Lifelong Neural Predictive Coding: Learning Cumulatively Online without Forgetting

Alexander G. Ororbia
Rochester Institute of Technology
Rochester, NY 14623, USA
ago@cs.rit.edu

Ankur Mali
University of South Florida
Tampa, FL 33620, USA
ankurarjunmali@usf.edu

C. Lee Giles
The Pennsylvania State University
State College, PA 16801, USA
clg20@psu.edu

Daniel Kifer
The Pennsylvania State University
State College, PA 16801, USA
duk17@psu.edu

Abstract

In lifelong learning systems based on artificial neural networks, one of the biggest obstacles is the inability to retain old knowledge as new information is encountered. This phenomenon is known as catastrophic forgetting. In this paper, we propose a new kind of connectionist architecture, the Sequential Neural Coding Network, that is robust to forgetting when learning from streams of data points and, unlike networks of today, does not learn via the popular back-propagation of errors. Grounded in the neurocognitive theory of predictive coding, our model adapts its synapses in a biologically-plausible fashion while another neural system learns to direct and control this cortex-like structure, mimicking some of the task-executive control functionality of the basal ganglia. In our experiments, we demonstrate that our self-organizing system experiences significantly less forgetting compared to standard neural models, outperforming a swath of previously proposed methods, including rehearsal/data buffer-based methods, on both standard (SplitMNIST, Split Fashion MNIST, etc.) and custom benchmarks even though it is trained in a stream-like fashion. Our work offers evidence that emulating mechanisms in real neuronal systems, e.g., local learning, lateral competition, can yield new directions and possibilities for tackling the grand challenge of lifelong machine learning.

1 Introduction

Lifelong learning is a part of machine learning and artificial intelligence research with the goal of developing a computational agent that can learn over time and continually accumulate new knowledge while retaining its previously learned experiences [1, 2]. For example, suppose an agent needs to learn to classify digits, then types of clothing, and then sketches of objects. As each new task arrives the agent is expected to process the accompanying data and learn the new task but still remember how to complete the old tasks, at least without significant degradation in performance or loss of generalization. Modern-day connectionist systems are typically trained on a fixed pool of data samples, collected in controlled environments, in isolation and random order, and then evaluated on a separate validation data pool. This is a far cry from what we really desire from learning machines.

When we look to humans or other animals, we see that they are more than capable of learning in a continual manner, making decisions based on sensorimotor input throughout their lifespans [3]. This ability to incrementally acquire and refine knowledge over long periods of time is driven by cognitive processes that come together to create the experience-driven specialization of motor and perceptual

36th Conference on Neural Information Processing Systems (NeurIPS 2022).

skills [4, 3]. Thus, evaluating how neural systems generalize on task sequences, as opposed to single, isolated tasks, proves to be a far greater challenge. In order to continually adapt, the brain must retain specific memories of prior tasks. In working towards the challenge of lifelong machine learning, this paper makes the following contributions: 1) we propose the sequential neural coding network, an interactive generative model that jointly reconstructs input and predicts its label, and an algorithm for updating its synapses, 2) we show that memory retention is vastly improved by integrating our model's multi-step nature with lateral competition that is driven by a task selection function inspired by the basal ganglia [5], and 3) we compare our model's performance against state-of-the-art baselines, including both regularization and rehearsal/replay-based methods, on publicly-available benchmarks.

2 Related Work

It is well-known that when artificial neural networks (ANNs) are trained on more than one task sequentially, the new information contained in subsequent tasks leads to catastrophic interference (a.k.a. catastrophic forgetting) with the information acquired in earlier tasks [6, 7, 8]. This happens in connectionist systems when the new data instances to be learned are significantly different from previously observed ones. This causes the new information to overwrite knowledge currently encoded in the system's synaptic weights, due to the sharing of neural representations over tasks [9, 6] (this is known as the representational overlap problem). In isolated, task-specific (offline) learning (though it still occurs [10]), this type of weight overwriting occurs to a lesser degree because the patterns are presented to the agent pseudo-randomly and multiple times, i.e., via multiple epochs.

Over the decades, there have been many approaches proposed to mitigate catastrophic forgetting in neural systems. Some of the earliest attempts proposed memory systems where prior data points were stored and regularly replayed to the network in a process called "rehearsal", which involved interleaving these data points with samples drawn from new datasets [11, 12, 13, 14, 15]. Though effective, the main drawback of these approaches is that they require explicitly storing old data. Such a mechanism is not known to exist in the brain and, as a practical matter, this leads to exploding working (hardware) memory requirements (inefficiency). In addition, rehearsal-based approaches do not offer any mechanisms to preserve consolidated knowledge in the face of acquiring new information [4]. Other approaches attempt to allocate additional neural "resources", i.e., growing the networks when required [16, 17], motivated by earlier findings [18]. However, this leads to dramatically increasing computational requirements over time as the networks grow larger. To compound these issues further, systems with growing capacity cannot know how many resources to allocate at a given time since the number of tasks and samples are not known a priori (without imposing strong assumptions on the input distribution). Other approaches try to block old information from being overwritten through regularization [19]. From this vast collection of research, each approach bearing strengths and weaknesses, three suggested remedies have emerged: 1) allocate additional neural resources to accommodate new knowledge, 2) use non-overlapping representations (or semi-distributed ones [20]) if resources are fixed, and 3) interleave old patterns with new ones as new information is acquired.

In this work, we consider the setting where the space available to the agent grows slowly compared to the rate of new tasks being presented. This means that we cannot just create a new, separate network for every task observed in the stream. Furthermore, storing, reshuffling, and re-presenting data in the stream is not feasible in this setting. Concretely, our approach could be classified as class incremental learning (Class-IL) [21] given that we **do not use task identifiers** at both training and test time.In addition to addressing the above, our contribution to lifelong learning is motivated by the premise that developing algorithms [22, 23, 24, 25, 26] that serve as alternatives to back-propagation will lead to the creation of promising architectures with mechanisms equipped to tackle catastrophic interference.

3 Cumulative Learning with Neural Coding

Notation: We start by defining key notation.$\odot$ indicates a Hadamard product while $\cdot$ denotes matrix/vector multiplication. $(\mathbf{v})^T$ is the transpose of $\mathbf{v}$. Matrices/vectors are depicted in bold-font, e.g., matrix $\mathbf{M}$ or vector $\mathbf{v}$ (scalars shown in italic). Finally, $||\mathbf{v}||_2$ denotes the Euclidean norm of $\mathbf{v}$.

3.1 Sequential Learning and the Data Continuum

This work focuses on adapting a neural system in the context of sequential learning. Starting from an early definition [27] of this form of learning, we assume that there is a sequence of tasks $\mathcal{T}_1, \mathcal{T}_2, \mathcal{T}_3, \ldots$ (with potentially no end) with each task presented to a system in order. When faced with the $(N+1)$th task, the system should use the knowledge that it has gained from the previous N tasks to aid in learning and performing the current task. The knowledge of a system is stored in a knowledge base (KB), e.g., the synapses of a neural model. Each task $\mathcal{T}_i$ has its own corresponding dataset $D_i = \{(\mathbf{y}_1, \mathbf{x}_1, t_i) \ldots (\mathbf{y}_{n_i}, \mathbf{x}_{n_i}, t_i)\}$ with n_i examples. Here $\mathbf{x}_j \in \mathcal{R}^{J_x \times 1}$ represents the feature vector of the jth example (J_x is its dimensionality), $\mathbf{y}_j \in \{0,1\}^{J_y \times 1}$ is the target (label), and t_i is the task descriptor that identifies $(\mathbf{y}_j, \mathbf{x}_j)$ as being a data point from task i. The task descriptor is one-hot encoded as $\mathbf{t} \in \{0,1\}^{(N+1)\times 1}$ and, when a new task is encountered, the size of the one-hot vector is increased by one – thus the network adds an extra randomly initialized input node if it accepts t_i as an extra input. Note that while we present the task data continuum with t_i explicitly depicted, we will generalize our models to not depend on t_i (making this problem task descriptor-free).

Dynamic Output Units: The output nodes in our setting get re-used for each new task. For example, output node 1 of the network could represent a prediction for the digit "1" in task $\mathcal{T}_1$ (e.g., digit recognition), while in task $\mathcal{T}_2$ (e.g., clothing recognition) the same node could represent a prediction for "pants". If a new task has more classes than previous tasks, we add output nodes with randomly initialized weights. For example, if prior tasks were binary and the new task has 4 targets, we add 2 more outputs to the network. Note this is a difficult form of cumulative learning [28].

Context Units: For the model that we develop in this study, when task t is encountered, for each layer ℓ in the network, we make use of a task embedding vector $\mathbf{g}_t^\ell$ (this new memory is much smaller than creating a new network for task t, which would require new weight matrices per layer rather than an extra vector). All $\mathbf{g}_t^\ell \in \mathcal{R}^{J_\ell \times 1}$ (J_ℓ is the number of units in layer ℓ) are stored in memory matrices $\mathbf{M}^\ell \in \mathcal{R}^{(N+1)\times J_\ell}$ ($\mathcal{M} = \{\mathbf{M}^1, ...\mathbf{M}^L\}$), where a context can be retrieved using a one-hot encoding of the task descriptor, i.e., $\mathbf{M}^\ell \cdot \mathbf{t}$ ($\mathbf{t}$ will be produced by another system – see Section 3.3).

3.2 The Interactive Generative Model

The sequential neural coding network (S-NCN) is designed to make flexible conditional predictions – e.g., predicting $\mathbf{y}$ given $\mathbf{x}$, predicting both $\mathbf{y}$ and $\mathbf{x}$, predicting missing parts of $\mathbf{x}$ given $\mathbf{y}$ and the observed parts of $\mathbf{x}$, etc. In order to do so, it treats inputs/outputs in a non-standard way. The input to the model is the set of task contexts $\{\mathbf{g}_t^1, ..., \mathbf{g}_t^L\}$ and the output units represent $(\mathbf{y}, \mathbf{x})$. To predict $\mathbf{y}_i$ given $\mathbf{x}_i$, we clamp the output nodes responsible for predicting $\mathbf{x}$, forcing their output to be $\mathbf{x}_i$.[1] During training, outputs are clamped to both $\mathbf{y}_i$ and $\mathbf{x}_i$, forcing the S-NCN to update latent states and synapses. The S-NCN can also operate as a probabilistic generative network by feeding in a random noise vector as input, but we leave this extension to future work (and focus on predicting $\mathbf{y}$ given $\mathbf{x}$).

The full computational process of the S-NCN is defined by three key steps: 1) layer-wise hypothesis generation, 2) state error-correction, and 3) synaptic weight evolution. The first two steps iteratively predict and correct the representations of the model for the input and target values of the task. After K iterations, model weights and the current task context memory are updated. In this section, we provide details of the above steps and then describe the objective function that our model optimizes.

3.2.1 Inference: Predicting and Correcting States

Layer-wise State Prediction. The architecture of the S-NCN can be viewed as a stack of parallel, stateful neural-based prediction layers $P_1, ..., P_\ell, ..., P_m$, where the goal of each predictor is to guess the internal state of the predictor in the layer below (i.e., the S-NCN is not a feedforward network). The state of P_ℓ (layer ℓ) is represented by the (zero-initialized) vector $\mathbf{z}^\ell \in \mathcal{R}^{J_\ell \times 1}$ (J_ℓ is the number of units in layer ℓ). P_ℓ makes a prediction $\mathbf{z}_\mu^{\ell-1}$ about the current state $\mathbf{z}^{\ell-1}$ of $P_{\ell-1}$ (layer $\ell - 1$). Furthermore, we let $\mathbf{z}_x^0$ and $\mathbf{z}_y^0$ denote clamped outputs. That is, if we want to predict the label $\mathbf{y}_i$ given the features $\mathbf{x}_i$, we set $\mathbf{z}_x^0 = \mathbf{x}_i$ and if we want to train, we set both $\mathbf{z}_x^0 = \mathbf{x}_i$ and $\mathbf{z}_y^0 = \mathbf{y}_i$. The

[1] Similarly, in the case of missing data, we can ask the network to predict the $\mathbf{y}_i$ and the missing parts of $\mathbf{x}_i$ given the observed parts of $\mathbf{x}_i$ by clamping outputs to only the observed parts of $\mathbf{x}_i$. See appendix for details.

values predicted by layer 1 are denoted by $\mathbf{z}^0_{\mu,x}$ and $\mathbf{z}^0_{\mu,y}$. Note that $\mathbf{z}^0 = [\mathbf{z}^0_x, \mathbf{z}^0_y]$, where the two are concatenated (if either is missing, the SNCN completes the required values – see Appendix).

With respect to neural structure, the parallel predictors of the S-NCN are locally connected through (forward) generative weights $\mathbf{W}^\ell \in \mathcal{R}^{J_\ell \times J_{\ell+1}}$ and error (feedback) weights $\mathbf{E}^\ell \in \mathcal{R}^{J_{\ell+1} \times J_\ell}$, which work to transmit error information to relevant regions of neural processing elements, effectively coordinating all of the S-NCN's predictors. Formally, a predictor (which embodies the computation of deep excitatory pyramidal neurons), with state $\mathbf{z}^{\ell+1}$, that guesses the state of $\mathbf{z}^\ell$, takes on the following form (given matrix $\mathbf{W}^{\ell+1}$ and activation $\phi^{\ell+1}$):

$$\mathbf{z}^\ell_\mu = \mathbf{W}^{\ell+1} \cdot \phi^{\ell+1}(\mathbf{z}^{\ell+1}), \quad \mathbf{e}^\ell = (\mathbf{z}^\ell - \mathbf{z}^\ell_\mu) \tag{1}$$

where $\mathbf{e}^\ell$ is a block of error units. Error units are paired with each predictor. Their task is to compute the disagreement or mismatch between the predictor's output and the target activity pattern $\mathbf{z}^\ell$ (embodying the mismatch computation carried out by superficial pyramidal neurons). The error unit vector $\mathbf{e}^\ell$ can also be derived from the total discrepancy reduction presented in Equation 5 (see Appendix). Note that for layer 0, $\mathbf{e}^0 = [\mathbf{e}^0_x, \mathbf{e}^0_y]$ (there are error neurons for $\mathbf{x}_i$ as well as others for $\mathbf{y}_i$, the concatenation of which makes up the bottom-most prediction error signal).

