NISPA: Neuro-Inspired Stability-Plasticity Adaptation for Continual Learning in Sparse Networks

Mustafa Burak Gurbuz [1] Constantine Dovrolis [1 2]

Abstract

The goal of continual learning (CL) is to learn different tasks over time. The main desiderata associated with CL are to maintain performance on older tasks, leverage the latter to improve learning of future tasks, and to introduce minimal overhead in the training process (for instance, to not require a growing model or retraining). We propose the Neuro-Inspired Stability-Plasticity Adaptation (NISPA) architecture that addresses these desiderata through a sparse neural network with fixed density. NISPA forms stable paths to preserve learned knowledge from older tasks. Also, NISPA uses connection rewiring to create new plastic paths that reuse existing knowledge on novel tasks. Our extensive evaluation on EMNIST, FashionMNIST, CIFAR10, and CIFAR100 datasets shows that NISPA significantly outperforms representative state-of-the-art continual learning baselines, and it uses up to ten times fewer learnable parameters compared to baselines. We also make the case that sparsity is an essential ingredient for continual learning. The NISPA code is available at https://github.com/BurakGurbuz97/NISPA

1. Introduction

Recently, deep neural networks (DNNs) have achieved impressive performance on a wide variety of tasks, and they often exceed human-level ability. However, they rely on shuffled, balanced, fixed datasets and stationary environments (LeCun et al., 2015; Silver et al., 2016; Guo et al., 2016). As a result, they lack a critical characteristic of general intelligence, which is to continually learn over time in a dynamic environment. Under this Continual Learning (CL) scenario, DNNs forget what was learned in earlier tasks when learning new tasks, leading to a daunting phenomenon known as Catastrophic Forgetting (CF) (McCloskey & Cohen, 1989).

In stark contrast, animals excel at learning and remembering many different tasks they encounter over time. At least so far, the brain is the only existing system for successful CL in complex environments. This highlights the importance of reviewing basic facts about the brain's structure and function, and suggests the design of neuro-inspired mechanisms to mitigate CF (Hadsell et al., 2020; Hassabis et al., 2017). We summarize next some insights that provide hints for successful CL models.

Sparse connectivity: In contrast to DNNs' dense and highly entangled connectivity that is prone to interference (French, 1999), the brain relies on sparse connectivity in which only few neurons respond to any given stimulus (Babadi & Sompolinsky, 2014). After extensive synaptic pruning during childhood (Chechik et al., 1998), the connection density of the brain stays roughly constant (in healthy adults). In other words, the brain does not compromise sparsity to accumulate knowledge – instead it rewires existing neurons to create more effective neural pathways. This suggests a first mechanism we can transfer in DNNs: *persistent sparsity over the course of continual learning* (Hadsell et al., 2020).

Functional and structural plasticity: The brain learns through two forms of plasticity. First, functional or Hebbian plasticity adjusts the strength of synaptic transmission. A specific instance of Hebbian plasticity is Spike Timing Dependent Plasticity (STDP), which controls the strength of a synapse based on the relative timing of the corresponding neurons' activity (Park et al., 2014; Cooke & Bliss, 2006). Second, structural plasticity changes the brain's circuitry by forming new, or removing existing, synapses. While the underlying mechanisms are not fully understood yet, recent studies have shown that synapse rewiring occurs rapidly during learning (Fu & Zuo, 2011; Kasai et al., 2010; Deger et al., 2012). This implies that learning takes place in the brain by simultaneously changing both connection weights and network architecture. This suggests a second mechanism we can transfer to DNNs: *connection rewiring to*

[1] School of Computer Science, Georgia Institute of Technology, USA. [2] KIOS Research and Innovation Center of Excellence, Cyprus. Correspondence to: Constantine Dovrolis <constantine@gatech.edu>.

Proceedings of the 39^{th} International Conference on Machine Learning, Baltimore, Maryland, USA, PMLR 162, 2022. Copyright 2022 by the author(s).

create new, and prune existing, paths in the network when learning novel tasks.

Synaptic stability to avoid forgetting: A remarkable trait of the brain is its capacity to assimilate new information throughout life without disrupting the stability of previous knowledge (Parisi et al., 2019). Dendritic spines (single synaptic inputs) are highly stable after a learning window. This suggests that stable spines serve as substrates for long-term information storage (Zuo et al., 2005; Yang et al., 2009; 2014; Grutzendler et al., 2002). For instance, after a mouse learns a new task the volume of individual dendritic spines in associated neurons increases. Furthermore, this increased volume is maintained even after learning additional tasks (Yang et al., 2009). On the other hand, the mouse forgets those tasks once those enlarged spines are experimentally removed. These results support that learning a new task requires forming task-specific stable synapse ensembles that are restrained from future change. (Cichon & Gan, 2015; Hayashi-Takagi et al., 2015). This suggests a third mechanism we can transfer to DNNs: *disable gradient updates of certain hidden units' inputs to retain previously learned knowledge.*

Absence of neurogenesis: There is evidence that adult neurogenesis takes place in few mammalian brain regions, especially in the cerebellum (Carletti & Rossi, 2008) and hippocampus (Ming & Song, 2011). However, the brain's capacity in terms of number of neurons remains mostly the same, despite learning more and more tasks over the course of life. Therefore, creating new neurons is *not* the brain's preferred mechanism to learn new tasks. This suggests a fourth mechanism we can transfer to DNNs: *maintain a fixed capacity model (fixed number of layers and units), even if the architecture is dynamic through rewiring.*

Absence of rehearsal: The brain does not store ”raw” examples (e.g., pixel-level images). Also, it does not need periodic retraining on all previously known tasks when learning new concepts (van de Ven et al., 2020; Hayes et al., 2021). Instead, and mostly during sleep, the brain consolidates the memories of important new experiences, avoiding interference between old and new memories (Niethard et al., 2017; Yang et al., 2014). This suggests a fifth mechanism we can transfer to DNNs: *instead of relying on biologically implausible rehearsal mechanisms to alleviate CF, embed new knowledge in the DNN's dynamic architecture.*

This paper proposes the Neuro-Inspired Stability-Plasticity Adaptation (NISPA) architecture for CL that is based on the five previous mechanisms:

(1) We utilize sparsity to improve CL. Diverging from the mainstream practice of utilizing a dense architecture, we start with a sparse network and maintain the same connection density throughout the learning trajectory.

(2) In contrast to fully connected networks, sparse networks let us rewire connections. The rewiring process has two goals: disentangle interfering units to avoid forgetting, and create novel pathways to encode new knowledge.

(3) Motivated by persistent dendritic spines, we create stable hidden units by “freezing” the incoming connections of those units. So, for each learned task, we select a small set of units that remain stable to avoid forgetting that task.

(4) NISPA does not require model expansion. It sequentially accumulates knowledge into a fixed set of hidden units.

(5) Similar to sleep and memory consolidation, NISPA requires some time during training to figure out which paths are essential for remembering a new task and whether new paths can be added without causing interference with prior tasks.

From a computational perspective, NISPA only uses an extra bit per unit to mark whether that unit is stable or plastic. Furthermore, thanks to sparsity and rewiring, it requires much fewer parameters to achieve better performance than state-of-the-art methods.

We have mostly evaluated NISPA on task incremental learning (van de Ven & Tolias, 2019) (i.e., task labels are available during testing). Our experiments on EMNIST, FashionMNIST, CIFAR10, and CIFAR100 datasets show that NISPA significantly outperforms representative state-of-the-art methods on both retaining learned knowledge and performing well on new tasks. It also uses up to ten times fewer learnable parameters compared to baselines.

In Section 4.5, we present a NISPA extension for class incremental learning, where task labels are not provided during testing. Unfortunately, that extension requires the storage and replay of few examples for previous classes. We will aim to address that limitation in future work.

2. Related Work

2.1. Dynamic Sparse Nets for Single-Task Learning

Training sparse neural networks has been extensively explored (Bellec et al., 2018; Evci et al., 2020; Mocanu et al., 2018; Liu et al., 2021b; 2020; 2021a). This line of work proposes different architectures by dropping and growing connections during training. Similar to the brain, these models simultaneously learn both connection strengths and a sparse architecture – but they are restricted to single task learning.

2.2. Regularization Methods for CL

Regularization-based CL approaches modulate gradient updates, aiming to identify and preserve the weights that are more important for each task (Kirkpatrick et al., 2017; Zenke et al., 2017; Aljundi et al., 2018; Li & Hoiem, 2017; Serra et al., 2018; Chaudhry et al., 2018). Besides their mathematical foundation, these methods are also motivated by studies of synaptic consolidation (Hadsell et al., 2020). NISPA can also be classified as a regularization-based CL method. Other such methods however need to store more complex regularization parameters (typically floating-point numbers) for every connection compared to NISPA's single-bit overhead per unit.

2.3. Rehearsal Methods for CL

These methods store (Rebuffi et al., 2017; Rolnick et al., 2019; Isele & Cosgun, 2018) or generate (Shin et al., 2017; Atkinson et al., 2018; Ramapuram et al., 2020) examples of previous tasks. They retain knowledge by replaying those examples periodically. From a neuroscience perspective storing raw data is not biologically plausible (Hadsell et al., 2020; van de Ven et al., 2020; Hayes et al., 2021; Ramirez et al., 2013). It also introduces a major computational overhead. Additionally, the generative model itself is susceptible to forgetting and mode collapse. Such methods require significantly more resources and they are not directly comparable to NISPA.

