00149-1

Poirazi, P., and Papoutsi, A. (2020). Illuminating dendritic function with computational models. *Nat. Rev. Neurosci*. 21, 303–321. doi: 10.1038/s41583-020-0301-7

Purushwalkam, S., Nickel, M., Gupta, A., and Ranzato, M. (2019). "Task-driven modular networks for zero-shot compositional learning," in *Proceedings of the IEEE International Conference on Computer Vision* (Long Beach, CA). doi: 10.1109/ICCV.2019.00369

Ramaswamy, S., and Markram, H. (2015). Anatomy and physiology of the thick-tufted layer 5 pyramidal neuron. *Front. Cell. Neurosci*. 9:233. doi: 10.3389/fncel.2015.00233

Rosch, E. (1975). Cognitive representations of semantic categories. *J. Exp. Psychol*. 104, 192–233. doi: 10.1037/0096-3445.104.3.192

Rosenbaum, C., Klinger, T., and Riemer, M. (2018). "Routing networks: adaptive selection of non-linear functions for multi-task learning," in *Proceedings of the 6th International Conference on Learning Representations* (Vancouver, BC).

Rosenblatt, F. (1958). The perceptron: a probabilistic model for information storage and organization in the brain. *Psychol. Rev*. 65:386. doi: 10.1037/h0042519

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature* 323, 533–536. doi: 10.1038/323533a0

Rusu, A. A., Colmenarejo, S. G., Gülçehre, C., Desjardins, G., Kirkpatrick, J., Pascanu, R., et al. (2016). "Policy distillation," in *Proceedings of the 4th International Conference on Learning Representations* (San Juan).

Sahni, H., Kumar, S., Tejani, F., and Isbell, C. (2017). "Learning to compose skills," in *Advances in Neural Information Processing Systems* (Long Beach, CA).

Schmidt-Hieber, C., Toleikyte, G., Aitchison, L., Roth, A., Clark, B. A., Branco, T., et al. (2017). Active dendritic integration as a mechanism for robust and precise grid cell firing. *Nat. Neurosci*. 20, 1114–1121. doi: 10.1038/nn.4582

Schoenfeld, G., Kollmorgen, S., Lewis, C., Bethge, P., Reuss, A. M., Aguzzi, A., et al. (2022). Dendritic integration of sensory and reward information facilitates learning. *bioRxiv*. doi: 10.1101/2021.12.28.474360

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017). Proximal policy optimization algorithms. *arXiv:1707.06347*. doi: 10.48550/arXiv.1707.06347

Sener, O., and Koltun, V. (2018). "Multi-task learning as multi-objective optimization," in *Advances in Neural Information Processing Systems* (Montréal, QC).

Sezener, E., Grabska-Barwínska, A., Kostadinov, D., Beau, M., Krishnagopal, S., Budden,D., et al. (2021). A rapid and efficient learning rule for biological neural circuits. *bioRxiv*. doi: 10.1101/2021.03.10.434756

Siegel, M., Körding, K. P., and König, P. (2000). Integrating top-down and bottom-up sensory processing by somato-dendritic interactions. *J. Comput. Neurosci*. 8, 161–173. doi: 10.1023/A:1008973215925

Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., et al. (2018). A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science* 362, 1140–1144. doi: 10.1126/science.aar6404

Snell, J., Swersky, K., and Zemel, R. S. (2017). "Prototypical networks for few-shot learning," in *Advances in Neural Information Processing Systems* (Long Beanch, CA).

Spruston, N. (2008). Pyramidal neurons: dendritic structure and synaptic integration. *Nat. Rev. Neurosci*. 9, 206–221. doi: 10.1038/nrn2286

Stuart, G., Spruston, N., and Häusser, M., editors (2016). *Dendrites, 3rd Edn*. Oxford: Oxford University Press. doi: 10.1093/acprof:oso/9780198745273.001.0001

Stuart, G. J., and Spruston, N. (2015). Dendritic integration: 60 years of progress. *Nat. Neurosci*. 18, 1713–1721. doi: 10.1038/nn.4157

Sutton, R. S., and Barto, A. G. (2018). *Reinforcement Learning: An Introduction, 2nd Edn*. Cambridge, MA: MIT Press.

Takahashi, N., Ebner, C., Sigl-Glöckner, J., Moberg, S., Nierwetberg, S., and Larkum, M. E. (2020). Active dendritic currents gate descending cortical outputs in perception. *Nat. Neurosci*. 23, 1–9. doi: 10.1038/s41593-020-0677-8

van de Ven, G. M., and Tolias, A. S. (2019). Three scenarios for continual learning. *arXiv:1904.07734*. doi: 10.48550/arXiv.1904.07734

Veness, J., Lattimore, T., Budden, D., Bhoopchand, A., Mattern, C., Grabska-Barwinska, A., et al. (2021). "Gated linear networks," in *Proceedings of the 35th AAAI Conference on Artificial Intelligence, Digital*.

Whittington, J. C. R., McCaffary, D., Bakermans, J. J. W., and Behrens, T. E. J. (2022). How to build a cognitive map: insights frommodels of the hippocampal formation. *arXiv [Preprint]. arXiv: 2202.01682*. Available online at: https://arxiv.org/pdf/2202.01682.pdf

Wilson, A., Fern, A., Ray, S., and Tadepalli, P. (2007). "Multi-task reinforcement learning: a hierarchical Bayesian approach," in *Proceedings of the 24th International Conference on Machine Learning* (Corvallis, OR). doi: 10.1145/1273496.1273624

Wortsman, M., Ramanujan, V., Liu, R., Kembhavi, A., Rastegari, M., Yosinski, J., et al. (2020). "Supermasks in superposition," in *Advances in Neural Information Processing Systems, Digital*.

y Cajal, S. R. (1894). Neue darstellung vom histologischen bau des centralnervensystems. *Am. J. Psychol*. 6:450. doi: 10.2307/1411662

Yang, G., Lai, C. S. W., Cichon, J., Ma, L., Li, W., and Gan, W.-B. (2014). Sleep promotes branch-specific formation of dendritic spines after learning. *Science* 344, 1173–1178. doi: 10.1126/science.1249098

Yang, R., Xu, H., Wu, Y., andWang, X. (2020). "Multi-task reinforcement learning with soft modularization," in *Advances in Neural Information Processing Systems*.

Yu, T., Kumar, S., Gupta, A., Levine, S., Hausman, K., and Finn, C. (2020). "Gradient surgery for multi-task learning," in *Advances in Neural Information Processing Systems*.

Yu, T., Quillen, D., He, Z., Julian, R., Narayan, A., Shively, H., et al. (2019). "Meta-world: a benchmark and evaluation for multi-task and meta reinforcement learning," in *Proceedings of the 3rd Conference on Robot Learning* (Osaka).

Zenke, F., Poole, B., and Ganguli, S. (2017). "Continual learning through synaptic intelligence," in *Proceedings of the 34th International Conference on Machine Learning* (Sydney, NSW).

Zhang, Y., and Yeung, D.-Y. (2014). A regularization approach to learning task relationships in multitask learning. *ACM Trans. Knowl. Discov. Data* 8, 1–31. doi: 10.1145/2538028

Conflict of Interest: SA, AI, KG, and LS are employed by Numenta Inc. Numenta has some patents relevant to the work. Numenta has stated that use of its intellectual property, including all the ideas contained in this work, is free for non-commercial research purposes. In addition, Numenta has released all pertinent source code as open source under a GPL V3 license (which includes a patent peace provision).

The remaining authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.