Latent State Correction and Context Updating. Once each layer ℓ has made a prediction about the layer below it and error units have been activated, the state of layer ℓ can be adjusted to take into account the local top-down and bottom-up error information. Using its current state and the error nodes (along with its task context embedding $\mathbf{g}^\ell_t$), layer ℓ in the S-NCN adjusts its state $\mathbf{z}^\ell$ as follows:

$$\mathbf{z}^\ell(k) = f^\ell(\mathbf{z}^\ell(k-1) + \beta \mathbf{d}^\ell, \mathbf{g}^\ell_t), \quad \text{where } \mathbf{d}^\ell = -\mathbf{e}^\ell + \mathbf{E}^\ell \cdot \mathbf{e}^{\ell-1} \tag{2}$$

where β is the state correction rate and k marks one step of the S-NCN's K-step inference process. Note that $\mathbf{d}^\ell$ is the perturbation that adjusts the values of the hidden states – it combines a top-down expectation of layer ℓ with a bottom-up pressure from layer $\ell - 1$. The error feedback weights $\mathbf{E}^\ell$ are parameters that play a crucial role in this calculation, as they are responsible for transmitting the error at ℓ back up to layer $\ell + 1$. Notably, part of the state-correction requires competition among the individual units in a given layer through the function $f^\ell(\mathbf{z}^\ell, \mathbf{g}^\ell_t)$. There are various ways in which this function can be implemented and, in this study, we implemented it as follows:

$$f^\ell(\mathbf{z}^\ell, \mathbf{g}^\ell_t) = \left(\mathbf{I} \odot \mathbf{V}^\ell\right) \cdot \mathbf{z}^\ell, \text{where, } \mathbf{V}^\ell = \text{BKWTA}(\mathbf{g}^\ell_t, K) \cdot \text{BKWTA}((\mathbf{g}^\ell_t)^T, K)$$

where $\text{BKWTA}(\mathbf{v}, K)$ is the binarized K winners-take-all function, yielding a binary vector with 1 at the index of each of the K winning units, or formally:

$$\text{BKWTA}(\mathbf{v}, K) = \left\{1 \text{ if } v_j \in \{K \text{ largest elements of } \mathbf{v}\} \text{ and } \phi(\mathbf{v}) = 0 \text{ otherwise}\right\}.$$

In the Appendix, we study other forms of the competition function (we found that the above performed best, so we report this in the main paper). Note that this lateral inhibition is a function of evolving context $\mathbf{g}^\ell$, triggered by a task pointer t_i (produced by a task selector model, which we define later).

In real neural systems, intra-layer competition between units is thought to facilitate contextual processing [29], where only some neuronal signals are strengthened while the activity of others is suppressed. Moreover, lateral competition, often classically modeled with anti-Hebbian learning [30], encourages the formation of sparse codes [31, 32]. Since the S-NCN is an interactive model, incorporating lateral competition is natural and computed online, highlighting its flexibility compared to ANNs.[2] Crucially, this lateral activity is an inductive bias that leads the S-NCN to acquire task-dependent representations that encode information for multiple, disjoint tasks. In a sense, this task specialization that we build into the neural dynamics is similar in spirit to activation sharpening [33].

3.2.2 Updating Synaptic Parameters

Given that the S-NCN is an interactive network [34], inferring its states requires running Equations 1 and 2 K times, where the model alternates between making predictions and then correcting states once error units have been computed. Once latent states have been inferred, the S-NCN then adjusts its synaptic values. The synaptic updates take the form of Hebbian-like rules:

$$\Delta \mathbf{W}^\ell = \mathbf{e}^\ell \cdot (\phi^\ell(\mathbf{z}^{\ell+1}))^T \odot \mathbf{S}^\ell_W, \text{ and, } \Delta \mathbf{E}^\ell = \alpha\left(\mathbf{d}^{\ell+1} \cdot (\mathbf{e}^\ell)^T\right) \odot \mathbf{S}^\ell_E \tag{3}$$

[2] Recurrent weights could model lateral activity in ANNs but this would require using backprop through time.

Algorithm 1 State inference procedure.

Input: sample $(\mathbf{y}, \mathbf{x}, \mathbf{t})$, β, $\mathcal{M}$, & Θ
function INFERSTATES($(\mathbf{y}, \mathbf{x}, \mathbf{t}), \Theta$)
  $(\mathbf{g}^1, ..., \mathbf{g}^\ell, ..., \mathbf{g}^L) \leftarrow getContexts(\mathbf{t}, \mathcal{M})$
  Set $\mathbf{z}^1, ..., \mathbf{z}^\ell, ..., \mathbf{z}^L$ to $\mathbf{0}$
  $\mathbf{e}^L = \mathbf{0}$, $\mathbf{z}^0_x = \mathbf{x}$, $\mathbf{z}^0_y = \mathbf{y}$
  for $k = 1$ to K **do**
    for $\ell = L$ to 1 **do** ▷ Run layerwise predictors
      if $\ell > 1$ **then** ▷ Latent prediction layer
        $\mathbf{z}^{\ell-1}_\mu = \mathbf{W}^\ell \cdot \phi^\ell(\mathbf{z}^\ell)$
        $\mathbf{e}^{\ell-1} = (\mathbf{z}^{\ell-1} - \mathbf{z}^{\ell-1}_\mu)$
      else ▷ Sensory prediction layer
        $\mathbf{z}^{\ell-1}_{\mu,x} = \mathbf{W}^\ell_x \cdot \phi^\ell(\mathbf{z}^\ell)$
        $\mathbf{e}^{\ell-1}_x = (\mathbf{z}^{\ell-1}_x - \mathbf{z}^{\ell-1}_{\mu,x})$
        $\mathbf{z}^{\ell-1}_{\mu,y} = \mathbf{W}^\ell_y \cdot \phi^\ell(\mathbf{z}^\ell)$
        $\mathbf{e}^{\ell-1}_y = (\mathbf{z}^{\ell-1}_y - \mathbf{z}^{\ell-1}_{\mu,y})$
    for $\ell = L$ to 1 **do** ▷ Correct internal states
      if ℓ == 1 **then**
        $\mathbf{d}^\ell = -\mathbf{e}^\ell + \mathbf{E}^\ell_x \cdot \mathbf{e}^{\ell-1}_x + \mathbf{E}^\ell_y \cdot \mathbf{e}^{\ell-1}_y$
      else
        $\mathbf{d}^\ell = -\mathbf{e}^\ell + \mathbf{E}^\ell \cdot \mathbf{e}^{\ell-1}$
      $\mathbf{z}^\ell = f^\ell\Big(\phi^\ell(\mathbf{z}^\ell) + \beta \mathbf{d}^\ell, \mathbf{g}^\ell\Big)$
  $\Lambda = (\mathbf{z}^1, ..., \mathbf{z}^\ell, ..., \mathbf{z}^L, \mathbf{d}^1, ..., \mathbf{d}^\ell, ..., \mathbf{d}^L,$
    $\mathbf{e}^0, ..., \mathbf{e}^\ell, ..., \mathbf{e}^{L-1})$
  Return Λ

Algorithm 2 Weight update computation.

Input: Λ, λ, γ, $\mathcal{M}$, & Θ
function UPDATEWEIGHTS(Λ, Θ)
  // Calculate weight displacements
  for $\ell = L$ to 2 **do**
    $\Delta\mathbf{W}^\ell = \Big(\mathbf{e}^{\ell-1} \cdot (\phi^\ell(\mathbf{z}^\ell))^T\Big) \odot \mathbf{S}^\ell_W$
    $\Delta\mathbf{E}^\ell = \gamma(\mathbf{d}^\ell \cdot (\mathbf{e}^{\ell-1})^T) \odot \mathbf{S}^\ell_E$
  $\Delta\mathbf{W}^1_x = \Big(\mathbf{e}^0_x \cdot (\phi^1(\mathbf{z}^1))^T\Big) \odot \mathbf{S}^1_{W,x}$
  $\Delta\mathbf{E}^1_x = \gamma(\mathbf{d}^1 \cdot (\mathbf{e}^0_x)^T) \odot \mathbf{S}^1_{E,x}$
  $\Delta\mathbf{W}^1_y = \Big(\mathbf{e}^0_y \cdot (\phi^1(\mathbf{z}^1))^T\Big) \odot \mathbf{S}^1_{W,y}$
  $\Delta\mathbf{E}^1_y = \gamma(\mathbf{d}^1 \cdot (\mathbf{e}^0_y)^T) \odot \mathbf{S}^1_{E,y}$

  // Update current weights
  for $\ell = L$ to 2 **do**
    $\mathbf{W}^\ell = \mathbf{W}^\ell + \lambda\Delta\mathbf{W}^\ell$
    $\mathbf{E}^\ell = \mathbf{E}^\ell + \lambda\Delta\mathbf{E}^\ell$
  $\mathbf{W}^1_x = \mathbf{W}^1_x + \lambda\Delta\mathbf{W}^1_x$
  $\mathbf{E}^1_x = \mathbf{E}^1_x + \lambda\Delta\mathbf{E}^1_x$
  $\mathbf{W}^1_y = \mathbf{W}^1_y + \lambda\Delta\mathbf{W}^1_y$
  $\mathbf{E}^1_y = \mathbf{E}^1_y + \lambda\Delta\mathbf{E}^1_y$
  Update $(\mathbf{g}^1, \cdots, \mathbf{g}^L, \mathcal{M})$ via Eqn. 4
  // Return new weights
  $\Theta = \{\mathbf{W}^1_x, \mathbf{W}^1_y, ..., \mathbf{W}^\ell ... \mathbf{W}^L,$
    $\mathbf{E}^1_x, \mathbf{E}^1_y, ..., \mathbf{E}^\ell, ..., \mathbf{E}^L\}$
  Return Θ

where α is a scaling factor, usually set to < 1.0, that makes the error feedback weights change at a slower rate than the forward weights (this improves convergence [35]). $\mathbf{S}^\ell_W \in \mathcal{R}^{J_\ell \times J_{\ell+1}}$ and $\mathbf{S}^\ell_E \in \mathcal{R}^{J_{\ell+1} \times J_\ell}$ are modulation factors that provide stability to the weight updates (see Appendix).

An important property of the above weight update rules is that they are local – to compute changes in the synapses, all we require is the information immediately available to the neuron(s) of interest (making these rules function similarly to classical Hebbian updates [36], although there are important differences to them, as discussed in [35]). Since a neuron is able to immediately generate a hypothesis given its own internal state, without requiring the active generation of other predictors, and its error can be readily computed after prediction by comparing to the current state of the target neuron state, the weight updates of any predictor layer may be computed in parallel to others. This would allow us to allocate dedicated computing cores to particular predictors, or "pieces", of the S-NCN. Observe that the S-NCN does not require activation derivatives in any of its computations (this is neurobiologically more realistic and favorable for specialized hardware implementations).

Furthermore, during the learning process, each context vector $\mathbf{g}^\ell_t$ and its corresponding memory matrix is adjusted according to the following simple contrastive rule:

$$\mathbf{g}^\ell_{t+1} = \mathbf{g}^\ell_t + \eta_e \mathbf{d}^\ell - \eta_g(\mathbf{g}^\ell_t - \frac{1}{t-1}\sum_{j=1}^{t-1}(\mathbf{g}^\ell_j)), \text{ and, } \Delta\mathbf{M}^\ell = \mathbf{g}^\ell_{t+1} * (\mathbf{t}_j)^T \tag{4}$$

where η_e modulates a long-term memory update using the current perturbation to be applied to layer ℓ. η_g controls the repulsion term, which "pushes" context codes away from each other (for diversity). These adapted codes, which influence inter-neuronal competition in a task-sensitive manner, could be viewed as a simplification of distributed temporal context [37], where contiguity, i.e., recall/generation of one item is influenced by the presence of another, is introduced into S-NCN distributed processing.

The pseudocode illustrating how the elements described so far are combined in an S-NCN system is presented in Algorithms 1 and 2. The transmission of bottom-up and top-down errors in the S-NCN is motivated by the theory of predictive processing [38, 39, 40, 41, 42, 43, 44] and classical work on interactive networks [34, 45, 46], where models undergo a settling process to process input stimuli more than once (see Appendix for intuition). Though this requires extra computation, the process endows the network with desirable properties, e.g., the ability to conduct constraint satisfaction [47, 48]. By using a multi-step, laterally-competitive processing scheme, the S-NCN is able to "select" subnetworks (portions of neurons) for specific tasks, reducing representational overlap and,

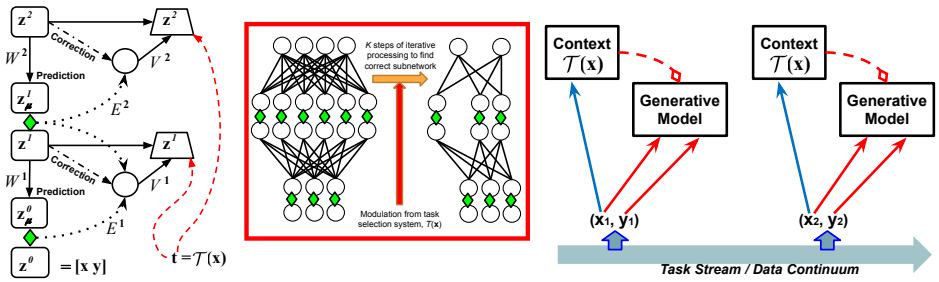

Figure 1: (Left) The sequential neural coding network is shown processing an input $(\mathbf{y}, \mathbf{x})$ over one step, given a task context $\mathbf{t}$ (produced by the task selection model $\mathcal{T}(\mathbf{x})$), overriding the current $\mathbf{z}^\ell$ after error-correction and application of lateral inhibition matrices $\{\mathbf{V}^1, \mathbf{V}^2\}$. Inside the red box is shown one possible emergent subnetwork. Green diamonds indicate error units. (Right) The full complementary neural system shown processing patterns from a data continuum.

ultimately, forgetting. This selection is driven by the task pointer $\mathbf{t}$, produced by the task selector, $\mathcal{T}(\mathbf{x})$, the final piece of the S-NCN system, which we describe in Section 3.3.