2.4. Parameter Isolation Methods for CL

Parameter isolation methods assign different parameters for each task to mitigate forgetting. This parameter isolation is often achieved by growing new branches for tasks and freezing previous task parameters (Rusu et al., 2016; Li et al., 2021; Sahbi & Zhan, 2021; Yoon et al., 2018; Aljundi et al., 2017; Rosenfeld & Tsotsos, 2018). However, such model expansion is often not acceptable in practice because it increases the computational requirements linearly with the number of tasks. Another line of work addresses this limitation by utilizing a fixed-capacity architecture. Similar to NISPA, such approaches remove certain connections to limit interference, and they freeze essential connections to ensure stable performance on previous tasks (Golkar et al., 2019; Jung et al., 2020b; Sokar et al., 2021). However, they often depend on hard-to-tune hyperparameters. Additionally they can suffer from freezing entire layers, which prohibits further learning (Golkar et al., 2019).

3. NISPA description

Figure 1 illustrates the key ideas in NISPA, while **Algorithm** 1 (see Appendix) presents the complete method. The training for each task is divided into "phases" (e epochs each). After each phase, we select *candidate stable units* (Section 3.3). This selection step is followed by *connection rewiring* (Section 3.7). This selection and rewiring cycle is repeated for several phases until a *stopping criterion* is met (Section 3.8). Once a task ends, we promote candidate units to stable units, freeze the incoming connections of all stable units (Section 3.2) and reinitialize the weights of plastic units.

3.1. Notation and Problem Formulation

In most of the paper we consider task incremental learning, with a sequence of T tasks. For each task t we have a training dataset D_t and a validation dataset V_t. Task identifiers are available during training and testing. The layer $l \in \{1, \dots, L\}$ has N_l units and let n_i^l ($i \in \{1, \dots, N_l\}$) be the i-th unit in that layer. Let $\theta_{i,j}^l$ denote the j-th incoming connection of n_i^l from n_j^{l-1}. We denote the activation of unit n_i^l as $a_{n_i^l}(x)$ and the total activation of layer l as $a_l(x)$, where x is the input that generates these activations. The activation function of all units, excluding the outputs, is ReLU.

NISPA maintains a certain connection density d throughout the training process. The density d is defined as the ratio between the number of connections after and before pruning. Pruning is performed randomly, at initialization, and on a per-layer basis. In other words, every layer has the same density d.

In convolutional neural networks (CNNs), a "unit" is replaced by a 3D convolution filter. Likewise, a "connection" is replaced by a 2D kernel.

3.2. Stabilizing a Plastic Unit

The activation of unit n_i^l is determined by its parent units' activations and the weight of incoming connections from those parents. So, any weight updates during training alter unit n_i^l in two ways: first, directly changing the weights of incoming connections into the unit. Second, indirectly, by changing the weights of incoming connections to the unit's ancestor units, i.e., units in any path from the inputs to n_i^l. NISPA ensures that stable units receive input only from other stable units, using connection rewiring (Section 3.7). Additionally, at the boundary between two tasks, it freezes connections into new stable units to stabilize those units, i.e., it does not allow the corresponding weights to change after that point.

3.3. Selecting Candidate Stable Units

At any given time, the units U^l of layer l are partitioned into three disjoint sets: S_c^l, S^l and P^l, namely candidate stable, stable, and plastic units. While learning task t, NISPA periodically transitions some plastic units from P^l into the set of candidate (stable) units S_c^l. At the end of the training for that task, the members of S_c^l are promoted into S^l.

Suppose we start with the sets P^l and S^l we inherited from the previous task (if there is no previous task all units are

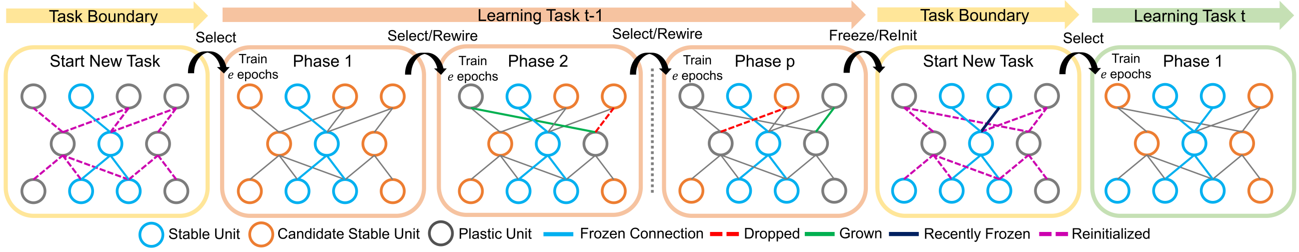

Figure 1. The training for each task is divided into "phases" (e epochs each). Between successive phases, we select candidate stable units, and the rewiring process takes place. Upon a task boundary, candidate stable units are promoted to stable units, the connections between stable units are frozen, and we reinitialize the remaining connections.

plastic). First, we compute the total activation at each layer across all training examples for that task, as follows:

$$A_l = \sum_{x \in D_t} a_l(x) = \sum_{x \in D_t} \sum_{i=1}^{N_l} a_{n_i^l}(x) \tag{1}$$

Next, for each layer we select the candidate stable units S_c^l as follows:

$$\min_{S_c^l \subseteq P^l} \left| S_c^l \right| \text{ subject to } \sum_{x \in D_t} \sum_{n_i^l \in S_c^l \cup S^l} a_{n_i^l}(x) \geq \tau A_l \tag{2}$$

The aim is to compute the smallest set of units $S_c^l \subseteq P^l$ that we need to add to S^l to capture at least a fraction τ of the total activation in layer l (the selection of τ is discussed in Section 3.4). Then we remove those elements of S_c^l from P^l.

The previous optimization problem is solved heuristically as follows. We start with $S_c^l \equiv \emptyset$. Then we add plastic units with the largest total activation one by one into S_c^l, until the τ criterion is satisfied. If S^l already captures at least τ fraction of the overall layer activation, the algorithm does not select any candidate stable units and S_c^l remains empty. Note that the selection process is performed in parallel at each layer. The input and output layers do not participate in this process because they are considered stable by definition.

The rationale of the previous approach is: any units that have remained in S_c^l at the end of that task's training are highly active while learning task t. So, to avoid forgetting that task in the future, we stabilize these units by disabling any gradient updates in their input paths.

3.4. Calculating τ

In the early phases of a task's training, the activations can vary erratically. So, it is better to start with a larger τ, resulting in more candidate stable units S_c^l. As the training proceeds, the network becomes more competent in that task, the activations are more stable, and we can further restrict the selection of candidate stable units.

This intuition suggests a gradual reduction of τ, starting with $\tau_1 = 1$ in the first phase, and decreasing τ in step sizes that increase with every phase. To do so we use the following cosine annealing schedule:

$$\tau_p = \frac{1}{2}\left(1 + \cos\left(\frac{p \times \pi}{k}\right)\right) \tag{3}$$

, where p is the phase number and k (typically 30 or 40) is a hyperparameter that determines the shape of the function.

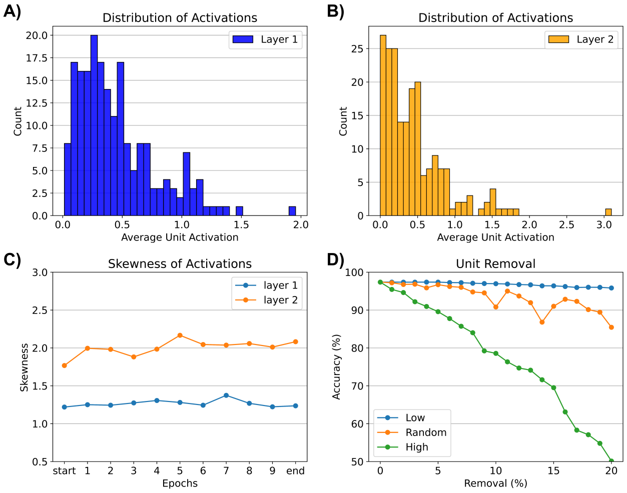

Figure 2. A and B show the activation distributions in a sparse (10% density) two-layer multilayer perceptron after training on MNIST – both distributions are highly skewed. C demonstrates that skewness does not change much during training. D shows that the removal of highly active units hurts the performance of the model much more than the removal of other units.

3.5. Skewness of activations

One may ask whether too many units will be selected as stable, not leaving enough plastic units for subsequent tasks. Prior work has observed that using ReLUs leads to a per-layer activation distribution that is highly skewed across different network architectures and datasets (Kurtz et al., 2020; Hu et al., 2016; Rhu et al., 2018; Georgiadis, 2019). The main reason for this skewness is that a ReLU maps all negative pre-activations to zero. As a result, the activation of most units is almost zero, while only few units have a large activation. **Figure** 2-A,B shows an example of this phenomenon on the MNIST classification task.

Figure 2-C shows that the skewness of the activation distribution (one per layer) is high even at the first epoch, and it remains high throughout the training period.

This is an important point for NISPA because it suggests we can satisfy the τ constraint by selecting only few units from the right tail of the distribution (i.e., the most active units). So, we expect that many units will remain plastic for learning future tasks.

3.6. Activation as a Measure of Importance

Another concern may be whether the total activation of a unit is a valid indicator of its contribution to learning a task. Activations are often used to determine which units are more important for a given task in problems such as CL (Jung et al., 2020a; Golkar et al., 2019), network pruning (Kurtz et al., 2020; Hu et al., 2016; Rhu et al., 2018), and model interpretation (Erhan et al., 2009; Zeiler & Fergus, 2014). There are several empirical results that support this choice, such as the observation that removing the most active units degrades performance much more than removing the same number of randomly chosen units (Jung et al., 2020a). Similar results for the MNIST classification task are shown in **Figure** 2-D, highlighting the strong correlation between a unit's activation and its importance for a given task.

3.7. Connection Rewiring

NISPA leverages connection rewiring for two reasons: mitigate forgetting and create novel pathways for forward transfer. Rewiring follows the selection of candidate stable units S_c^l at the end of each phase, and it consists of (1) dropping and (2) growing connections, as described next.

(1) We remove connections from plastic units to (possibly candidate) stable units. So, future changes in a plastic unit's functionality will not propagate to stable units. More formally, given S^l, P^l, and S_c^l, all connections $\theta_{i,j}^l$ where $n_i^l \in S^l \cup S_c^l$ and $n_j^{l-1} \in P^{l-1}$ are dropped.

(2) Dropping some connections from a layer l is followed by growing the same number of new connections in layer l, maintaining the per-layer density. The new connections are selected randomly as long as they do not form new inputs to stable units (i.e., $\theta_{i,j}^l$ where $n_i^l \in P^l$ and $n_j^{l-1} \in U^{l-1}$). This guarantees that connection growth will not disrupt representations learned by stable units.

The weight of new connections is initialized based on existing weights, as follows. Let μ_l be the mean and σ_l the standard deviation of the existing weights at layer l. We sample a new weight as $\theta^l \sim \mathcal{N}_l(\mu_l, \sigma_l)$.

NISPA only grows connections between plastic units (type-1) or from (possibly candidate) stable units to plastic units (type-2). Type-1 and type-2 connections serve different purposes. Type-1 connections may enable learning new representations for future tasks. In contrast, type-2 connections promote forward transfer, as plastic units can utilize learned and stable representations. Depending on the similarities across tasks and the layer at which the connections are added, type-1 could be more or less valuable than type-2.

3.8. Stopping criterion

The number of phases is not fixed. Instead, we track the highest accuracy achieved so far on the validation dataset V_t. At the end of each phase, if the new accuracy is worse than the best seen accuracy, we stop training and revert the model to the end of the previous phase. In other words, we perform early stopping at the level of phases instead of epochs.

Note that a task's training ends with a final sequence of e epochs. This allows the network to recover from any performance loss due to the last rewiring process.

We observed that plastic units start with a bias from the last task's training, which hinders learning the new task. For this reason we re-initialize the weights of all non-frozen connections before the training for a new task starts.

4. Experimental Results

In this section, we compare NISPA against state-of-the-art CL methods and other baselines. We also conduct ablation studies to evaluate the importance of different NISPA mechanisms. We primarily consider three task sequences. First, a sequence of 5 tasks derived from EMNIST (Cohen et al., 2017) and FashionMNIST (Xiao et al., 2017) that we refer to as **EF-MNIST** – Task1: 10 digits, Task2: initial 13 uppercase letters, Task3: remaining 13 uppercase letters, Task4: 11 lowercase letters (different than their uppercase counterparts), Task5: 10 FashionMNIST classes. Second, five tasks with two classes each from CIFAR10 (Krizhevsky et al., a). And third, 20 tasks derived from CIFAR100 (Krizhevsky et al., b) with five classes per task.

In the case of EF-MNIST, we use a three-layer multilayer perceptron (MLP) with 400 units each. For CIFAR10/CIFAR100, we use a network with four convolutional layers (3x3 kernel, stride=1 – 64 filters at first two layers, 128 filters at next two layers – second and fourth convolutional layers use max-pooling), followed by two linear layers (hidden layer with 1024 units followed by output layer).

The output layer relies on a multi-head approach in which the activations of irrelevant output units are masked out during training and testing.

NISPA utilizes uniform random pruning (the edge weight is set to zero or each 2D filter is set to a zero-matrix, with equal probability) at initialization. The MLP density is 20% and the CNN density is 10%, unless stated otherwise.

4.1. Supervised Learning on Vision Datasets

4.1.1. Comparison with Single Task Learners

Suppose we use NISPA with density d to sequentially learn T tasks. First, let us compare with two baselines: a single task learner (STL) with density d represents a performance upper bound because it dedicates all parameters to learn only a single task. Also, an STL with density $\frac{d}{T}$ (referred to as "STL ISO") represents a lower bound as it corresponds to partitioning the network into T equal-size subnetworks and learning a different task in each subnetwork.

Figure 3 shows that NISPA outperforms STL ISO, and performs close to STL, in the first four tasks of EF-MNIST. However, it performs rather poorly on Task-5. The first four tasks consist of handwritten digits and letters, while Task-5 is fashion items. Therefore, NISPA cannot leverage its knowledge of previous tasks in Task-5.

Figure 3 also shows that in the case of CIFAR10 and CIFAR100, where all tasks come from the same domain, NISPA dominates STL ISO and matches the performance of STL on most tasks. Note that STL ISO fails to learn the CIFAR100 task – its accuracy is less than 50%. This highlights that NISPA's success is not only due to parameter isolation (dropping and freezing connections). Sharing representations via novel connections from stable to plastic units plays a crucial role in successful CL.

4.1.2. Comparison with State-of-the-Art Methods

We compare NISPA to three well-known regularization methods: EWC (Kirkpatrick et al., 2017), SI (Zenke et al., 2017), and MAS (Aljundi et al., 2018). We also compare with the parameter isolation method CLNP (Golkar et al., 2019). All four methods are suitable baselines because, similar to NISPA, they do not rely on model expansion or rehearsal. These baselines use dense networks (density $d = 1$), giving them five to ten times more learnable parameters compared to NISPA (see Appendix **Table** 5 for an exact comparison).

Figure 4 shows results for each dataset. First, NISPA matches or outperforms CLNP on EF-MNIST, and it does much better than other baselines although they have five times more learnable parameters. On CIFAR10 and CIFAR100, we observe a more significant gap between NISPA and all baselines. EWC has the closest performance to NISPA in the CIFAR100 tasks (on the average, 70.96% versus 75.88% respectively). This is remarkable because NISPA uses ten times fewer learnable parameters than EWC.

Interestingly, in contrast to CLNP's good performance on an MLP, it performs similarly to other regularization-based methods on CNNs. CLNP aggressively freezes convolutional filters, leading to a mostly frozen network after few initial tasks. This limitation is also mentioned by the CLNP authors (Golkar et al., 2019) – the first task alone almost freezes the entire first and second convolutional layers, leaving little room for learning new low-level features in subsequent tasks. In contrast, NISPA's phased approach selects gradually the plastic units that will be stabilized for each task, and the training process continues during rewiring allowing the network to adapt to rewiring changes.

On the other hand, the gap between regularization-based methods and NISPA is more significant in initial tasks (see **Figure** 4-Right). We argue that this is because those baselines only address one aspect of forgetting: they penalize weight changes of some important connections. However, they overlook the indirect interference caused by altering the weights of an ancestor of a stable unit. So, the effect of small changes accumulates throughout the network, and when the task sequence is long enough, those baselines still suffer from forgetting.

Table 4 (see Appendix) shows the standard deviation of the average accuracy across all tasks and 5 runs. NISPA has the most stable performance compared to other CL baselines.

4.2. Sparsity Improves Continual Learning

In general, sparse networks are desirable since they require less computation and storage than their dense counterparts. However, we also argue that sparsity can also help with the CL objective. To understand why, consider first a fully connected network: the stabilization of a unit with NISPA will cause all units at the previous layer to either lose or freeze one of their outgoing connections. In contrast, in a sparse network only few connections would be affected. More generally, a sparse network allows NISPA to reduce the "interference" across tasks because each task relies on relatively few (compared to the size of the entire network) interconnected stable units that do not receive any input from plastic units.