3.2.3 Objective Function

During training, when presented with stimulus $(\mathbf{y}_i, \mathbf{x}_i, \mathbf{t}_i)$, the S-NCN adjusts its internal states and synapses so that the output $\mathbf{z}^0_\mu$ of layer 1 (P_1) is as close as possible to $(\mathbf{y}_i, \mathbf{x}_i)$. It does this by minimizing *total discrepancy* (ToD) [49] – a measure of its total internal disorder,which is the sum of all mismatches between layerwise guesses and actual states. Formally, the ToD for an S-NCN is:

$$\mathcal{L}(\Theta) = \sum_{\ell=0}^{L-1} \mathcal{L}^\ell(\Theta^\ell), \text{ where } \mathcal{L}^\ell(\Theta^\ell) = \frac{1}{2}(||\mathbf{z}^\ell - \mathbf{z}^\ell_\mu||_2)^2. \tag{5}$$

Θ contains all of the synaptic parameters, i.e., $\Theta = \{\mathbf{W}^1, \mathbf{E}^1, \cdots, \mathbf{W}^L, \mathbf{E}^L\}$ and $\Theta^\ell = \{\mathbf{W}^\ell, \mathbf{E}^\ell\}$.

The above loss decomposes the problem of credit assignment in the S-NCN into sub-problems with each sub-problem focusing on the comparison between the prediction $\mathbf{z}^\ell_\mu$ made by layer $\ell + 1$ and the actual state value $\mathbf{z}^\ell$ of layer ℓ. The resulting updates to each state $\mathbf{z}^\ell$, as well as relevant synapses, will then depend on a bottom-up transmitted error signal and the top-down influence of the mismatch with the expectation of the predictor immediately above [49, 35]. While we have motivated aspects of our model from a neuro-cognitive perspective, the error units and weight updates can be derived from the total discrepancy function above [35] (and cast as approximately minimizing free energy [50]).

3.3 The Neural Task Selection Model

Earlier, we described the S-NCN as taking in a task signal $\mathbf{t}$ that drives the system's task context memory $\mathbf{g}^\ell$ for layer ℓ (i.e., the routine $getContexts(\mathbf{t}, \mathcal{M})$ in Algorithm 1). To create this task pointer, we design a second neural circuit that we call the task selector $\mathcal{T}(\mathbf{x})$ – this will allow the S-NCN to automatically decide when a new task has been encountered and to determine, at test time, what task observed data points belong to. The motivation behind $\mathcal{T}(\mathbf{x})$ comes from the neuroscientific idea that the basal ganglia plays the role of information routing (among other roles), which serves as a form of task selection [51, 52]). In other words, it selects/enables various cognitive programs stored in other cortical regions [5]. As a result, we develop a type of complementary learning system (CLS) (different than that of [53], which models an interaction between the hippocampus and neocortex), which pairs our cortex-like model with a basal ganglia-like model. We refer to our task selector $\mathcal{T}(\mathbf{x})$ as the "functional neural basal ganglia" (FNBG) in order to emphasize that the actual basal ganglia in the human brain is more complex and does more than what our computational model does. Our FNBG model, built with competitive learning, has two roles: 1) task shift detection - deciding whether data from an input stream indicates the presence of a new task, and 2) task recognition - deciding whether incoming data requires switching to an existent task context or creating a new one.

Task Shift Detection: In order to detect the occurrence of a new task while processing data from the pattern stream, $\mathcal{T}(\mathbf{x})$ utilizes the output error neurons of the generative model described earlier to detect spikes in their values which might indicate distributional drift. Specifically, $\mathcal{T}(\mathbf{x})$ maintains an

exponentially weighted running mean $\mu_{\mathcal{L}}$ and variance $\sigma^2_{\mathcal{L}}$ of the norm of the cortex model's label error neurons $\mathbf{e}^0_y$. The necessary statistics are calculated as follows:

$$\Delta = ||\mathbf{e}^0_y||^2_2 - \mu_{\mathcal{L}}(t-1) \tag{6}$$

$$\mu_{\mathcal{L}}(t) = \mu_{\mathcal{L}}(t-1) + \eta\Delta \tag{7}$$

$$\sigma^2_{\mathcal{L}}(t) = (1-\eta)\sigma^2_{\mathcal{L}}(t-1) + \eta(\Delta)^2 \tag{8}$$

with $\eta = 0.1$ (determined by preliminary experiments). Using the above dynamic statistics, a shift is detected by determining if the following inequality evaluates to true (repeatedly for a series of 5 consecutive batches): $\mu_{\mathcal{L}}(t) > \mu_{\mathcal{L}}(t-1) + 2\sqrt{\sigma^2_{\mathcal{L}}(t-1)}$. Upon detection of a boundary, we suppress the check until 1000 samples have been seen after the last detected shift (creating a refractory period to allow the competitive learning model, described next, to acquire enough data to learn).

Task Recognition through Competitive Learning: To conduct task recognition, $\mathcal{T}(\mathbf{x})$ first randomly projects the input $\mathbf{x}$ to a low-dimensional space ("key") $\mathbf{k} = \mathbf{R} \cdot \mathbf{x}$ ($\mathbf{R}$ is initialized from a Gaussian distribution). We then update a rolling average estimate of the streaming input using this generated key as follows: $\mathbf{k}_\mu = (1-\tau)\mathbf{k}_\mu + \tau\mathbf{k}$ (with $\tau = 0.65$). Finally, with the matrix $\mathbf{Q}$, the FNBG maps this rolling representation $\mathbf{k}_\mu$ to a decision as to which task context the S-NCN generative cortex is to utilize, i.e., $\hat{\mathbf{t}} = \text{BKWTA}(\mathbf{Q} \cdot (\mathbf{k}_\mu/||\mathbf{k}_\mu)||_2, K = 1)$.The task pointer is then used to retrieve contexts $\{\mathbf{g}^1_t, ..., \mathbf{g}^L_t\}$ where $\mathbf{g}^\ell = \mathbf{M} \cdot \hat{\mathbf{t}}$ (implementing $getContexts(\mathbf{t} = \hat{\mathbf{t}}, \mathcal{M})$).

To update the FNBG weights, while also avoiding catastrophic interference in $\mathcal{T}(\mathbf{x})$ itself, we propose a biologically-inspired learning rule based on competitive learning. Specifically, we develop what we call "guided competitive learning", since during the act of processing a stream of certain samples from the task that we know that we are operating on, we also know which neuron out of a set of $T-1$ task output neurons should be selected. This leads to the following update rule formally defined as:

$$\Delta\mathbf{Q} = -(\rho\mathbf{t}) \cdot (\mathbf{k}_\mu/||\mathbf{k}_\mu||_2)^T \tag{9}$$

where ρ is the competitive weight adjustment factor (a value we found works well in the range of $[0.5, 1]$). Note that $\mathbf{t} = \hat{\mathbf{t}}$ recovers an unsupervised classical competitive Hebbian learning. However, we force the model to a specific task pointer value by using $\mathbf{t}$, the one-hot encoding of the dynamic integer variable t maintained by the FNBG itself, initialized as $t = 0$. Every time a task shift is detected according to Equations 6-8, this dynamic variable is incremented by one, i.e., $t \leftarrow t + 1$.

For both task recognition and the FNBG's synaptic update, note that the rolling representation $\mathbf{k}_\mu$ is normalized by its Euclidean norm so that we may utilize a vectorizable form of competitive learning based on dot products (taking advantage of GPU-based matrix multiplication). In essence, we take the (normalized) rolling average representation of the input stream for a given task, compute its dot-product with all currently-available task weight vectors, and choose the dot product with maximal value as the winner. Finally, we re-normalize the matrix Q by its column Euclidean norms after each of its updates, i.e., $Q[:, i] = (Q[:, i] + \Delta Q[:, i])/||Q[:, i]||_2$ where $Q[:, i]$ indicates the extraction of all values in column i from Q (this normalization is similar to that of adaptive resonance theory [54]).

3.4 Putting It All Together: A Complementary System

At a high level, given the above, the full S-NCN complementary system, depicted in Figure 1 (Right), consists of: 1) a task selection model (inspired by the information routing/executive control behavior of the basal ganglia [5]) which creates the task contexts that laterally inhibit/gate the activities of the generative S-NCN, and 2) a generative model that learns to predict inputs/labels given a task context. In essence, the FNBG $\mathcal{T}(\mathbf{x})$ takes in $\mathbf{x}_j$ to produce $\hat{\mathbf{t}}$ (a one-hot representation of a task pointer t_i) which is then fed into the generative S-NCN (along with $\mathbf{x}$ and $\mathbf{y}$) to compute predictions. [3]

4 Experiments

Simulation Setup: In our experiments, we train models with three hidden layers, whether they be multilayer perceptrons (MLPs) or S-NCNs and compare against baselines from the literature. All models were restricted to contain (a maximum of) 500 units per layer. For the S-NCN, weights were initialized from a Gaussian distribution scaled by each layer's fan-in and were optimized using

[3] Please see the Appendix where we summarize symbols, notation, and abbreviation used in this work.

stochastic gradient descent with learning rate of $\lambda = 0.01$. Baseline models were trained on each task for 40 epochs while the S-NCN only made a single pass. The output layer for each MLP was a maximum entropy classifier and the objective was to minimize Categorical cross entropy – in the S-NCN, this was encoded in its label error neurons $\mathbf{e}_y^0$. (See Appendix for experimental details, computing infrastructure, hyper-parameter details, and code details.)

Evaluation Metrics: To measure model generalization over the sequence of tasks, we make use of the resulting task matrix R (as in [55]), an $N \times N$ matrix of task accuracy scores (normalized to $[0, 1]$), where in this study $N = T$. We measure average accuracy (ACC) (mean performance across tasks) and backward transfer (BWT). BWT measures the influence that learning a task T_t has on the performance of task $T_k < T_t$. A positive BWT indicates that a learning task T_t increases performance on preceding task T_k. As such, higher BWT is better and a strongly negative BWT means there is stronger (more catastrophic) forgetting. Mean and standard deviation (10 trials) are reported for ACC and mean (10 trials) for BWT (see Appendix for its standard deviation). The formulas for ACC, BWT, and a new set of metrics we created to analyze memory retention, are provided in the Appendix.

4.1 The Multi-Dataset Task Stream

To start, we tested the S-NCN model on a complex task sequence composed of several learning benchmarks [4].We create task sequences by breaking apart MNIST (*M*), Fashion MNIST (*FM*), and Google Draw (*GD*) each into two "sub-tasks" (e.g., for MNIST, *M1* and *M2*), or portions of data with a particular subset of the original dataset's classes. See the Appendix for details on sub-tasks/task sequence creation. In Table 1, we present two task orderings (Ordering #1 is "High Color Sim." and Ordering #2 is "Low Color Sim.") each under two conditions: sub-tasks that have 1) an equal number of classes (5 each), and 2) an unequal number of classes (number of classes was chosen randomly, omitting 5 as an option). The number of classes was sampled once and held constant for all trials.

We evaluate our proposed S-NCN system (as well as four variations of it in the Appendix) – hyperparameters were $\beta = 0.05$, $K = 10$, $\eta_g = 0.9$, $\eta_e = 0.01$, $\alpha = 0.98$). The baselines include an MLP trained only with backprop (Backprop), Elastic Weight Consolidation (EWC) [19], the Mode-IMM method [56], the Md-IMM method combined with either DropTransfer (DT+Md-IMM) or both L2-Transfer and DropTransfer (L2-DT-Mode-IMM) [56], and the competitive model, hard attention to task (HAT) [57]. For each baseline, we tuned hyper-parameters based on their accuracy on each task's development set. See Appendix for additional baseline results.

Discussion: Results are reported in Table 1 (see Appendix for more results). Each simulation was run 10 times (each trial used a unique seed).As observed in our results, **we see that the S-NCN outperforms all baselines consistently, in terms of ACC and BWT, exhibiting improved memory retention over modern, backprop-centric baselines** (and, in the Appendix, our expanded results show that the FNBG-driven lateral inhibition is key to improving memory retention the most). This result is robust across both task sequences and equal/unequal class settings. Meta-parameters for the S-NCNs were only tweaked minorly, with the same values used in all scenarios. The observation that lateral inhibition improves the neural computation (in our model) also corroborates the result of [48].

4.2 Continual Learning Benchmarks

To connect our model to current learning results, we experimented with a wide swath of approaches on three benchmarks – Split MNIST, Split NotMNIST, and Split Fashion MNIST (FMNIST). Furthermore, we compare to multi-head (below dashed line in Table 2) and single-head models (above dashed line). We compare the S-NCN to both replay/rehearsal and non-replay methods: naïve rehearsal with memory (NR+M), EWC, synaptic intelligence (SI) [58], MAS [59], Lwf [60], GEM [61], ICarl [62], Lucir [63], and Mnemonics [64] (additional baselines can be found in the Appendix).

Discussion: In Table 2, we report ACC and BWT, averaged over 10 trials, offering a comprehensive comparison of methods and demonstrating that, for all three benchmarks, **the proposed S-NCN outperforms all of them in terms of ACC**, and nearly all in terms of BWT (and on par with GDumb and GEM, the difference in BWT being negligible) demonstrating the power afforded by designing

[4] S-NCN is single head model as opposed to HAT ,GEM and others that use multiple-head

Table 1: Generalization metrics (10 trials) for sequence orderings # 1 & #2 (higher values are better).