Figure 5 shows the interplay between network density and the performance of NISPA. We observe that there is always a critical density at which the performance is optimal. Denser models suffer from highly entangled units, while sparser models suffer from under-fitting. An interesting open ques-

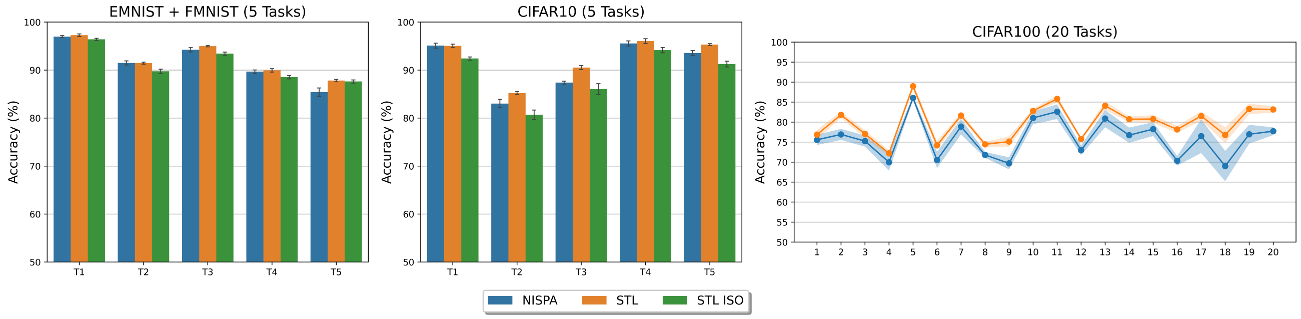

Figure 3. Accuracy for each task after learning is complete. Each data point indicates the average accuracy across five runs (± one std.deviation). STL ISO performs poorly on CIFAR100 tasks – its accuracy is less than 50% – so we omit STL ISO from the CIFAR100 plot.

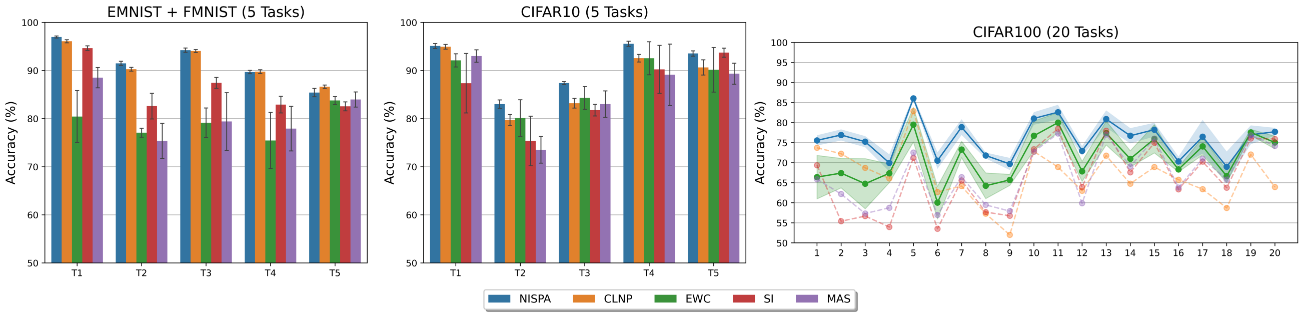

Figure 4. Accuracy for each task after learning is complete. Each data point indicates the average accuracy across five runs (± one std.deviation). We highlight the top-2 competing methods (NISPA and EWC) in the CIFAR100 plot.

tion is to better characterize and even predict that critical network density for a given sequence of tasks.

4.3. NISPA Efficiently Reuses Representations

NISPA aims to reuse representations and promote forward transfer, by maintaining connections from stable units to plastic units at the next layer. So, we may ask: how does the similarity across two tasks affect the number of new stable units that will be required to learn the second task? We expect that learning a second task that is similar to the first will require the stabilization of fewer new units than learning a very different second task.

To examine this hypothesis quantitatively we train an MLP on classifying MNIST. The second task is the same but operating on permuted MNIST images, in which we have permuted randomly a fraction p_r of the pixels. We chose this task because random permutations are equally challenging for an MLP as classifying the original images. Therefore, the only variable in this experiment is p_r, which is a knob to adjust the similarity of the two tasks (see Appendix C.4 for details).

Figure 6 shows the number of additional stable units that NISPA selects for the second task. As expected, the number of additional stable units increases as p_r increases, because previously learned representations become less helpful. In this particular task sequence we do not observe a pattern about the layer in which new stable units are formed. An interesting open question is to examine whether a visual task that is based on similar low-level features would create stable units only at the higher layers.

Table 1. Average accuracy across all tasks for different connection growth methods.

Growth	10% Permutation	100% Permutation
Novel	97.3	97.3
Random	97.2	97.2
Transfer	97.3	97.0

4.4. Other Connection Growth Mechanisms

In section 3.7, we mentioned that type-1 connections (between plastic units) create novel paths while type-2 connections (from stable units to plastic units) promote forward transfer between tasks. NISPA randomly selects between these two connection types. In this section, we explore how different connection growth mechanisms perform depend-

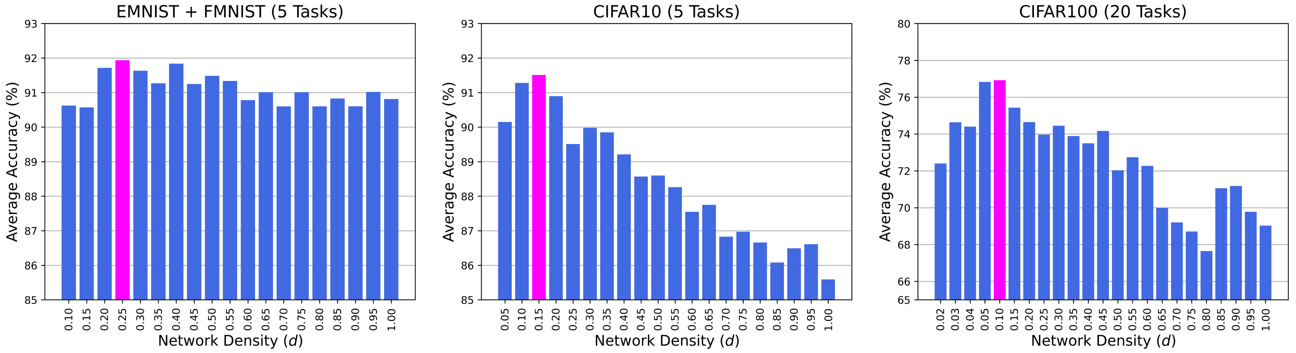

Figure 5. The effect of network density on NISPA's performance. Bars show the average accuracy across all tasks. The magenta bar shows the density level at which NISPA performs best.

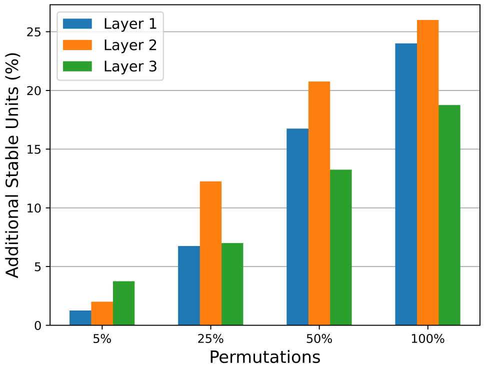

Figure 6. Bars show the percentage of additional stable units selected after the first task. The first task is standard MNIST classification, while the second task is the same but with permuted input pixels.

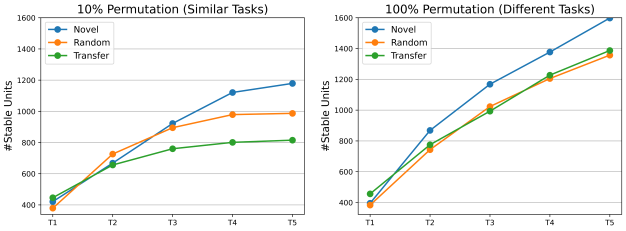

Figure 7. Growth in the number of stable units on permuted-MNIST tasks using different wiring algorithms.

ing on the similarity of consecutive tasks. We generate two task sequences, each with 5 tasks. The first sequence includes five permuted MNIST tasks, where in each task we randomly choose 10% of all input pixels and permute them. This corresponds to a sequence with quite similar tasks. In the second sequence, each task includes independent permutations of all pixels. Therefore, those tasks do not share common features.

In addition to NISPA's random connection growth mechanism, we consider the following two mechanisms. The first, referred to as *Novel*, only grows type-1 connections to create novel paths in the network. The second, referred to as *Transfer*, only grows type-2 connections. Details are given in the Appendix C.5.

Figure 7 shows the growth in the number of stable units, while **Table** 1 shows NISPA's performance on the two task sequences. First, note that growing only type-2 connections is beneficial if the tasks are similar because those connections promote forward transfer and reduce the number of new stable units required. However, the Transfer growth mechanism results in the worst accuracy when the tasks are quite different.

On the other hand, growing only type-1 connections increases rapidly the number of stable units even though that is not needed in terms of accuracy. Also, this Novel growth mechanism does not attempt to exploit similarities across tasks, and it achieves approximately the same accuracy on both sequences of this experiment.

These results suggest that we can tweak the growing mechanism and get better performance with fewer stable units as long as we have some prior knowledge about the similarity of different tasks. However, without such knowledge, NISPA's Random growth mechanism is a good tradeoff between exploiting forward transfer and creating novel network pathways.

4.5. NISPA in Class Incremental Learning

NISPA requires task labels to pick the correct classification head – lower layers are agnostic to task labels. This is significantly different than methods that utilize task information throughout the architecture, such as (Rusu et al., 2016; Mallya et al., 2018). Therefore, we claim that the main ideas in NISPA are not specific to task incremental learning.