	Ordering #1: $\{M1, M2, GD1, FM1, FM2, GD2\}$ **(High Color Sim.)**			
	Equal		**Unequal**	
	ACC	**BWT**	**ACC**	**BWT**
Backprop	0.241 ± 0.050	-0.759 ± 0.030	0.185 ± 0.048	-0.791 ± 0.048
EWC	0.280 ± 0.023	-0.714 ± 0.030	0.185 ± 0.046	-0.726 ± 0.039
Md-IMM	0.521 ± 0.027	-0.392 ± 0.023	0.480 ± 0.039	-0.240 ± 0.040
DT+Md-IMM	0.530 ± 0.024	-0.387 ± 0.021	0.551 ± 0.042	-0.220 ± 0.042
L2+DT+Md-IMM	0.532 ± 0.025	-0.237 ± 0.027	0.520 ± 0.040	-0.240 ± 0.045
HAT	0.550 ± 0.019	-0.211 ± 0.020	0.492 ± 0.031	-0.231 ± 0.036
S-NCN (ours)	$\mathbf{0.716 \pm 0.013}$	$\mathbf{-0.031 \pm 0.017}$	$\mathbf{0.713 \pm 0.011}$	$\mathbf{-0.041 \pm 0.012}$
	Ordering #2: $\{GD2, M1, FM2, M2, GD1, FM1\}$ **(Low Color Sim.)**			
Backprop	0.303 ± 0.030	-0.644 ± 0.037	0.287 ± 0.043	-0.671 ± 0.043
EWC	0.303 ± 0.031	-0.643 ± 0.033	0.291 ± 0.039	-0.663 ± 0.047
Md-IMM	0.584 ± 0.027	-0.091 ± 0.030	0.533 ± 0.034	-0.230 ± 0.036
DT+Md-IMM	0.591 ± 0.020	-0.088 ± 0.032	0.528 ± 0.036	-0.211 ± 0.039
L2+DT+Md-IMM	0.630 ± 0.029	-0.076 ± 0.030	0.551 ± 0.037	-0.201 ± 0.041
HAT	0.596 ± 0.026	-0.114 ± 0.029	0.563 ± 0.031	-0.210 ± 0.044
S-NCN (ours)	$\mathbf{0.721 \pm 0.014}$	$\mathbf{-0.042 \pm 0.013}$	$\mathbf{0.667 \pm 0.011}$	$\mathbf{-0.097 \pm 0.013}$

Table 2: Generalization metrics (10 trials) for Split MNIST, Split Fashion MNIST (FMNIST) and Not-MNIST benchmarks. Above dashed line are multi-head models & below are single-head models.

	MNIST		**Fashion MNIST**		**NotMNIST**	
	ACC	**BWT**	**ACC**	**BWT**	**ACC**	**BWT**
EWC	0.760 ± 0.030	-0.210	0.739 ± 0.020	-0.201	0.790 ± 0.020	-0.176
VCL	0.980 ± 0.210	-0.003	0.980 ± 0.20	-0.002	0.953 ± 0.003	-0.004
IMM	0.951 ± 0.018	-0.007	0.950 ± 0.013	-0.005	0.925 ± 0.011	-0.006
HAT	0.972 ± 0.011	-0.040	0.968 ± 0.011	-0.004	0.942 ± 0.009	-0.005
GEM	0.922 ± 0.110	$\mathbf{+0.001}$	0.930 ± 0.12	$+0.001$	0.919 ± 0.021	-0.003
A-GEM	0.950 ± 0.09	$\mathbf{+0.001}$	0.955 ± 0.11	$+0.001$	0.925 ± 0.020	-0.002
ER	0.938 ± 0.06	$\mathbf{-0.002}$	0.945 ± 0.10	-0.004	0.927 ± 0.016	-0.004
EWC	0.190 ± 0.030	-0.357	0.199 ± 0.06	-0.350	0.186 ± 0.020	-0.361
NR+M	0.950 ± 0.470	-0.100	0.948 ± 0.380	-0.090	0.880 ± 0.028	-0.103
SI	0.197 ± 0.110	-0.367	0.198 ± 0.100	-0.370	0.161 ± 0.030	-0.370
MAS	0.195 ± 0.290	-0.340	0.180 ± 0.250	-0.340	0.178 ± 0.060	-0.341
Lwf	0.846 ± 0.340	-0.120	0.875 ± 0.300	-0.130	0.626 ± 0.091	-0.130
ICarl	0.940 ± 0.410	-0.100	0.960 ± 0.400	-0.110	0.887 ± 0.102	-0.109
Lucir	0.940 ± 0.310	-0.103	0.950 ± 0.340	-0.110	0.935 ± 0.093	-0.101
GDumb	0.978 ± 0.09	$\mathbf{-0.005}$	0.973 ± 0.09	-0.006	0.940 ± 0.080	-0.004
Mnem	0.960 ± 0.320	-0.091	0.968 ± 0.300	$\mathbf{+0.007}$	0.950 ± 0.071	-0.011
S-NCN	$\mathbf{0.981 \pm 0.300}$	-0.005	$\mathbf{0.982 \pm 0.400}$	-0.003	$\mathbf{0.957 \pm 0.400}$	$\mathbf{-0.004}$

models with stronger grounding in neurobiology. Furthermore, it is promising to see that the S-NCN outperforms/matches performance with not only the single-head models but also with multi-head models (except GEM), which enjoy an easier version of the problem, i.e., they can utilize a different classifier per task. Finally, note that the S-NCN, due to the FNBG-driven competition, learns to compose task contexts in a data-dependent manner. Desirably, the S-NCN is a single-head model, meaning that it does not grow out a separate softmax classifier per task (as in multi-headed models, e.g., HAT, IMM, GEM), which means that it tackles the harder form of the forgetting problem, learning representations that preserve knowledge across disjoint tasks. Furthermore, our model is online, whereas models such as IMM or SI require multiple passes per dataset, and does not require validation data in order to run an expensive neural architecture search (NAS) as in [65].

4.3 Discussion: On Model Limitations

Although our results with respect to catastrophic forgetting are promising, the S-NCN does have several limitations that are important to discuss. First and foremost, the S-NCN attempts to learn a generative model of the joint distribution over inputs and labels, i.e., $p(\mathbf{y}, \mathbf{x})$, which has (classically) been shown to be a much more difficult problem compared to directly learning a conditional mapping between inputs and labels [66]. Second, to learn $p(\mathbf{y}, \mathbf{x})$, the S-NCN adapts to data through an expectation-maximization process where, given an input, it must iteratively infer a maximum a

posteriori (MAP) estimate of its latent neural activities over a K-step stimulus window. Despite the fortunate fact that, for the benchmarks studied in this work, values for K were fairly low (only 10 to 20 steps were needed per sample/mini-batch), for more complex data types, such as natural images with multiple objects and background scenery, the value of K will quite likely need to be much higher, increasing the S-NCN's computation time when conducting its online inference. We remark that this drawback could be mitigated by integrating mechanisms to support amortized inference, e.g., as in predictive sparse decomposition [67], and by designing custom software/hardware implementations that exploit the S-NCN's layer-wise parallelism (which would also work in asynchronous settings) in both its inference and weight updating cycles. Future work should explore adapting the S-NCN to only learning a conditional mapping as opposed to a full joint distribution as well as develop mechanisms for pre-training the generative side of the system (which would allow freezing of the generative synapses and only require updating discriminative ones – this could reduce the value of K for more complex sensory inputs). While the S-NCN's objective of minimizing ToD is important for breaking free of backprop and its limitations, offering benefits such as resolving weight transport, eschewing the need for derivatives, facilitating local learning without the need for a global feedback pathway, inferring richer stateful representations, it also creates a more challenging optimization problem, i.e., the model must match the values imposed by input data as well as ensure that its internal activities and local predictions are aligned. While the full complementary system largely mitigates catastrophic interference, this primarily benefits measurements of BWT and could potentially damage the model's per-task performance on more complex datasets, i.e., the main diagonal of task matrix R. Since we do not impose any distributional assumptions over the latent activities (such as a clean Gaussian prior as in variational autoencoders), if the S-NCN's estimated value of the latent activities is far from their true posterior distribution, then the S-NCN might experience sub-optimal performance in complex setups. Even though all continual learning systems suffer from this issue (especially most modern-day ANN-based ones), our model's dual optimization nature could experience this problem more frequently. We believe that integrating memory-aware retrieval driven by (a brain-like form of) replay, which would exploit the S-NCN's generative nature, could be a plausible solution moving forward (helping the system come closer to its true posterior and avoiding bad local optima).

4.4 On Negative Societal Impacts

The potential negative societal impact of the proposed S-NCN complementary system is indirect – while the model and algorithmic framework we develop is foundational in nature, it could potentially affect the myriad of applications/systems currently in development today or those to be developed in the future. As a result, at best, the same negative consequences that result from using backprop-based ANNs are still present when using the S-NCN training process instead. At worst, given that we have presented promising results across several lifelong learning benchmarks, the S-NCN could facilitate the development of better-performing robotic agents that might be used in military applications that might result in the loss of life, such as drones. Despite the benefits that S-NCN offers to the statistical learning and cognitive neuroscience communities, one should consider the drawbacks of building powerful neural systems to drive applications. To safeguard against its potential negative impact, we emphasize that developing an ethical framework to guide the design and training of general intelligent systems will be important for ensuring safe integration into human society.

5 Conclusion

In this paper, we proposed the sequential neural coding network (S-NCN), an interactive generative model, and its local learning procedure for lifelong machine learning. As demonstrated on several benchmarks and setups, this model retains the knowledge it acquires from prior tasks when learning new ones in the face of task data streams, primarily when lateral inhibition, driven by a self-organizing task selection model, sharpens neural activities within its layers. As a result, this work marks an important step towards developing brain-inspired agents that are capable of robustly combating catastrophic forgetting, especially ones that do not require task descriptors to guide the learning process, offering a promising pathway towards achieving the ultimate goal of designing agents that act and adapt in ways more similar to animals and humans.

Acknowledgements

We would like to thank Alexander Ororbia (Sr.) for useful discussions related to cognitive types, a concept that served as key motivation for the task contexts that drove lateral inhibition in this work.

References

[1] S. Thrun and T. M. Mitchell, "Lifelong robot learning," *Robotics and autonomous systems*, vol. 15, no. 1-2, pp. 25–46, 1995.

[2] G. I. Parisi, R. Kemker, J. L. Part, C. Kanan, and S. Wermter, "Continual lifelong learning with neural networks: A review," *arXiv preprint arXiv:1802.07569*, 2018.

[3] J. D. Power and B. L. Schlaggar, "Neural plasticity across the lifespan," *Wiley Interdisciplinary Reviews: Developmental Biology*, vol. 6, no. 1, 2017.

[4] F. Zenke and W. Gerstner, "Hebbian plasticity requires compensatory processes on multiple timescales," *Phil. Trans. R. Soc. B*, vol. 372, no. 1715, p. 20160259, 2017.

[5] G. Leisman, O. Braun-Benjamin, and R. Melillo, "Cognitive-motor interactions of the basal ganglia in development," *Frontiers in systems neuroscience*, vol. 8, p. 16, 2014.

[6] M. McCloskey and N. J. Cohen, "Catastrophic interference in connectionist networks: The sequential learning problem," *The psychology of learning and motivation*, vol. 24, no. 109, p. 92, 1989.

[7] R. Ratcliff, "Connectionist models of recognition memory: constraints imposed by learning and forgetting functions." *Psychological review*, vol. 97, no. 2, p. 285, 1990.

[8] S. Lewandowsky, "On the relation between catastrophic interference and generalization in connectionist networks," *Journal of Biological Systems*, vol. 2, no. 03, pp. 307–333, 1994.

[9] R. M. French, "Catastrophic forgetting in connectionist networks," *Trends in cognitive sciences*, vol. 3, no. 4, pp. 128–135, 1999.

[10] M. Toneva, A. Sordoni, R. T. d. Combes, A. Trischler, Y. Bengio, and G. J. Gordon, "An empirical study of example forgetting during deep neural network learning," *arXiv preprint arXiv:1812.05159*, 2018.

[11] A. Robins, "Catastrophic forgetting in neural networks: the role of rehearsal mechanisms," in *Artificial Neural Networks and Expert Systems, 1993. Proceedings., First New Zealand International Two-Stream Conference on*. IEEE, 1993, pp. 65–68.

[12] R. Anthony, "Catastrophic forgetting, rehearsal and pseudorehearsal," *Connection Science*, vol. 7, no. 2, pp. 123–146, 1995.

[13] A. Robins, "Consolidation in neural networks and in the sleeping brain," *Connection Science*, vol. 8, no. 2, pp. 259–276, 1996.

[14] A. Gepperth and C. Karaoguz, "A bio-inspired incremental learning architecture for applied perceptual problems," *Cognitive Computation*, vol. 8, no. 5, pp. 924–934, 2016.

[15] S.-A. Rebuffi, A. Kolesnikov, and C. H. Lampert, "icarl: Incremental classifier and representation learning," in *Proc. CVPR*, 2017.

[16] A. A. Rusu, N. C. Rabinowitz, G. Desjardins, H. Soyer, J. Kirkpatrick, K. Kavukcuoglu, R. Pascanu, and R. Hadsell, "Progressive neural networks," *arXiv preprint arXiv:1606.04671*, 2016.

[17] G. I. Parisi and S. Wermter, "Lifelong learning of action representations with deep neural self-organization," in *The AAAI 2017 Spring Symposium on Science of Intelligence: Computational Principles of Natural and Artificial Intelligence, Standford, US*, 2017, pp. 608–612.

[18] E. Meier, L. Hertz, and A. Schousboe, "Neurotransmitters as developmental signals," *Neurochemistry international*, vol. 19, no. 1-2, pp. 1–15, 1991.

[19] J. Kirkpatrick, R. Pascanu, N. Rabinowitz, J. Veness, G. Desjardins, A. A. Rusu, K. Milan, J. Quan, T. Ramalho, A. Grabska-Barwinska *et al.*, "Overcoming catastrophic forgetting in neural networks," *Proceedings of the National Academy of Sciences*, vol. 114, no. 13, pp. 3521–3526, 2017.

[20] R. M. French, "Semi-distributed representations and catastrophic forgetting in connectionist networks," *Connection Science*, vol. 4, no. 3-4, pp. 365–377, 1992.

[21] G. M. Van de Ven and A. S. Tolias, "Three scenarios for continual learning," *arXiv preprint arXiv:1904.07734*, 2019.

[22] J. R. Movellan, "Contrastive hebbian learning in the continuous hopfield model," in *Connectionist Models*. Elsevier, 1991, pp. 10–17.

[23] B. Scellier and Y. Bengio, "Equilibrium propagation: Bridging the gap between energy-based models and backpropagation," *Frontiers in computational neuroscience*, vol. 11, p. 24, 2017.

[24] D.-H. Lee, S. Zhang, A. Fischer, and Y. Bengio, "Difference target propagation," in *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer, 2015, pp. 498–515.

[25] A. G. Ororbia and A. Mali, "Biologically motivated algorithms for propagating local target representations," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, 2019, pp. 4651–4658.

[26] S. Bartunov, A. Santoro, B. Richards, L. Marris, G. E. Hinton, and T. Lillicrap, "Assessing the scalability of biologically-motivated deep learning algorithms and architectures," in *Advances in Neural Information Processing Systems*, 2018, pp. 9390–9400.

[27] S. Thrun, "Is learning the n-th thing any easier than learning the first?" in *Advances in neural information processing systems*, 1996, pp. 640–646.

[28] G. Fei, S. Wang, and B. Liu, "Learning cumulatively to become more knowledgeable," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2016, pp. 1565–1574.

[29] H. Adesnik and M. Scanziani, "Lateral competition for cortical space by layer-specific horizontal circuits," *Nature*, vol. 464, no. 7292, p. 1155, 2010.

[30] P. Földiak, "Forming sparse representations by local anti-hebbian learning," *Biological cybernetics*, vol. 64, no. 2, pp. 165–170, 1990.

[31] B. A. Olshausen and D. J. Field, "Sparse coding with an overcomplete basis set: A strategy employed by v1?" *Vision research*, vol. 37, no. 23, pp. 3311–3325, 1997.

[32] A. D. Szlam, K. Gregor, and Y. L. Cun, "Structured sparse coding via lateral inhibition," in *Advances in Neural Information Processing Systems*, 2011, pp. 1116–1124.

[33] R. M. French, "Using semi-distributed representations to overcome catastrophic forgetting in connectionist networks," in *Proceedings of the 13th annual cognitive science society conference*. Erlbaum, 1991, pp. 173–178.

[34] J. L. McClelland and D. E. Rumelhart, "An interactive activation model of context effects in letter perception: I. an account of basic findings." *Psychological review*, vol. 88, no. 5, p. 375, 1981.

[35] A. Ororbia, A. Mali, C. L. Giles, and D. Kifer, "Continual learning of recurrent neural networks by locally aligning distributed representations," *IEEE Transactions on Neural Networks and Learning Systems*, 2020.

[36] D. O. Hebb *et al.*, "The organization of behavior," 1949.

[37] M. W. Howard and M. J. Kahana, "A distributed representation of temporal context," *Journal of Mathematical Psychology*, vol. 46, no. 3, pp. 269–299, 2002.

[38] R. P. Rao and D. H. Ballard, "Dynamic model of visual recognition predicts neural response properties in the visual cortex," *Neural computation*, vol. 9, no. 4, pp. 721–763, 1997.

[39] R. R. PN and B. D. H, "Predictive coding in the visual cortex: a functional interpretation of some extra-classical receptive-field effects." *Nature neuroscience*, vol. 2, no. 1, 1999.

[40] Y. Huang and R. P. Rao, "Predictive coding," *Wiley Interdisciplinary Reviews: Cognitive Science*, vol. 2, no. 5, pp. 580–593, 2011.

[41] C. Wacongne, J.-P. Changeux, and S. Dehaene, "A neuronal model of predictive coding accounting for the mismatch negativity," *Journal of Neuroscience*, vol. 32, no. 11, pp. 3665–3678, 2012.

[42] R. Chalasani and J. C. Principe, "Deep predictive coding networks," *arXiv preprint arXiv:1301.3541*, 2013.

[43] A. Clark, *Surfing uncertainty: Prediction, action, and the embodied mind*. Oxford University Press, 2015.

[44] E. Santana, M. S. Emigh, P. Zegers, and J. C. Principe, "Exploiting spatio-temporal structure with recurrent winner-take-all networks," *IEEE Transactions on Neural Networks and Learning Systems*, 2017.

[45] J. L. McClelland, "The grain model: A framework for modeling the dynamics of information processing," *Attention and Performance (Volc. XIV): Synergies in Experimental Psychology, Artificial Intelligence, and Cognitive Neuroscience.*, 1993.

[46] R. C. O'Reilly, "Biologically plausible error-driven learning using local activation differences: The generalized recirculation algorithm," *Neural computation*, vol. 8, no. 5, pp. 895–938, 1996.

[47] O. R. C, "Six principles for biologically based computational models of cortical cognition," *Trends in cognitive sciences*, vol. 2, no. 11, pp. 455–462, 1998.

[48] R. C. O'Reilly, "Generalization in interactive networks: The benefits of inhibitory competition and hebbian learning," *Neural Computation*, vol. 13, no. 6, pp. 1199–1241, 2001.

[49] A. G. Ororbia II, P. Haffner, D. Reitter, and C. L. Giles, "Learning to adapt by minimizing discrepancy," *arXiv preprint arXiv:1711.11542*, 2017.

[50] K. Friston, "The free-energy principle: a rough guide to the brain?" *Trends in cognitive sciences*, vol. 13, no. 7, pp. 293–301, 2009.

[51] E. Yehene, N. Meiran, and N. Soroker, "Basal ganglia play a unique role in task switching within the frontal-subcortical circuits: evidence from patients with focal lesions," *Journal of Cognitive Neuroscience*, vol. 20, no. 6, pp. 1079–1093, 2008.

[52] T. J. Buschman and E. K. Miller, "Goal-direction and top-down control," *Philosophical Transactions of the Royal Society B: Biological Sciences*, vol. 369, no. 1655, p. 20130471, 2014.

[53] J. L. McClelland, B. L. McNaughton, and R. C. O'reilly, "Why there are complementary learning systems in the hippocampus and neocortex: insights from the successes and failures of connectionist models of learning and memory." *Psychological review*, vol. 102, no. 3, p. 419, 1995.

[54] S. Grossberg, "Adaptive resonance theory: How a brain learns to consciously attend, learn, and recognize a changing world," *Neural networks*, vol. 37, pp. 1–47, 2013.

[55] D. Lopez-Paz *et al.*, "Gradient episodic memory for continual learning," in *Advances in Neural Information Processing Systems*, 2017, pp. 6470–6479.

[56] S. Lee, J. Kim, J. Ha, and B. Zhang, "Overcoming catastrophic forgetting by incremental moment matching," *CoRR*, vol. abs/1703.08475, 2017. [Online]. Available: http://arxiv.org/abs/1703.08475

[57] J. Serrà, D. Surís, M. Miron, and A. Karatzoglou, "Overcoming catastrophic forgetting with hard attention to the task," *CoRR*, vol. abs/1801.01423, 2018. [Online]. Available: http://arxiv.org/abs/1801.01423

[58] F. Zenke, B. Poole, and S. Ganguli, "Continual learning through synaptic intelligence," in *International Conference on Machine Learning*. PMLR, 2017, pp. 3987–3995.

[59] R. Aljundi, F. Babiloni, M. Elhoseiny, M. Rohrbach, and T. Tuytelaars, "Memory aware synapses: Learning what (not) to forget," in *Proceedings of the European Conference on Computer Vision (ECCV)*, September 2018.

[60] Z. Li and D. Hoiem, "Learning without forgetting," *IEEE transactions on pattern analysis and machine intelligence*, vol. 40, no. 12, pp. 2935–2947, 2017.

[61] D. Lopez-Paz and M. Ranzato, "Gradient episodic memory for continual learning," *arXiv preprint arXiv:1706.08840*, 2017.

[62] S.-A. Rebuffi, A. Kolesnikov, G. Sperl, and C. H. Lampert, "icarl: Incremental classifier and representation learning," in *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, 2017, pp. 2001–2010.

[63] S. Hou, X. Pan, C. C. Loy, Z. Wang, and D. Lin, "Learning a unified classifier incrementally via rebalancing," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 831–839.

[64] Y. Liu, Y. Su, A.-A. Liu, B. Schiele, and Q. Sun, "Mnemonics training: Multi-class incremental learning without forgetting," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 12 245–12 254.

[65] X. Li, Y. Zhou, T. Wu, R. Socher, and C. Xiong, "Learn to grow: A continual structure learning framework for overcoming catastrophic forgetting," *arXiv preprint arXiv:1904.00310*, 2019.

[66] A. Y. Ng and M. I. Jordan, "On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes," in *Advances in neural information processing systems*, 2002, pp. 841–848.

[67] K. Kavukcuoglu, M. Ranzato, and Y. LeCun, "Fast inference in sparse coding algorithms with applications to object recognition," *arXiv preprint arXiv:1010.3467*, 2010.

[68] A. Ororbia and D. Kifer, "The neural coding framework for learning generative models," *Nature communications*, vol. 13, no. 1, pp. 1–14, 2022.

[69] M. Jaderberg, W. M. Czarnecki, S. Osindero, O. Vinyals, A. Graves, D. Silver, and K. Kavukcuoglu, "Decoupled neural interfaces using synthetic gradients," in *International conference on machine learning*. PMLR, 2017, pp. 1627–1635.

[70] W. M. Czarnecki, G. Świrszcz, M. Jaderberg, S. Osindero, O. Vinyals, and K. Kavukcuoglu, "Understanding synthetic gradients and decoupled neural interfaces," in *International Conference on Machine Learning*. PMLR, 2017, pp. 904–912.

[71] T. Salvatori, Y. Song, Y. Hong, L. Sha, S. Frieder, Z. Xu, R. Bogacz, and T. Lukasiewicz, "Associative memories via predictive coding," *Advances in Neural Information Processing Systems*, vol. 34, 2021.

[72] G. M. van de Ven, H. T. Siegelmann, and A. S. Tolias, "Brain-inspired replay for continual learning with artificial neural networks," *Nature communications*, vol. 11, no. 1, pp. 1–14, 2020.

[73] G. G. Turrigiano, "The self-tuning neuron: synaptic scaling of excitatory synapses," *Cell*, vol. 135, no. 3, pp. 422–435, 2008.

[74] K. Ibata, Q. Sun, and G. G. Turrigiano, "Rapid synaptic scaling induced by changes in postsynaptic firing," *Neuron*, vol. 57, no. 6, pp. 819–826, 2008.

[75] T. C. Moulin, D. Rayêe, M. J. Williams, and H. B. Schiöth, "The synaptic scaling literature: A systematic review of methodologies and quality of reporting," *Frontiers in cellular neuroscience*, vol. 14, p. 164, 2020.

[76] S. Beaulieu, L. Frati, T. Miconi, J. Lehman, K. O. Stanley, J. Clune, and N. Cheney, "Learning to continually learn," *arXiv preprint arXiv:2002.09571*, 2020.

[77] S. Madireddy, A. Yanguas-Gil, and P. Balaprakash, "Neuromodulated neural architectures with local error signals for memory-constrained online continual learning," *arXiv preprint arXiv:2007.08159*, 2020.

[78] N. Imam and T. A. Cleland, "Rapid online learning and robust recall in a neuromorphic olfactory circuit," *Nature Machine Intelligence*, vol. 2, no. 3, pp. 181–191, 2020.

[79] B. Tsuda, K. M. Tye, H. T. Siegelmann, and T. J. Sejnowski, "A modeling framework for adaptive lifelong learning with transfer and savings through gating in the prefrontal cortex," *Proceedings of the National Academy of Sciences*, vol. 117, no. 47, pp. 29 872–29 882, 2020.

[80] A. Prabhu, P. H. Torr, and P. K. Dokania, "Gdumb: A simple approach that questions our progress in continual learning," in *European conference on computer vision*. Springer, 2020, pp. 524–540.

[81] Z. Mai, R. Li, J. Jeong, D. Quispe, H. Kim, and S. Sanner, "Online continual learning in image classification: An empirical survey," *Neurocomputing*, vol. 469, pp. 28–51, 2022.

[82] D. Rolnick, A. Ahuja, J. Schwarz, T. Lillicrap, and G. Wayne, "Experience replay for continual learning," *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[83] G. Hu, W. Zhang, H. Ding, and W. Zhu, "Gradient episodic memory with a soft constraint for continual learning," *arXiv preprint arXiv:2011.07801*, 2020.

[84] J. Yoon, E. Yang, J. Lee, and S. J. Hwang, "Lifelong learning with dynamically expandable networks," *arXiv preprint arXiv:1708.01547*, 2017.

[85] M. J. Swain and D. H. Ballard, "Color indexing," *International journal of computer vision*, vol. 7, no. 1, pp. 11–32, 1991.

[86] N. Kamra, U. Gupta, and Y. Liu, "Deep generative dual memory network for continual learning," *arXiv preprint arXiv:1710.10368*, 2017.

[87] Y. Wu, Y. Chen, L. Wang, Y. Ye, Z. Liu, Y. Guo, and Y. Fu, "Large scale incremental learning," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 374–382.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] Limitations are discussed in the Appendix/Supplementary Material.
   (c) Did you discuss any potential negative societal impacts of your work? [Yes] Potential negative societal impacts of the work are discussed in the conclusion and in the Appendix/Supplementary Material.
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [No] The code and the data are proprietary. We will provide complete codebase to support this paper upon acceptance. We do provide detailed pesudocode of our model, describing overall flow which can be used to re-create our model.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes]
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] We report one standard deviation of uncertainty (10 trials).
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] Yes in appendix we provide details about the compute used to run all these experiments.
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes] Yes, we do cite all the existing assests in our paper as well as in our codebase.
   (b) Did you mention the license of the assets? [Yes] We do mention licensing in our codebase.
   (c) Did you include any new assets either in the supplemental material or as a URL? [Yes] In the appendix, we provide details for creating the special challenge benchmark of this paper.
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

Appendix

Computational Resource Setup: All experiments were performed with 128GB RAM on an intel Xeon server with 3.5GHZ processors, consisting of 4 1080Ti GPUs. Any of our models can easily fit into single 1080Ti GPUs – a multi GPU setup was only used to speed up the computation. All models are coded in Tensorflow 2.2 and we only use basic parallelism provided by the Tensorflow library to speed up computation.