Table 2. Average accuracy across all tasks once learning is complete (± one std.dev). The best model (excluding NI, which is an upper bound) is presented in bold.

Buffer Size	10 Samples Per Class			50 Samples Per Class			100 Samples Per Class		
	MNIST	EF-MNIST	CIFAR10	MNIST	EF-MNIST	CIFAR10	MNIST	EF-MNIST	CIFAR10
NI	97±0.2	82±0.3	80±0.2	97±0.2	82±0.3	80±0.2	97±0.2	82±0.3	80±0.2
NISPA-Replay	**77±1.2**	**66±1.4**	38 ±2.0	**86±1.9**	74±0.7	**51±1.5**	**90±0.6**	77±0.5	**57±0.8**
DER	57±2.0	64±0.9	19±4.7	84±1.3	**77±0.3**	30±9.9	87±0.9	**79±0.6**	33±1.2
ER	60±4.2	56±0.6	19±1.0	80±4.3	70±0.5	22±6.8	86±1.5	74±0.7	33±1.9
iCaRL	73±1.4	51±0.6	**40±2.7**	74±1.9	45±0.7	47±2.5	61±3.7	57±0.2	30±1.8
A-Gem	51±4.7	53±3.5	23±1.1	58±8.9	55±3.7	22±1.7	46±7.5	54±2.1	22±1.0

To support this claim, we present a variation of NISPA for Class Incremental Learning that we refer to as *NISPA-Replay*. First, NISPA-Replay does not freeze the connections in the final layer. Second, it utilizes a replay buffer that stores few random examples for each task, and train on those while learning a new task. Since all paths from inputs to the stable units of the penultimate layer are frozen, replay only affects the input weights of output units. We admit that replay of "raw" examples is not a biologically plausible approach – but it is still much simpler that other approaches for incremental continual learning based on complex sampling/replay or generative models (see Appendix D for details).

We benchmark NISPA-Replay against four well-known baselines that also rely on replay, namely, Experience Replay (ER) (Chaudhry et al., 2019b), Dark Experience Replay (DER) (Buzzega et al., 2020), iCaRL (Rebuffi et al., 2017), and A-GEM (Chaudhry et al., 2019a). Furthermore, the Non-Incremental (NI) model learns all classes simultaneously and serves as an upper bound. Baselines are randomly pruned to have the same number of parameters as NISPA for a fair comparison. For reference, we also present results for dense baselines in the Appendix **Table** 6.

Table 2 shows the average accuracy across all tasks for various buffer sizes. With few exceptions, NISPA outperforms other baselines across datasets and buffer sizes. This confirms that the main ideas in NISPA are promising even when task labels are not available during inference. On the other hand, the gap between the NI model (where there is no continual learning) and NISPA-Replay is quite large, suggesting that there is still plenty of space for improving the application of NISPA in the context of class incremental learning.

Note that DER outperforms NISPA on the EF-MNIST dataset. DER stores logits along with raw samples. Logits is a vector of 57 entries (for EF-MNIST), which is comparable to the size of those images. So DER has a valuable additional signal about the previous state of the network, which is not available to other baselines including NISPA. We observe that once the size of the logits signal becomes insignificant compared to the size of the input (e.g., in CIFAR10), the performance of DER drops considerably.

The second exception is iCaRL on CIFAR10, when we have 10 replay samples per class. This "tiny buffer" setting requires excellent use of the few stored examples, especially for natural images. Therefore, we attribute the success of iCaRL to its sophisticated sample selection strategy, while NISPA selects samples randomly.

5. Conclusions and Future Work

NISPA is a neuro-inspired approach for continual learning. To the best of our knowledge, it is the first method that works with a constant-density sparse network throughout the learning trajectory. Combining sparsity with rewiring, NISPA dominates state-of-the-art approaches on benchmark datasets with a large margin while having orders of magnitude less learnable parameters.

In future work, we aim to adapt NISPA to class incremental learning without requiring the replay of ”raw” examples. Second, we will explore strategies to “unfreeze” carefully selected stable units, when the number of remaining plastic units drops below a certain level, so that NISPA can keep learning new tasks while controlling the degree of forgetting for older tasks. Finally, although random connection growth is appropriate for exploring different network configurations, we aim to develop a more sophisticated approach that considers additional signals such as network-theoretic metric to maximize the benefit of growing new connections.

Acknowledgements

This work was supported by the National Science Foundation (Award: 2039741) and by the DARPA Lifelong Learning Machines (L2M) program of MTO (Cooperative Agreement HR0011-18-2-0019). The authors are grateful to the ICML 2022 reviewers and to Cameron E. Taylor, Qihang Yao, and Shreyas M. Patil for their constructive comments.

References

Aljundi, R., Chakravarty, P., and Tuytelaars, T. Expert gate: Lifelong learning with a network of experts. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

Aljundi, R., Babiloni, F., Elhoseiny, M., Rohrbach, M., and Tuytelaars, T. Memory aware synapses: Learning what (not) to forget. In *The European Conference on Computer Vision (ECCV)*, 2018.

Atkinson, C., McCane, B., Szymanski, L., and Robins, A. V. Pseudo-recursal: Solving the catastrophic forgetting problem in deep neural networks. *arXiv preprint*, abs/1802.03875, 2018.

Babadi, B. and Sompolinsky, H. Sparseness and expansion in sensory representations. *Neuron*, 83, 2014.

Bellec, G., Kappel, D., Maass, W., and Legenstein, R. Deep rewiring: Training very sparse deep networks. In *International Conference on Learning Representations*, 2018.

Buzzega, P., Boschini, M., Porrello, A., Abati, D., and Calderara, S. Dark experience for general continual learning: a strong, simple baseline. In *Advances in Neural Information Processing Systems*, volume 33, 2020.

Carletti, B. and Rossi, F. Neurogenesis in the cerebellum. *The Neuroscientist : a review journal bringing neurobiology, neurology and psychiatry*, 14, 2008.

Chaudhry, A., Dokania, P. K., Ajanthan, T., and Torr, P. H. Riemannian walk for incremental learning: Understanding forgetting and intransigence. In *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018.

Chaudhry, A., Ranzato, M., Rohrbach, M., and Elhoseiny, M. Efficient lifelong learning with a-GEM. In *International Conference on Learning Representations*, 2019a.

Chaudhry, A., Rohrbach, M., Elhoseiny, M., Ajanthan, T., Dokania, P. K., Torr, P. H. S., and Ranzato, M. Continual learning with tiny episodic memories. *arXiv preprint*, abs/1902.10486, 2019b.

Chechik, G., Meilijson, I., and Ruppin, E. Synaptic pruning in development: A computational account. *Neural computation*, 10, 1998.

Cichon, J. and Gan, W.-B. Branch-specific dendritic ca2+ spikes cause persistent synaptic plasticity. *Nature*, 520, 2015.

Cohen, G., Afshar, S., Tapson, J., and van Schaik, A. EMNIST: an extension of MNIST to handwritten letters. *arXiv preprint*, abs/1702.05373, 2017.

Cooke, S. and Bliss, T. Plasticity in human central nervous system. *Brain : a journal of neurology*, 129, 2006.

Deger, M., Helias, M., Rotter, S., and Diesmann, M. Spike-timing dependence of structural plasticity explains cooperative synapse formation in the neocortex. *PLOS Computational Biology*, 8, 2012.

Erhan, D., Bengio, Y., Courville, A., and Vincent, P. Visualizing higher-layer features of a deep network. *Technical Report, Univeristé de Montréal*, 2009.

Evci, U., Gale, T., Menick, J., Rivadeneira, P. S. C., and Elsen, E. Rigging the lottery: Making all tickets winners. In *International Conference of Machine Learning*, 2020.

French, R. M. Catastrophic forgetting in connectionist networks. *Trends in Cognitive Sciences*, 3, 1999.

Fu, M. and Zuo, Y. Experience-dependent structural plasticity in the cortex. *Trends in Neurosciences*, 34, 2011.

Georgiadis, G. Accelerating convolutional neural networks via activation map compression. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019.

Golkar, S., Kagan, M., and Cho, K. Continual learning via neural pruning. *arXiv preprint*, abs/1903.04476, 2019.

Grutzendler, J., Kasthuri, N., and Gan, W.-B. Long-term dendritic spine stability in the adult cortex. *Nature*, 420, 2002.

Guo, Y., Liu, Y., Oerlemans, A., Lao, S., Wu, S., and Lew, M. S. Deep learning for visual understanding: A review. *Neurocomputing*, 187, 2016.

Hadsell, R., Rao, D., Rusu, A. A., and Pascanu, R. Embracing change: Continual learning in deep neural networks. *Trends in Cognitive Sciences*, 24, 2020.

Hassabis, D., Kumaran, D., Summerfield, C., and Botvinick, M. Neuroscience-inspired artificial intelligence. *Neuron*, 95, 2017.

Hayashi-Takagi, A., Yagishita, S., Nakamura, M., Shirai, F., Wu, Y., Loshbaugh, A., Kuhlman, B., Hahn, K., and Kasai, H. Labelling and optical erasure of synaptic memory traces in the motor cortex. *Nature*, 525, 2015.

Hayes, T. L., Krishnan, G. P., Bazhenov, M., Siegelmann, H. T., Sejnowski, T. J., and Kanan, C. Replay in deep learning: Current approaches and missing biological elements. *arXiv preprint*, abs/2104.04132, 2021.

Hu, H., Peng, R., Tai, Y., and Tang, C. Network trimming: A data-driven neuron pruning approach towards efficient deep architectures. *arXiv preprint*, abs/1607.03250, 2016.

Isele, D. and Cosgun, A. Selective experience replay for lifelong learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

Jung, S., Ahn, H., Cha, S., and Moon, T. Continual learning with node-importance based adaptive group sparse regularization. In *Advances in Neural Information Processing Systems*, volume 33, 2020a.

Jung, S., Ahn, H., Cha, S., and Moon, T. Adaptive group sparse regularization for continual learning. *arXiv preprint*, abs/2003.13726, 2020b.

Kasai, H., Fukuda, M., Watanabe, S., Hayashi-Takagi, A., and Noguchi, J. Structural dynamics of dendritic spines in memory and cognition. *Trends in Neurosciences*, 33, 2010.

Kirkpatrick, J., Pascanu, R., Rabinowitz, N., Veness, J., Desjardins, G., Rusu, A. A., Milan, K., Quan, J., Ramalho, T., Grabska-Barwinska, A., Hassabis, D., Clopath, C., Kumaran, D., and Hadsell, R. Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences*, 114, 2017.

Krizhevsky, A., Nair, V., and Hinton, G. Cifar-10 (canadian institute for advanced research), a.

Krizhevsky, A., Nair, V., and Hinton, G. Cifar-100 (canadian institute for advanced research), b.

Kurtz, M., Kopinsky, J., Gelashvili, R., Matveev, A., Carr, J., Goin, M., Leiserson, W., Moore, S., Shavit, N., and Alistarh, D. Inducing and exploiting activation sparsity for fast neural network inference. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119, 2020.

LeCun, Y., Bengio, Y., and Hinton, G. Deep learning. *Nature*, 521, 2015.

Li, Z. and Hoiem, D. Learning without forgetting. *IEEE transactions on pattern analysis and machine intelligence*, 40, 2017.

Li, Z., Meng, M., He, Y., and Liao, Y. Continual learning with laplace operator based node-importance dynamic architecture neural network. In *International Conference on Neural Information Processing*, 2021.

Liu, J., Xu, Z., Shi, R., Cheung, R. C. C., and So, H. K. Dynamic sparse training: Find efficient sparse network from scratch with trainable masked layers. In *International Conference on Learning Representations*, 2020.

Liu, S., Mocanu, D. C., Matavalam, A. R. R., Pei, Y., and Pechenizkiy, M. Sparse evolutionary deep learning with over one million artificial neurons on commodity hardware. *Neural Computing and Applications*, 33, 2021a.

Liu, S., Yin, L., Mocanu, D. C., and Pechenizkiy, M. Do we actually need dense over-parameterization? in-time over-parameterization in sparse training. In *International Conference on Machine Learning*, 2021b.

Mallya, A., Davis, D., and Lazebnik, S. Piggyback: Adapting a single network to multiple tasks by learning to mask weights. In *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018.

McCloskey, M. and Cohen, N. J. Catastrophic interference in connectionist networks: The sequential learning problem. *Psychology of Learning and Motivation*, 1989.

Ming, G.-L. and Song, H. Adult neurogenesis in the mammalian brain: Significant answers and significant questions. *Neuron*, 70, 2011.

Mocanu, D., Mocanu, E., Stone, P., Nguyen, P., Gibescu, M., and Liotta, A. Scalable training of artificial neural networks with adaptive sparse connectivity inspired by network science. *Nature Communications*, 9, 2018.

Niethard, N., Burgalossi, A., and Born, J. Plasticity during sleep is linked to specific regulation of cortical circuit activity. *Frontiers in Neural Circuits*, 11, 2017.

Parisi, G. I., Kemker, R., Part, J. L., Kanan, C., and Wermter, S. Continual lifelong learning with neural networks: A review. *Neural Networks*, 113, 2019.

Park, J., Jung, S.-C., and Eun, S.-Y. Long-term synaptic plasticity: Circuit perturbation and stabilization. *The Korean Journal of Physiology & Pharmacology*, 18, 2014.

Ramapuram, J., Gregorova, M., and Kalousis, A. Lifelong generative modeling. *Neurocomputing*, 404, 2020.

Ramirez, S., Liu, X., Lin, P.-A., Suh, J., Pignatelli, M., Redondo, R., Ryan, T., and Tonegawa, S. Creating a false memory in the hippocampus. *Science (New York, N.Y.)*, 341, 2013.

Rebuffi, S.-A., Kolesnikov, A., Sperl, G., and Lampert, C. icarl: Incremental classifier and representation learning. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

Rhu, M., O'Connor, M., Chatterjee, N., Pool, J., Kwon, Y., and Keckler, S. Compressing dma engine: Leveraging activation sparsity for training deep neural networks. In *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2018.

Rolnick, D., Ahuja, A., Schwarz, J., Lillicrap, T., and Wayne, G. Experience replay for continual learning. In *Advances in Neural Information Processing Systems*, volume 32, 2019.

Rosenfeld, A. and Tsotsos, J. K. Incremental learning through deep adaptation. *IEEE transactions on pattern analysis and machine intelligence*, 42, 2018.

Rusu, A. A., Rabinowitz, N. C., Desjardins, G., Soyer, H., Kirkpatrick, J., Kavukcuoglu, K., Pascanu, R., and Hadsell, R. Progressive neural networks. *arXiv preprint*, abs/1606.04671, 2016.

Sahbi, H. and Zhan, H. FFNB: forgetting-free neural blocks for deep continual visual learning. *arXiv preprint*, abs/2111.11366, 2021.

Serra, J., Suris, D., Miron, M., and Karatzoglou, A. Overcoming catastrophic forgetting with hard attention to the task. In *International Conference on Machine Learning*, 2018.

Shin, H., Lee, J. K., Kim, J., and Kim, J. Continual learning with deep generative replay. *Advances in neural information processing systems*, 30, 2017.

Silver, D., Huang, A., Maddison, C., Guez, A., Sifre, L., Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T., and Hassabis, D. Mastering the game of go with deep neural networks and tree search. *Nature*, 529, 2016.

Sokar, G., Mocanu, D. C., and Pechenizkiy, M. Spacenet: Make free space for continual learning. *Neurocomputing*, 439, 2021.

van de Ven, G., Siegelmann, H., and Tolias, A. Brain-inspired replay for continual learning with artificial neural networks. *Nature Communications*, 11, 2020.

van de Ven, G. M. and Tolias, A. S. Three scenarios for continual learning. *arXiv preprint*, abs/1904.07734, 2019.

Xiao, H., Rasul, K., and Vollgraf, R. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint*, abs/1708.07747, 2017.

Yang, G., Pan, F., and Gan, W.-B. Stably maintained dendritic spines are associated with lifelong memories. *Nature*, 462, 2009.

Yang, G., Lai, C., Cichon, J., Ma, L., and Gan, W.-B. Sleep promotes branch-specific formation of dendritic spines after learning. *Science (New York, N.Y.)*, 344, 2014.

Yoon, J., Yang, E., Lee, J., and Hwang, S. J. Lifelong learning with dynamically expandable networks. In *International Conference on Learning Representations*, 2018.

Zeiler, M. D. and Fergus, R. Visualizing and understanding convolutional networks. In *European conference on computer vision*, 2014.

Zenke, F., Poole, B., and Ganguli, S. Continual learning through synaptic intelligence. In *International Conference on Machine Learning*, 2017.

Zuo, Y., Lin, A., Chang, P., and Gan, W.-B. Development of long-term dendritic spine stability in diverse regions of cerebral cortex. *Neuron*, 46, 2005.