Experimental Code: Code to support this work can be found at the following URL/repository: `https://github.com/ago109/lifelong_pc.git`.

Theoretical Foundations and Technical Details of Sequential Neural Coding

In this appendix section, we present definitions and mathematical derivations, answers to key questions, and further detail core intuitions related to sequential neural coding (i.e., the S-NCN system presented in the main paper).

Definition Table: In Table 3, we explain what each mathematical symbol/operation/abbreviation in the main paper represents.

Derivation of State & Weight Updates: As mentioned in the main paper, the S-NCN's generative circuit minimizes an objective function known as total discrepancy (ToD) when it is presented with input stimuli $(\mathbf{x}_i, \mathbf{y}_i)$. The ToD is formally:

$$\mathcal{L}(\Theta) = \sum_{\ell=0}^{L-1} \frac{1}{2}(||\mathbf{z}^\ell - \mathbf{z}^\ell_\mu||_2)^2 = \sum_{\ell=0}^{L-1} \frac{1}{2} \sum_j \left(\mathbf{z}^\ell[j] - \mathbf{z}^\ell_\mu[j]\right)^2 \tag{10}$$

where $\mathbf{z}^\ell[j]$ means that we extract the j-th element of vector $\mathbf{z}^\ell$ (and we have simplified the expression by squaring the square root operator of the L2 norm, giving us a sum of squared dimensions). Since all of the latent states of the generative circuit are continuous, the updates will follow the form of the exact gradient, i.e., differentiation (which would permit the use of gradient descent), to optimize the latent variables and the synaptic weight parameters. Given this, the partial derivative of Equation 10 with respect to any layer of neural activities (or latent state) $\mathbf{z}^\ell$ would be:

$$\frac{\partial \mathcal{L}(\Theta)}{\partial \mathbf{z}^\ell} = \left(\frac{\partial \mathbf{z}^{\ell-1}_\mu}{\partial \mathbf{z}^\ell} \cdot \left((\mathbf{z}^{\ell-1} - \mathbf{z}^{\ell-1}_\mu) \right) \right) - (\mathbf{z}^\ell - \mathbf{z}^\ell_\mu) \tag{11}$$

$$= \left[(\mathbf{W}^\ell)^T \cdot (\mathbf{z}^{\ell-1} - \mathbf{z}^{\ell-1}_\mu) \right] \odot \frac{\partial \phi^\ell(\mathbf{z}^\ell)}{\partial \mathbf{z}^\ell} - (\mathbf{z}^\ell - \mathbf{z}^\ell_\mu) \tag{12}$$

$$= (\mathbf{W}^\ell)^T \cdot (\mathbf{e}^{\ell-1}) \odot \frac{\partial \phi^\ell(\mathbf{z}^\ell)}{\partial \mathbf{z}^\ell} - \mathbf{e}^\ell \tag{13}$$

where we notice that the error neurons are derived directly from the ToD objective as well, i.e., $\mathbf{e}^\ell = \frac{\partial \partial \mathcal{L}(\Theta)}{\partial \mathbf{z}^\ell_\mu} = \mathbf{z}^\ell - \mathbf{z}^\ell_\mu$ (allowing us to write Equation 12 in terms of error neurons as in Equation 13). Alternatively, by replacing the term $\frac{\partial \mathbf{z}^{\ell-1}}{\partial \mathbf{z}^\ell}$ with a learnable error matrix $\mathbf{E}^\ell$ instead, Equation 12 can be simplified to the following:

$$\frac{\partial \mathcal{L}(\Theta)}{\partial \mathbf{z}^\ell} \approx \mathbf{d}^\ell = \mathbf{E}^\ell \cdot \mathbf{e}^{\ell-1} - \mathbf{e}^\ell \tag{14}$$

which is a stable derivative-free perturbation $\mathbf{d}^\ell$ (so long as the activation function $\phi^\ell()$ is monotonically increasing) to the neural activities (and as noted in [68], the dampening effect that the activation derivative $\frac{\partial \phi^\ell(\mathbf{z}^\ell)}{\partial \mathbf{z}^\ell}$ would have can be approximated with biologically-plausible dampening functions if needed). The final update to the latent neural activities is then performed using a gradient-ascent like operation, i.e., $\mathbf{z}^\ell(k) = f^\ell(\mathbf{z}^\ell(k-1) + \beta \mathbf{d}^\ell)$ (this was presented in the main paper).

Table 3: Table of key symbol/operator/abbreviation definitions.

Item	Explanation
S-NCN	Sequential neural coding network (model)
FNBG	Functional neural basal ganglia (model)
$\mathbf{v} \in \mathcal{R}^{D \times 1}$	A column vector $\mathbf{v}$ of shape $D \times 1$
$\mathbf{M} \in \mathcal{R}^{B \times D}$	A matrix $\mathbf{M}$ of shape $B \times D$
$\cdot$	Matrix/vector multiplication
$\odot$	Hadamard product (element-wise multiplication)
$(\mathbf{v})^T$	Transpose of $\mathbf{v}$
$\|\|\mathbf{v}\|\|_2$	Euclidean norm of $\mathbf{v}$
$\mathbf{x}_j$	The jth data point (image) sampled from task $\mathcal{T}_i$
$\mathbf{y}_j$	The jth data label (one-hot encoded) sampled from task $\mathcal{T}_i$
P_ℓ	The ℓ-th predictor/layer of the S-NCN generative circuit.
J_x	Dimensionality of input $\mathbf{x}_j$
J_ℓ	Number of neurons in layer P_ℓ of the S-NCN generative circuit.

Deriving the updates to the synaptic generative parameters is also done in a similar fashion as above, i.e., by taking the gradient of ToD with respect to $\mathbf{W}^\ell$.

$$\frac{\partial \mathcal{L}(\Theta)}{\partial \mathbf{W}^\ell} \propto \Delta \mathbf{W}^\ell = \frac{\partial \mathcal{L}(\Theta)}{\partial \mathbf{z}^\ell_\mu} \cdot \left(\phi^{\ell+1}(\mathbf{z}^{\ell+1})\right)^T, \text{where, } \mathbf{z}^\ell_\mu = \mathbf{W}^\ell \cdot \phi^{\ell+1}(\mathbf{z}^{\ell+1}) \tag{15}$$

$$= (\mathbf{z}^\ell - \mathbf{z}^\ell_\mu) \cdot (\phi^{\ell+1}(\mathbf{z}^{\ell+1}))^T = \mathbf{e}^\ell \cdot (\phi^{\ell+1}(\mathbf{z}^{\ell+1}))^T. \tag{16}$$

If we are using $\mathbf{E}^\ell$ feedback/error matrices (as we do in this paper), we can leverage a simple Hebbian update $\Delta \mathbf{E}^\ell = \alpha\Big(\phi^{\ell+1}(\mathbf{z}^{\ell+1}) \cdot (\mathbf{e}^\ell)^T\Big)$ [68] (which, if applied to $\mathbf{E}^\ell$ every time that Equation 16 is applied to $\mathbf{W}^\ell$, allows $\mathbf{E}^\ell$ to converge to the approximate transpose of $\mathbf{W}^\ell$). Much as was done for the states, synaptic weight matrices are updated via gradient ascent: $\mathbf{W}^\ell = \mathbf{W}^\ell + \lambda \Delta \mathbf{W}^\ell$ and $\mathbf{E}^\ell = \mathbf{E}^\ell + \lambda \Delta \mathbf{E}^\ell$ (λ is the learning rate/step size).

How would a model with symmetric connections behave? A model without separate feedback connections (in contrast to the S-NCN we experiment that uses asymmetric forward/feedback weights) would behave quite similarly yet favorably offer a reduction in memory cost (one does not need to store separate feedback/error matrices in memory). In other words, one could certainly swap out $\mathbf{E}^\ell$ with $(\mathbf{W}^\ell)^T$ if this memory cost reduction was desired/necessary. However, by utilizing separate learnable feedback synapses, the S-NCN in the form presented in this study resolves the weight transport problem, a well-known biological criticism of backprop where error/teaching information is carried backwards along the same synapses that were used to forward propagate information.Interestingly enough, in preliminary experimentation, we found that using separate learnable feedback synapses improved the generative modeling/reconstruction ability of the generative cortex (particularly in the online case). Although we will investigate this effect in future work, we note this change in generative performance did not really seem to impact the classification accuracy (arguably because discrimination is least complex when compared with generation).

Initializing Latent States: In the S-NCN's generative circuit, there are several layers of neural activities that are not clamped to data, e.g., $\mathbf{z}^1, \mathbf{z}^2, ..., \mathbf{z}^L$. These activity vectors, as mentioned in the main paper, are initialized to zero vectors (i.e., $\mathbf{z}^\ell = \mathbf{0}$) before they are updated/modified by the message passing that occurs over K steps of processing input stimuli $(\mathbf{x}, \mathbf{y})$. While these initially zero vectors will eventually become non-zero vectors, particularly after a minimum of $K = L$ steps (for example: after $k = 1$, $\mathbf{z}^1$ will be non-zero given that the error neurons in layer 0 will be non-zero and thus the layer 1 state perturbation vector $\mathbf{d}^1$ will contain non-zero entries; after $k = 2$, $\mathbf{z}^2$ will contain non-zero values, and so on and so forth), it is entirely possible to randomly initialize these states with non-zero vectors (though it is recommended to keep the magnitude of the randomly chosen initial numbers relatively small). We leave investigation of alternative initialization schemes for future work.

Relationship to Surrogate Gradients: A particular line work that shares interesting relationships with the generative circuit of the S-NCN is that of surrogate gradients, such as decoupled neural interfaces (DNIs) [69, 70]. In essence, this class of methods aims to resolve one of backprop's central

issues – the update-locking problem (where updates to one layer's synaptic parameters must for other layer's updates to be computed as error/teaching information is backwards propagated down a serial feedback pathway). The key module driving this class of methods is the introduction of a gradient predictor, which can be adapted/taught to approximate actual gradients as produced by backprop, ultimately, after training the predictor's well enough removing the need for backprop later in training and permitting parallel, asynchronous updates to be made to deep, even recurrent, network architectures. In contrast to surrogate gradient-based approaches, the S-NCN works to compute synaptic updates without resorting to predicting gradients given that its generative circuit is naturally layer-wise parallelizable. In effect, each layer-wise prediction is made independently of the others (unlike the typical forward passes in modern-day deep networks) and the synaptic updates for each layer (both forward and error/feedback synapses) can be made without others having been computed/completed. This opens the door to potential parallel asynchronous implementations of the S-NCN that could drastically speed up its computation further. In contrast to DNIs, the S-NCN's generative cortex does not require training gradient predictors (DNIs typically require access to true gradients provided by backprop in order to train them properly) and, without incurring synthetic approximation errors (as in a fully-unlocked network using DNI, where now even the layer activities require additional modules to be trained to predict actual layer-wise activities) furthermore, resolves both update and forward locking problems. Crucially, the S-NCN's updates are biologically plausible – it only requires simple (multi-term) Hebbian updates for the generative cortex and competitive Hebbian updates for the basal ganglia.

General S-NCN System Process Intuition: From a high-level intuitive point-of-view, the S-NCN system described in the main paper is composed of two complementary neural circuits: 1) an interactive generative (cortical) circuit that learns to predict its input stimuli (pixel images and their respective labels), and 2) the functional neural basal ganglia (FNBG) which is a specialized circuit that learns to group pixel images into unique "task" contexts. When presented with a sample or mini-batch at any point within the task stream being sampled, the S-NCN does the following:

1. The FNBG task selection model determines if the currently sampled data is coming from the same task that the S-NCN has currently been processing or if it comes from either new/different task. (Note, not mentioned in the paper, the FNBG can also determine that the current data belongs to a task that it has previously seen by letting its set of neurons compete and determining if the winner has a very high dot product score – usually checked against a threshold). Before letting the S-NCN generative circuit process/adapt to current data, if the FNBG determines that the data is coming from a novel task, it will create a new task context memory $\mathbf{g}_t^\ell$ for each $\mathbf{M}^\ell$ to subsequently drive the S-NCN as it processes current data.
2. Given the task context provided by the FNBG, the S-NCN generative circuit will then process the current data over a fixed stimulus window (or for K discrete time steps), adjusting its synapses so that it may better generate the input sensory sample better in the future as well as predict its label. Note that the update to the S-NCN's synapses will also trigger an update to the task context memory vectors as well as an update to the FNBG's synapses (which are themselves adjusted through competitive Hebbian learning).

Task Shift Detection Intuition: To provide further intuition as to how the FNBG task selector operates when detecting the occurrence of a novel task, i.e., "task shift detection", we explain what the key equations presented in the main paper are doing:

1. The FNBG lets the S-NCN generative circuit continue as it normally would and have it make a prediction of both $\mathbf{x}_j$ and $\mathbf{y}_j$. Then, it extracts the label error neuron vector $\mathbf{e}_y^0$ from the generative model and computes its squared Euclidean norm $||\mathbf{e}_y^0||_2^2$.
2. The FNBG maintains two particular scalar parameters $\mu_\mathcal{L}$ and $\sigma_\mathcal{L}^2$, which are the mean and variance of the squared Euclidean norm of the S-NCN's label error neuron activities. The key Equations 7-9 in the main paper depict how the FNBG updates these mean $\mu_\mathcal{L}(t)$ and variance $\sigma_\mathcal{L}^2(t)$ parameters (based on an exponentially weighted moving mean and variance).
3. Once the FNBG updates these two statistical parameters, it then performs a check of its current new mean $\mu_\mathcal{L}(t)$ against the previous value $(t-1)$ of its mean and variance parameters, i.e., $\mu_\mathcal{L}(t-1)$ and $\sigma_\mathcal{L}^2(t-1)$. This check is specifically (as in the main paper) the following: $\mu_\mathcal{L}(t) > \mu_\mathcal{L}(t-1) + 2\sqrt{\sigma_\mathcal{L}^2(t-1)}$ which says that, if the current mean of the label error neuron activity (at time t) is greater than the sum of the previous mean (at time $t-1$) plus two times the previous standard deviation, the S-NCN system has encountered a

shift in task, which means that, if this inequality evaluates to true, the currently encountered data sample comes from either a novel task or a previously encountered task (but not from the current task). If this task shift inequality evaluates to true, the FNBG will not allow the generative circuit to update its synapses and instead force it to recompute its prediction of the current data using a newly generated task context. Otherwise, if the inequality evaluates to false, then the FNBG will let the generative circuit continue and update its synapses.