A. NISPA Pseudocode

Algorithm 1 NISPA on task t (repeated for every task). Let M be the network and U be the set of all units in the network. Also, S, S_c, and P denote the set of all stable, candidate stable, plastic units, respectively. Furthermore, e is the number of epochs in a phase, a_f is the validation accuracy loss we are willing to accept, and k is the hyperparameter for scheduling τ (see Section 3.4).

```
Require: $S$, $P$, $M$, $e$, $a_f$, $k$                 # $S$, $P$, and $M$ are inherited from the previou task.
$a_{max} \leftarrow 0, \tau \leftarrow 1, p \leftarrow 1$     # $p$ is the phase index and $a_{max}$ is the best accuracy so far.
# Unit selection and rewiring cycle.
loop
    $M \leftarrow$ Train$(M, D_t, e)$                     # Train $M$ on $D_t$ for $e$ epochs.
    CacheM$[p] \leftarrow M$                              # Store the parameters of $M$.
    $a \leftarrow$ Validate$(M, V_t)$                     # Compute the validation accuracy $a$ on $V_t$
    $a_{max} \leftarrow \max(a_{max}, a)$                 # Update the $a_{max}$ if needed.
    # If stopping criterion (Section 3.8) is not met, perform selection/rewiring.
    if $a_{max} - a_f \leq a$ then
        $\tau \leftarrow \frac{1}{2}\left(1 + \cos\left(\frac{p \times \pi}{k}\right)\right)$   # Update $\tau$ using phase index $p$.
        $S_c \leftarrow$ SelectCandidates$(M, P, S, \tau)$   # Candidate unit selection as described in Section 3.3.
        CacheS$[p] \leftarrow S \cup S_c$                 # Store the set of stable units.
        $M \leftarrow$ Drop$(M, P, S, S_c)$               # Drop connections from $P^l$ to $S \cup S_c$ (Section 3.7).
        $M \leftarrow$ Grow$(M, P, S, S_c)$               # Randomly grow connections from $U$ to $P$ (Section 3.7).
        $p \leftarrow p + 1$                              # Increase the phase index.
    else
        $M \leftarrow$ CacheM$[p - 1]$                    # Revert the network to the end of the previous phase.
        $S \leftarrow$ CacheS$[p - 2]$                    # Restore saved stable units associated with the restored network.
        $M \leftarrow$ Freeze$(M, S)$                     # Freeze input connections to units in $S$.
        $M \leftarrow$ Reinit$(M, P)$                     # Reinitialize connections from $U$ to $P$.
        return $M$, $S$                                   # Start task $t + 1$.
    end if
end loop
```

B. Additional Results

B.1. Weight Re-initialization and τ-Schedules

Here we evaluate the following two aspects of NISPA's design. First, to re-initialize the plastic unit connections on task boundaries. Second, to decrease τ across successive phases based on a cosine annealing function, with steps of increasing size.

We compare NISPA with its ablated versions that do not re-initialize weights and that use a linear decrease schedule for τ (decrease with a constant step size of 0.05). **Table 3** shows the results for this comparison. We observe that re-initialization and the cosine annealing schedule are most effective when used together in both the MLP (EF-MNIST) and CNN (CIFAR100) architectures. The improvements are small but consistent.

Table 3. Average accuracy across five runs and all tasks (± 1 std.deviation). R and NR stand for "re-initialization" and "no re-initialization", respectively. Also, Cos and Lin stand for "cosine annealing" and "linear decrease", respectively.

Dataset	R+Cos	R+Lin	NR+Cos	NR+Lin
EF-MNIST	91.6±0.2	91.0±0.1	90.7±0.3	91.0±0.2
CIFAR100	75.9±1.3	74.7±1.0	74.9±0.6	75.5±0.5

B.2. Stability of NISPA

Table 4 presents the standard deviation of the average task accuracy across 5 runs. NISPA has the most stable performance compared to other CL baselines.

Table 4. Standard deviation of the average task accuracy across 5 runs.

Methods	EF-MNIST	CIFAR10	CIFAR100
NISPA	0.15	0.19	1.30
CLNP	0.18	0.67	5.14
EWC	2.11	2.43	2.00
SI	0.55	2.36	1.83
MAS	2.02	1.26	1.97

B.3. Comparison of Learnable Parameters

Table 5 compares the number of learnable parameters between NISPA and the baselines we consider. CLNP's exact number of parameters slightly varies during training because it drops some connections between tasks. For simplicity, we calculated all multipliers based on the initial number of parameters. We note that the final number of parameters in CLNP is still multiple times larger than NISPA. For example, in our experiments, CLNP has 73% density before starting task-5 on EF-MNIST, while NISPA has 20% density of throughout training.

The actual difference is not exactly $5\times$ and $10\times$ because we do not prune the bias terms and the first convolutional layer for NISPA (see Appendix C.3).

Table 5. Multipliers indicate the number of parameters compared to NISPA.

Methods	EF-MNIST	CIFAR10	CIFAR100
NISPA	$1\times$	$1\times$	$1\times$
CLNP	$4.94\times$	$9.94\times$	$9.97\times$
EWC	$4.94\times$	$9.94\times$	$9.97\times$
SI	$4.94\times$	$9.94\times$	$9.97\times$
MAS	$4.94\times$	$9.94\times$	$9.97\times$

B.4. Class Incremental Learning – Dense Baselines

In **Table** 6, we present class incremental results for dense baselines. Dense baselines have up to 10 times more learnable parameters than NISPA, so they do not represent a fair comparison.

C. Experimental Details

C.1. Datasets

In all datasets (CIFAR10, CIFAR100, MNIST, EMNIST, and FashionMNIST), we report the accuracy on the official test dataset and use 10% of the training dataset for validation. We perform early stopping (including early stopping at the phase level) based on validation accuracy. Likewise, we fine-tune the hyperparameters for NISPA and all baselines using the validation datasets. We have not performed any data augmentation.

C.2. Hyperparameters

We train all models using the Adam optimizer, unless noted otherwise. We tuned the hyperparameters of all baselines and report each method's best performance. Our hyperparameter search space included the suggested values for baselines, when available.

NISPA has the following hyperparameters:

- e: the number of epochs for each phase.

Table 6. Average accuracy across tasks and 5 runs (± 1 std.deviation). Dense baselines are denoted with the suffix ”-D” (e.g., DER-D or ER-D).

Buffer Size	10 Samples Per Class			50 Samples Per Class			100 Samples Per Class		
	MNIST	EF-MNIST	CIFAR10	MNIST	EF-MNIST	CIFAR10	MNIST	EF-MNIST	CIFAR10
NI-D	96 ±0.3	78±1.1	79±0.5	96±0.3	78±1.1	79±0.5	96±0.3	78±1.1	79±0.5
NI	97±0.2	82±0.3	80±0.2	97±0.2	82±0.3	80±0.2	97±0.2	82±0.3	80±0.2
NISPA-Replay	77±1.2	66±1.4	38 ±2.0	86±1.9	74±0.7	51±1.5	90±0.6	77±0.5	57±0.8
DER	57±2.0	64±0.9	19±4.7	84±1.3	77±0.3	30±9.9	87±0.9	79±0.6	33±1.2
ER	60±4.2	56±0.6	19±1.0	80±4.3	70±0.5	22±6.8	86±1.5	74±0.7	33±1.9
iCaRL	73±1.4	51±0.6	40±2.7	74±1.9	45±0.7	47±2.5	61±3.7	57±0.2	30±1.8
A-Gem	51±4.7	53±3.5	23±1.1	58±8.9	55±3.7	22±1.7	46±7.5	54±2.1	22±1.0
DER-D	73±4.0	71±0.4	25±1.0	90±1.2	80±0.4	31±10.7	93±2.0	81±0.4	39±14.5
ER-D	69±1.2	60±0.6	22±0.7	87±0.8	73±0.4	22±9.9	90±0.6	76±0.3	41±1.3
iCaRL-D	72±0.9	66±0.4	55±1.4	67±0.8	52±0.3	57±0.3	74±0.6	65±0.3	58±0.5
A-Gem-D	40±6.3	55±1.8	22±1.5	26±2.9	56±1.6	28±0.7	32±9.4	51±4.4	23±0.6

- a_f: the validation accuracy loss we are willing to accept between successive phases.
- k: the shape parameter of the cosine annealing function that governs the step size for τ.
- d: the per-layer density level. Note that if this is relatively high (e.g., 80% or more), NISPA does not guarantee a constant density level because growing a new connection for every dropped connection is not always possible.

NISPA's performance is not “fragile” with respect to any of these hyperparameters. They only control natural trade-offs. For example, decreasing a_f puts more emphasis on early tasks and results in more aggressive freezing, reducing the number of available units for future tasks. e and k determine the number of epochs. For instance, a smaller k decreases τ faster, resulting in fewer phases, and decreasing the number of epochs in which the model is trained. **Figure** 8 shows the shape of the cosine annealing function for various k values. Finally, a density between 0.05 and 0.2 works best for NISPA. Extremely low density hinders training since the network underfits the given tasks. On the other hand, dense networks have highly entangled units, making the isolation between tasks challenging. **Table** 7 summarizes all hyperparameter values used in this paper.

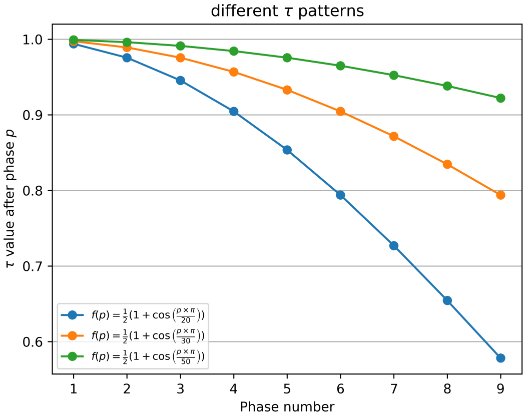

Figure 8. Cosine annealing function.

C.3. Weight Initialization and Sparsification

Weights are initialized using Kaiming's method. Specifically, they are sampled from a zero-centered Gaussian distribution with variance inversely proportional to each layer's width: $\sigma_l^2 = \frac{2}{N_l}$.

Table 7. Hyperparameters for NISPA for each task sequence. λ is an extra hyperparameter used only for replay in incremental class learning (see Section D).

Dataset	Learning Rate	Batch Size	e	a_f	k	d	λ
EMNIST + FashionMNIST	0.01	512	5	0.75	30	0.2	–
CIFAR100	0.002	64	5	2.5	30	0.1	–
CIFAR10	0.002	128	5	0.75	30	0.1	–
Permuted MNIST (5 tasks, 10%)	0.01	1024	2	0.75	40	0.1	–
Permuted MNIST (5 tasks, 100%)	0.01	1024	3	0.75	40	0.1	–
Permuted MNIST (2 tasks)	0.01	256	5	0.5	40	0.25	–
Replay MNIST (10 per class)	0.05	512	3	0.5	30	0.1	5
Replay EF-MNIST (10 per class)	0.05	128	5	1.5	30	0.3	0.5
Replay CIFAR10 (10 per class)	0.001	256	5	2	40	0.1	1
Replay MNIST (50 per class)	0.05	64	3	1	30	0.1	5
Replay EF-MNIST (50 per class)	0.1	256	5	2.5	30	0.3	0.5
Replay CIFAR10 (50 per class)	0.001	256	5	1	40	0.1	7.5
Replay MNIST (100 per class)	0.1	256	5	0.5	30	0.1	5
Replay EF-MNIST (100 per class)	0.05	128	5	2	30	0.3	1
Replay CIFAR10 (100 per class)	0.002	512	5	2	40	0.1	5

Newly grown connections are initialized to small values based on existing connection weights. Formally, for each layer l we compute the mean μ_l and the standard deviation σ_l of existing weights. Then, at layer l, we sample newly added connection weights as $\theta \sim \mathcal{N}_l(\mu_l, \sigma_l)$.

Networks are initially sparsified using unstructured random pruning on a per-layer basis. So, each connection of the same layer has equal chance of removal, and the density is the same across all layers.

The only exception is that we do not prune the first layer in convolutional architectures. Given a three-channel image input, units in the first layer are connected to the input via only three connections, and so if we prune that first layer, we will likely create a lot of ”dead units” (without any inputs) at the first layer. For example, at 0.1 density, units at the first layer will be dead with a probability $(1 - 0.1)^3 = 0.729$. Keeping the first layer dense avoids this problem. Note that this change has almost no effect on the total number of connections.

C.4. Permuted MNIST Experiments

In the permuted MNIST task sequences of Sections 4.3 and 4.4, we use MNIST classification as the first task. The following tasks are based on the same dataset but with permuted input features (i.e., MNIST pixels). For each permutation, we randomly selected p_r of the pixels and shuffle them.

In Section 4.3, we use an MLP with three hidden layers that consist of 400 units each. In Section 4.4, we consider an MLP with two hidden layers, each with 2000 units. This architectural change was necessary to ensure that (1) all models reach similar accuracy on all tasks, and (2) they always have enough plastic units to stabilize if needed. Otherwise, it is not possible to make a fair comparison (if a model picks fewer stable units but does not perform similarly with its counterparts).

C.5. Other Connection Growth Mechanisms

In Section 4.4, we experiment with two different connection growing mechanisms, namely, *Transfer* and *Novel*. The *Novel* mechanism randomly samples which connection to grow among type-1 candidate connections. On the other hand, the *Transfer* mechanism randomly samples among type-2 candidate connections.

In some corner cases, such as when the number of stable and plastic units is highly imbalanced, there may not be enough available candidates from one of these types. To make a fair comparison between different mechanisms, we need to ensure that all models have the same number of parameters (i.e., the same density). Therefore, we let each growth mechanism select the other connection type when needed to reach a certain density. This is a corner case that rarely happens in our experiments.

C.6. NISPA in Convolutional Neural Networks:

In CNNs, a “unit” represents a 3D convolution filter. Likewise, a “connection” represents a 2D kernel instead of a single weight. Therefore, when we perform an operation, such as dropping or freezing a connection, we consider the entire 2D kernel representing the connection between two units. Note that this formulation reduces the FLOP count immensely by eliminating operations performed on most 2D feature maps.

CNN units output a 2D matrix instead of a scalar value. So, we consider the sum of all entries of that matrix as the activation of the unit.

C.7. Handling “Dead Units”

The NISPA rewiring process can rarely result in units without any incoming (for stable units) or outgoing (for plastic units) connections. We refer to them as “dead units” and they are handled as follows:

- If a plastic unit lost all its outputs, we connect it to another plastic unit (randomly chosen) at the next layer.
- If a stable unit does not have any incoming connections from stable units, we degrade it to a plastic unit.

C.8. The Last Task

We do not assume that the number of tasks is known beforehand. If NISPA knew which task will be the final, it could select all free units as stable for that task, and use all the remaining capacity of the network to learn that final task as well as possible. We do not assume such knowledge, and so NISPA selects the smallest number of stable units so that it can continue learning new tasks in the future.

C.9. The Multi-Head Setting

In task-incremental learning, we use a multi-head setting at the output layer. However, instead of replacing the output head for each task, all heads are present from the start. Interference is avoided by masking out activations (during training and testing) depending on the task. This is effectively the same with the standard multi-head setting. Having all heads available from the start however makes the connection growth process simpler at the penultimate layer. For example, when we drop incoming connections to Task 1 output units (which are stable at that point), we can add the same number of connections from units at the penultimate layer to output heads of future tasks (that are still plastic at that point). When we reach the last task, there are no plastic units at the output layer anymore. So when we drop connections from that layer, we cannot add any connections back. This means that the density slightly decreases during the last task. This density decrease is negligible.

D. NISPA-Replay and Class Incremental Learning

In Section 4.5, we present a variation of NISPA called *NISPA-Replay* for incremental class learning. We evaluate this variation against four baselines using the following cumulative average accuracy metric. Suppose we have n tasks, and let a_j^{end} be the model accuracy evaluated on the held-out test set of the j-th task after learning all tasks. The average cumulative accuracy is: $\frac{1}{n}\sum_{i=1}^{n} a_i^{end}$.

D.1. NISPA-Replay

Training:
NISPA-Replay splits the loss terms for memory and task samples. Specifically, the loss function for task t becomes:

$$\mathbb{E}_{(x,y)\sim D_t}\left[\mathcal{L}_{CE}(y, f(x)\right] + \lambda\, \mathbb{E}_{(x,y)\sim B}\left[\mathcal{L}_{CE}(y, f(x))\right] \tag{4}$$

f is the model, D_t is the dataset of task t, B is the memory buffer, $\mathcal{L}_{CE}$ is the standard cross-entropy loss, and λ is a hyperparameter (see **Table** 7). In other words, for every gradient update, we get samples from the task dataset along with samples from the memory buffer, and perform one training step.

We omit the weight re-initialization step in NISPA-Replay for the following reason. Assigning random weights to existing units arbitrarily changes the activation of output units dedicated for future classes. This affects all units at the softmax layer and prevents trained output units (for previously seen classes) from making valid predictions.

Sampling:
We initialize a fixed-sized buffer with random samples. We ensure that each class seen so far has the same number of samples stored. We could implement a more sophisticated buffering strategy for NISPA-Replay. However, our goal here is to show that the good performance of NISPA-Replay in class incremental learning is due to the main ideas behind NISPA – and not due to any sophisticated methods we use for replay buffering or sample selection.

Datasets:
EF-MNIST and CIFAR10 class sequences are as presented in Section 4. The MNIST experiment is a standard split-MNIST sequence with five tasks, where each task consists of two consecutive digits.

Architectures:
For EF-MNIST and MNIST, we use an MLP, and for CIFAR10, we used a CNN. The MLP and CNN details are the same as in Section 4. Furthermore, NISPA-Replay and baselines are initially sparsified using uniform random pruning, as described in the main paper. The MLP density is 10% on MNIST, 30% on EF-MNIST, and the CNN density is 10% on CIFAR10. NISPA-Replay hyperparameters are given in **Table** 7.

Optimization:
We tuned all baselines searching the hyperparameter space for best performance. Also, we tried using both Adam and SGD optimizers. We observed that rehearsal baselines (ER, DER, iCaRL, A-GEM) perform best with SGD on all datasets. On the other hand, NISPA-Replay performs best with SGD on MNIST and EF-MNIST, and with Adam on CIFAR10.

D.2. Baselines

Experience Replay (ER): This simple replay algorithm sequentially learns new classes while also training on a small memory. It uses the same loss function as NISPA-Replay. This method can be thought of as a lower bound for any replay method. Surprisingly, it has been shown to outperform more complex CL approaches with memory replay (Buzzega et al., 2020; Chaudhry et al., 2019b).

Dark Experience Replay (DER): This recent method is a simple extension to ER (Buzzega et al., 2020), and it also replays raw samples. The only difference is that instead of storing hard class labels, it stores the logits of the trained model, and these logits are used as targets for replay samples. It has been shown to outperform seven well-known replay-based methods by a large margin on several datasets (Buzzega et al., 2020).

iCaRL: This is one of the most well-known replay methods (Rebuffi et al., 2017). It is novel in several aspects: clustering-based classification, sample selection, and loss function.

A-GEM: This method is different than other baselines as it stores raw samples but does not directly use them in training (Chaudhry et al., 2019a). Instead, it projects gradients of novel tasks based on gradients computed for memory samples. It also needs to store raw examples to compute gradients at a particular time in the training process.