General Design Motivation/Benefits: The key motivations behind the S-NCN's design are: 1) to develop a neurobiologically-inspired online approach to learn a generative model of $p(\mathbf{x}, \mathbf{y})$ (one motivated by perceptual cortical circuits), 2) to develop an online information routing model (one motivated/inspired by the basal ganglia) that suppresses/drives the neural activities in the generative model of item (1) depending on the task that it decides that the system is operating on – this crucially removes the need for user-provided task descriptors. In addition, by design, the S-NCN: 1) exhibits no need for activation function derivatives, 2) exhibits no update-locking (i.e., it is layer-wise parallelizable), 3) does not require/need a global feedback pathway to drive/facilitate credit assignment (i.e., side-stepping exploding/vanishing gradient problems), and 4) it resolves the weight transport problem by using asymmetric generative and error-correction synapses (those these could be tied/shared to reduce the S-NCN's memory footprint if need be). In contrast to many current lifelong learning approaches, the S-NCN differs in that it is a complementary system that focuses on the relationship between the basal ganglia (as an information router) and (generative/predictive) cortical regions. This is what allows the S-NCN system to offer the advantage of internal, automatic task selection and task boundary detection, which few modern methods provide.

Although the S-NCN shares predictive processing's (PP's) iterative inference/learning (note that [71] focused on PP's auto-association abilities and further note that [68] focused on PP's generative/sampling abilities); its ability to combat forgetting comes from the interaction between the task selector and the generative circuit (the former drives lateral suppression/excitation in the latter). Note that the S-NCN's generative model could be useful for inducing a form of (internal) layer-wise generative replay (each layer could refresh itself in a sleep phase). This we observe is additional, untapped potential for using the S-NCN's learned directed generative model in a spirit similar to the backprop-based model of [72]. We argue that this self-induced form of replay could prove useful for longer task sequences with far more tasks.

Empirically, we note that asymmetric forward and backwards (error) weights were found to work best for NCN systems without activation derivatives in their state updates, yielding fewer inference steps. However, since backwards weights converge to the approximate transpose of the forward ones, using symmetric weights would reduce memory usage (see earlier discussion/section "How would a model with symmetric connections behave?"). We further remark that, although we do not explore them in this work, fixed, random backward synapses (as in feedback alignment) would work as well, which we found, in preliminary experimentation, reduced/harmed the S-NCN's ACC by only a few points.

Relationship to Temporal Neural Coding: Prior related work on temporal neural coding [49, 35] has investigated the design and development of predictive coding frameworks for handling the modeling of time-varying data points (such as frames in a video). However, in contrast to the generative circuit in this work, these previous models were either not layer-wise parallelizable [49] or were created under a specialized recurrent formulation of PC for processing data over multiple epochs [35] (such as bouncing ball or digit videos). These models would not be naturally resistant to forgetting like the S-NCN is, specifically given that the S-NCN is a complementary system where one circuit aids the other in learning task-sensitive/dependent sub-networks. Furthermore, this earlier related work [49, 35] focused on unsupervised sequence modeling and did not investigate discriminative forms of learning (e.g., classification) where forgetting occurs, according to our experience, more quickly and its effects are stronger/more easily observed. Note that in [35], although some improved memory retention across the three sequence modeling tasks was observed, forgetting over tasks was still apparent, and any improvement in memory retention could be be viewed more as a pleasant side-effect rather than the result of a mechanism specialized for safe-guarding against forgetting.

On Weight Update Modulation

As presented in the main paper, the synaptic weight updates for the generative circuit of the S-NCN applies modulation matrices to improve learning stability over time (specifically invoking a form of synaptic scaling – note that these modulation matrices are not meant to mitigate the occurrence of

forgetting). The modulation factors are (locally) computed as a function of the synapses as follows:

$$\widehat{\mathbf{m}}^{\ell}_W = \Sigma_{j=1}^{J_{\ell+1}} \mathbf{W}^{\ell}[:, j], \text{ and, } \mathbf{m}^{\ell}_W = \min\left(\frac{\gamma_s \widehat{\mathbf{m}}^{\ell}_W}{\max(\widehat{\mathbf{m}}^{\ell}_W)}, 1\right) \tag{17}$$

$$\mathbf{S}^{\ell}_W = (\mathbf{W}^{\ell} * 0 + 1) \odot \mathbf{m}^{\ell}_W \tag{18}$$

where $\mathbf{W}[:, j]$ denotes the extraction of the jth column of W, $\max(\widehat{\mathbf{m}}^{\ell}_W)$ returns the maximum scalar value of $\widehat{\mathbf{m}}^{\ell}_W$, and $\gamma_s = 2$. We note that the first two formulae collapse the forward matrix to a column vector of normalized multiplicative weighting factors and the third formula converts the column vector to a tiled matrix of the same shape as $\mathbf{W}^{\ell}$. The error weight modulation factor is computed in fashion similar to that of the forward weights:

$$\widehat{\mathbf{m}}^{\ell}_E = \Sigma_{j=1}^{J_{\ell}} \mathbf{E}^{\ell}[:, j], \text{and, } \mathbf{m}^{\ell}_E = \min\left(\frac{2\widehat{\mathbf{m}}^{\ell}_E}{\max(\widehat{\mathbf{m}}^{\ell}_E)}, 1\right) \tag{19}$$

$$\mathbf{S}^{\ell}_E = (\mathbf{E}^{\ell} * 0 + 1) \odot \mathbf{m}^{\ell}_E \tag{20}$$

where we observe that modulation factors are computed across the pre-synaptic dimension/side of either matrix $\mathbf{W}^{\ell}$ or $\mathbf{E}^{\ell}$. The multiplicative modulation terms come from the insight in neuroscience that synaptic scaling, driven by competition across synapses, serves as a global (negative) feedback mechanism for regulating the magnitude of synaptic adjustments [73, 74, 75]. From a practical standpoint, we found that using the above modulation/scaling factors meant we did not have to craft a synaptic normalization scheme (such as in sparse coding schemes, where the columns/rows of a synaptic matrix must be normalized such that they of unit length each time the matrix is updated).

We remark that the modulation factors we introduce could instead be adapted such that they are useful for mitigating forgetting instead, as has been done in other continual learning approaches [76, 77, 78, 79]. This could help to reduce the cost for growing out new task contexts each time a new task is encountered. For example, one could adapt the modulation factor matrices to instead be conditioned on the output of the S-NCN's functional neural basal ganglia (or another type of task selector circuit, such as one that mimics the cognitive control capabilities of the prefrontal cortex).

On Partial Pattern Completion

In the event that incomplete input $\mathbf{x}_i$ is provided to the S-NCN, i.e., portions of $\mathbf{x}_i$ are masked out by the variable $\mathbf{m} \in \{0, 1\}^{J_0 \times 1}$, as mentioned in the main paper, we may infer the remaining portions of $\mathbf{x}_i$ by using the relevant output error neurons of the S-NCN and treating the bottom sensory/input layer $\mathbf{z}^0_x$ as a partial latent state. Specifically, we update the missing portions, i.e., $1 - \mathbf{m}$, of $\mathbf{z}^0$ as:

$$\mathbf{z}^0_x = \mathbf{x} \odot \mathbf{m} + \left(\mathbf{z}^0_x - \beta \mathbf{e}^0_x\right) \odot (1 - \mathbf{m}) \tag{21}$$

where $\mathbf{e}^0_x = \mathbf{z}^0_x - \mathbf{z}^0_{\mu,x}$ (error neuron signals related to $\mathbf{x}_i$).

Parameter Optimization Setup and Baseline Details

S-NCN Optimization: For the S-NCN, we used SGD with a learning rate of $\lambda = 0.01$ (this rate was only minorly tuned on the validation set of the first task in preliminary investigation) and mini-batches each containing 10 samples. Based on preliminary experiments, the S-NCN, in general, was found to be robust w.r.t. such hyper-parameters. However, the final meta-parameter values used, i.e., $\beta = 0.05$, $K = 10$, $\eta_g = 0.9$, $\eta_e = 0.01$, $\alpha = 0.98$, were obtained by conducting a grid search (using the validation sets to find best generalization). This meant that we searched over the ranges: $\beta = [0.01, 0.1]$, $K = [5, 30]$, $\eta_g = [0.5, 1.0]$, $\eta_e = [0.005, 0.25]$, $\alpha = [0.5, 1.0]$.

Baseline Descriptions: The baselines include an MLP trained exclusively with backprop (Backprop), an MLP trained by backprop but regularized by drop-out (Backprop+DO), Elastic Weight Consolidation (EWC) [19], EWC further regularized by drop-out (EWC-DO), the mean incremental moment matching method (IMM) or Mean-IMM [56], the Mode-IMM method [56], the Mean-IMM method combined with either DropTransfer (DT+Mean-IMM) or L2-transfer (L2-Mean-IMM) or both (L2+DT+Mean-IMM) [56], the Md-IMM method combined with either DropTransfer (DT+Md-IMM) or L2-Transfer (L2+Md-IMM) or both (L2-DT-Mode-IMM) [56], and the state-of-the-art competitive model, hard attention to task (HAT) [57]. Furthermore, as mentioned in the main paper, we examine other methods including those based on replay/rehearsal: naïve rehearsal with memory

Model General Hyperparameters			
Model	**MNIST** *Configuration*	**FMNIST** *Configuration*	**NotMNIST** *Configuration*
EWC	lr = 1e-3, SGD NH = 1024, NL = 2	lr = 1e-4, SGD+M NH = 512, NL = 3	lr = 1e-3, SGD NH = 1024, NL = 2
VCL	lr = 2e-4, Adam NH = 512, NL = 3	lr = 2e-4, SGD+M NH = 512, NL = 3	lr = 2e-4, Adam NH = 512, NL = 3
IMM	lr = 2e-3, SGD+M NH = 512, NL = 3	lr = 1e-5, SGD NH = 512, NL = 3	lr = 1e-4, SGD+M NH = 512, NL = 3
HAT	lr = 1e-4, SGD NH = 512, NL = 3	lr = 2e-4, SGD NH = 512, NL = 3	lr = 2e-4, SGD+M NH = 512, NL = 3
A-GEM	lr = 1e-4, SGD+M NH = 512, NL = 3	lr = 2e-5, SGD NH = 512, NL = 3	lr = 2e-4, SGD+M NH = 512, NL = 3
ER	lr = 2e-4, Adam NH = 512, NL = 3	lr = 1e-4, AdamW NH = 512, NL = 3	lr = 1e-4, AdamW NH = 512, NL = 3
EWC	lr = 1e-3, SGD+M NH = 1024, NL = 2	lr = 1e-3, SGD+M NH = 1024, NL = 2	lr = 1e-3, SGD NH = 512, NL = 3
NR+M	lr = 1e-4, SGD+M NH = 512, NL = 3	lr = 1e-3, SGD+M NH = 512, NL = 3	lr = 1e-4, SGD NH = 512, NL = 3
SI	lr = 1e-3, SGD NH = 512, NL = 3	lr = 1e-3, SGD+M NH = 512, NL = 3	lr = 1e-3, SGD NH = 512, NL = 3
MAS	lr = 1e-4, Adam NH = 512, NL = 3	lr = 1e-4, Adam NH = 512, NL = 3	lr = 2e-4, Adam NH = 512, NL = 3
Lwf	lr = 1e-3, SGD NH = 1024, NL = 2	lr = 1e-3, SGD NH = 1024, NL = 2	lr = 1e-3, SGD NH = 1024, NL = 2
ICarl	lr = 1e-3, SGD+M NH = 1024, NL = 2	lr = 1e-3, SGD+M NH = 1000, NL = 2	lr = 1e-4, SGD+M NH = 1000, NL = 2
Lucir	lr = 1e-4, Adam NH = 512, NL = 3	lr = 2e-4, AdamW NH = 512, NL = 3	lr = 2e-5, AdamW NH = 512, NL = 3
GDumb	lr = 2e-4, AdamW NH = 512, NL = 3	lr = 2e-5, Adam NH = 512, NL = 3	lr = 2e-4, AdamW NH =512 , NL = 3
Mnem	lr = 1e-4, Adam NH = 512, NL = 3	lr = 2e-5, Adam NH = 500, NL = 3	lr = 2e-4, Adam NH = 512, NL = 3
S-NCN	lr = 0.0105, SGD NH = 500, NL = 3	lr = 0.01, SGD NH = 500, NL = 3	lr = 0.011, SGD NH = 500, NL = 3

Table 4: General hyper-parameter values selected from tuning. Models above double horizontal line are multi-head models and models below it are single-head models. "NH" stands for "number of hidden neurons", "NL" stands for "number of hidden layers", "lr" stands for "learning rate", and optimizer choice is either "SGD" for "stochastic gradient descent" ("SGA" means "stochastic gradient ascent"), "Adam", "AdamW", and 'SGD+M" for "SGD with momentum").

(NR+M), EWC, synaptic intelligence (SI), MAS [59], Lwf [60], GEM [61], ICarl [62], Lucir [63], and Mnemonics [64]. With respect to very modern baselines, we also include, in the main paper, results for the greedy sampler and dumb learner (GDumb) [80, 81], experience replay (ER) [82, 81], and average gradient episodic memory (A-GEM) [83, 81].

Baseline Meta-parameter Tuning: For all baselines we take/use the hyper-parameters from each model/algorithm's source work as a starting point and tuned each, using grid search, the batch size, learning rate, number of hidden units in each layer of the target MLP classifier, and optimizer choice. We tuned across the following ranges: learning rate range was $[3e-5, 0.1]$, the optimizer choice was tuned across the discrete set ["SGD", "momentum", "Adam", "AdamW"], the hidden layer size range was $[128, 512]$, and number of layers range was $[2-5]$. For each baseline, we tuned hyper-parameters based on their accuracy on each task's development set (as mentioned in the next sub-section). For IMM, we used the same settings proposed in the original paper as a starting point [56]. However, we found that HAT [57] was quite sensitive to the choice of its two key hyper-parameters: 1) the stability parameter $\mathrm{s}_{\max}$, and 2) the "compressibility" parameter c. After extensive tuning, we used $\mathrm{s}_{\max} = 450$ and $\mathrm{c} = 0.78$. For other baseline-specific hyperparameters, e.g., A-GEM has a gamma and soft-constraint meta-parameter, we used the best-practice values reported in the literature (as we found that these values worked well in general, even after some preliminary experimentation).

Limitations (Expanded Discussion): Our model jointly predicts the target label and learns to generate the sensory input, further driven/modulated by a simple complementary neural system that mitigates neural cross-talk. The dual nature of our model/system helps to uncover distributed representations that facilitate robust learning and adaptation over sequences of tasks/datasets. Even though this design scheme provides flexibility and seems to offer many advantages compared to other backprop-based models, it does come with several limitations. Mainly, finding the true posterior distribution over latent neural activities is harder than just learning a forward mapping between inputs and output targets and, notably, it can be expensive to find a good set of neural activity values as the problem complexity increases (notably the K hyper-parameter, which controls the amount of steps taken per data point/mini-batch by the S-NCN to iteratively infer a potential maximum a posterior estimate of its state variables). Currently, the S-NCN conducts inference and learning through a sort of expectation-maximization process and, fortunately, in the problems we studied, the value of K was fairly low (only 10 to 20 steps at most were needed to find useful state values per sample/mini-batch). However, for more complex data types, such as natural images with multiple objects and even background scenery, the value of K will quite likely need to be much higher, increasing the computation time needed to conduct online inference. This drawback could be mitigated by integrating amortized inference, e.g., predictive sparse decomposition [67], and by designing custom software/hardware implementations to exploit the S-NCN's layer-wise parallelism.

Furthermore, the fact that the S-NCN (in its current form/implementation in this study) must solve a dual optimization problem that entails jointly learning to predict the target label and generate the sensory input (image) might compromise the model's overall accuracy when tested on large-scale images. It is often an easier problem to directly learn a conditional mapping between the input and label as opposed to learning a full generative model as the S-NCN does [66]. Future work should explore adapting the S-NCN to only learn a conditional mapping as opposed to a full joint distribution over inputs and labels as well as potential mechanisms for pre-training the generative side of the system (which would allow freezing of the synaptic weights for generation and only require updating discriminative parameters – this could potentially reduce the value of K even for more complex sensory inputs). Another drawback, yet also simultaneously a strength, is the fact that the S-NCN is attempting to optimize (online) total discrepancy as opposed to a single, global surface loss. While total discrepancy is one important key to breaking free of backprop and its limitations it also creates a more challenging optimization problem in general, i.e., the neural system must now not only match the values created by data but also ensure its internal activities and its local predictions of each of them are aligned. While the overall complementary system mitigates catastrophic interference (or the neural cross-talk that would trigger the loss/deletion of previously acquired knowledge), this primarily affects measurements of backward transfer (BWT) but could potentially damage the model's per-task performance, i.e., the main diagonal of its task matrix. Since we do not impose any strong distributional assumptions over the latent activities (such as a clean Gaussian prior as is often done in variational autoencoders), if the S-NCN's estimated value of the latent activities are far from the true posterior, then the S-NCN might produce sub-optimal performance, especially for complex problems. Even though all continual learning systems suffer from this issue (especially most modern-day continual learning ANNs), our model's dual optimization nature could experience this problem more frequently. We believe that integrating memory-aware retrieval from synapses, a mechanism guided by (a brain-like form of) replay, could help direct the system to be closer to the true posterior by avoiding bad local minima when learning continuously.

Additionally, with respect to our proposed task selection mechanism (the functional neural basal ganglia circuit), one notable drawback is that a small refractory period is imposed in order to ensure that enough data is accumulated from the stream to update the competitive task selector's weights. This would be an issue for task streams that constantly introduce tasks with fewer samples than the pre-set refractory period. A subject of our future work is to improve the power /adaptability of the task selection model in the face of more volatile task sequence streams. Another limitation is that the S-NCN is, in effect, a dynamically-expanding architecture: there is an overhead for the task-context memory – one new task context vector would need to be generated/grown for each new (disjoint) task is encountered. While this required parameter growth/generation is not as high as other dynamically-expanding architecture approaches (such as progressive networks [16] or dynamically-expanding networks [84], where many new parameters must be created per task). While the inclusion of relatively few, additional context vector parameters is more desirable, requiring the growing out of parameters at a rate far less than methods such as [16, 84]. To mitigate the cost that even the S-NCN imposes, we remark that the S-NCN's generative cortex could be adapted to induce its

own form of layer-wise replay as one alternative, similar to [72], or that another circuit could be designed to potentially learn how to compress these task contexts by reducing redundancy exploiting overlap/redundancy between contexts (serving as a form of efficient long-term memory).

Finally, a more obvious drawback is that the S-NCN's error synapses also increase the memory footprint of the overall model. It would be advisable, when using/applying a model like the S-NCN on other continual learning problems, to select the number of hidden layers and number of neurons in each layer based on model capacity, i.e., compute the total number of (generative and error) synapses that would result from making the neural structure more complex or deeper. A more long-term, promising means of mitigating the increased memory footprint would be to design error units further inspired by actual neurons – instead of assuming a one-to-one mapping (one error neuron per state neuron), design small pools of neurons that are responsible for computing the mismatch activities for large groups of state neurons. This is a key solution to investigate in future work.

Creating Task Orderings #1 and #2

To create our sequential learning benchmarks, we utilize the MNIST, Fashion MNIST, and Google Draw datasets to create various sets of "subtasks", or rather, classification problems that involve different classes of the original set of each full dataset. In this paper, we create a 6 task sequence, $\{T_1, T_2, T_3, T_4, T_5, T_6\}$, from these datasets, where two tasks are generated from each specific dataset. To create the task splits, we create data subsets based on minimizing the amount of knowledge transfer across data splits, specifically by examining the amount of stroke overlap in the images across classes, yielding a challenging problem. For equal number of classes, the splits we created were: MNIST set #1, *M1* = $\{0, 8, 3, 5, 2\}$, MNIST set #2, *M2* = $\{1, 4, 6, 7, 9\}$, Fashion MNIST set #1, *FM1* = $\{$top, trouser, pullover, dress, coat$\}$, Fashion MNIST set #2, *FM2* = $\{$sandal, shirt, sneaker, bag, ankle boot$\}$, Google Draw set #1, *GD1* = $\{$objects that were car or bike variants $\}$, and Google Draw set #2, *GD2* = $\{$objects belong to variants of airplanes or submarines $\}$. For a task sequence, we create two scenarios: 1) where number of classes are equal for all tasks (i.e., 5 classes in our setup), and 2) where number of classes are unequal (number of classes per task was chosen randomly, omitting the number 5 as an option). In our experiments, we investigate two task orderings (Ordering #1 and Ordering #2). We compute the color index similarity [85] between every pair of tasks (as a proxy for task similarity) and randomly chose Orderings # 1 and # 2 so that the color similarity between adjacent tasks was higher for Ordering #1 ("High Color Sim." for high color similarity) than for Ordering #2 ("Low Color Sim." for low color similarity), hence task transfer should be easier for Ordering #1 than #2. These particular orderings could be considered to be "harder" and "easier" task orderings, respectively, since it is possible that a the difference in color-index would make it easier to differentiate the tasks (the diversity of inputs from the first few tasks might even improve the performance, as it would be in the case of Ordering # 2). We can see this reflected by the fact that all models (the S-NCN and the baselines) perform a bit better in general on Ordering # 2 (low color similarity or "easier" ordering) than on Ordering # 1 (high color similarity or "harder" ordering).

Expanded Results for Task Orderings # 1 and # 2

Metrics for Quantifying Memory Retention: The formulas for ACC and BWT are:

$$\text{ACC} = \frac{1}{T}\sum_{i=1}^{T} R_{T,i}, \quad \text{BWT} = \frac{1}{T-1}\sum_{i=1}^{T-1} R_{T,i} - R_{i,i}. \tag{22}$$

In addition, **we propose two additional, complementary metrics**, with the motivation that these metrics examine aspects of forgetting and capacity not clearly captured by ACC or BWT. Our two measures, True BWT (TBWT) and Cumulative BWT for task T_t (CBWT(t)), are defined as follows:

$$\text{CBWT(t)} = \frac{1}{T-t}\sum_{i=t+1}^{T} R_{i,t} - R_{t,t} \tag{23}$$

$$\text{TBWT} = \frac{1}{T-1}\sum_{i=1}^{T-1} R_{T,i} - G_{i,i} \tag{24}$$

where $G_{i,i}$ is the performance of an independent classifier trained on task i (in our experiments, this was a full capacity MLP trained via backprop). TBWT relates the degradation in prior task performance by replacing the diagonal of task matrix R with a "gold standard", which is the performance of a model that, in isolation, is able to allocate its full capacity to a particular task. CBWT(t) is a task-specific metric, where we instead examine a particular column of R, and measure the total amount of forgetting throughout the sequential learning process, instead of simply examining the final performance at the end (bottom row of R) as BWT can only do. CBWT(t) would punish models that suffer large dips in performance in the middle of learning (but not necessarily at the end), and would be better suited for characterizing forgetting in stream settings than BWT.

Discussion: Results are reported in Tables 5 and 6 (an expanded version of the original one in the main paper). Each simulation was run 10 times, each trial using a unique seed for pseudo-random number generation, we report both mean and standard deviation. As we observe in our experimental results, we see that all of our S-NCN models exhibit improved memory retention over simple baselines, such as backprop, and more notably, EWC. However, we see that incorporating task-driven lateral inhibition in facilitating gradual forgetting as opposed to catastrophic forgetting, as evidenced by the very competitive performance of both Lat1-S-NCN and Lat2-S-NCNs, with Lat1-S-NCN outperforming all baselines consistently, in terms of both ACC and BWT. This result is robust across both task sequences and equal/ unequal class settings. It is further important to note that the meta-parameter settings used for the various S-NCNs were only tweaked minorly with the same values across all of the settings/scenarios. The observation that lateral inhibition improves the neural computation of our interactive network further corroborates the result of [48], though it focused on models trained via contrastive Hebbian learning.

Upon examination of Table 5, in terms of TBWT and CBWT(1)[5], the proposed lateral S-NCNs still outperform the baselines. The key difference is that we see that the lateral S-NCNs actually do retain prior information throughout learning and do not simply just recover it at the end.

Expanded Benchmark Results

To start, we describe the three forms of the lateral competition we designed for the S-NCN. They were specifically as follows:

1. $f^\ell(\mathbf{z}^\ell, \mathbf{g}_t^\ell) = \mathbf{I} \cdot \mathbf{z}^\ell$, which means that the lateral inhibitory matrix is fixed to a diagonal matrix $\mathbf{I}$ and forces the model to ignore the task embedding (in the Appendix, we denote this as "NoLat-S-NCN"),
2. $f^\ell(\mathbf{z}^\ell, \mathbf{g}_t^\ell) = \left(\mathbf{I} \odot \mathbf{V}^\ell\right) \cdot \mathbf{z}^\ell$, where the matrix $\mathbf{V}^\ell = \text{BKWTA}(\mathbf{g}_t^\ell, K) \cdot \text{BKWTA}((\mathbf{g}_t^\ell)^T, K)$ and $\text{BKWTA}(\mathbf{v}, K)$ is the binarized K winners-take-all function, yielding a binary vector with 1 at the index of each of the K winning units (in the Appendix, we denote this as "Lat1-S-NCN"),
3. $f^\ell(\mathbf{z}^\ell, \mathbf{g}_t^\ell) = \max\left(0, \mathbf{z}^\ell - \left((\mathbf{V}^\ell \odot (\mathbf{g}_t^\ell \cdot (\mathbf{g}_t^\ell)^T)) \cdot \mathbf{z}^\ell\right)\right)$, where: $\mathbf{V}_{i,j}^\ell = \{\alpha, \text{if } i \neq j, \text{else, } 0\}$ (in the Appendix, we denote this as "Lat2-S-NCN").

In the last two forms of the competition function, we see that lateral inhibition is a function of an evolving context vector $\mathbf{g}^\ell$, triggered by the presence of the task signal t_i. The above competition functions correspond to different designs of lateral suppression patterns: (1) corresponds to no lateral suppression, (2) corresponds to shutting off neurons that are not task-relevant driven by a task selector, (3) corresponds to a task-driven, real-valued lateral matrix that scales neural activities.

On the custom benchmarks (including both orderings #1 and #2), we evaluate four variations of the S-NCN: 1) an S-NCN, with hyperbolic tangent activations and no context-dependent lateral inhibition (S-NCN), 2) an S-NCN with sparsity created by the use of a linear rectifier activation function and no lateral inhibition (S-NCN-relu), 3) an S-NCN with the second variant of our proposed lateral inhibition (Lat1-S-NCN), and 4) an S-NCN with the third variant of our proposed lateral inhibition (Lat2-S-NCN) (All variants used: $\beta = 0.05$, $K = 10$, $\eta_g = 0.9$, $\eta_e = 0.01$, $\alpha = 0.98$). The last two S-NCN models ("Lat1-S-NCN" and "Lat2-S-NCN") were driven by the FNBG model that we described in Section 3.3 ("The Neural Task Selection Model").

[5] We measure CBWT for task T_1, since this measures total forgetting over the full length of the task sequence.

Table 5: Alternative metrics reported for task sequence orderings #1 and #2 (higher values are better).

	Ordering #1 (High Color Sim.)				**Ordering #2 (Low Color Sim.)**			
	Equal		**Unequal**		**Equal**		**Unequal**	
	TBWT	**CBWT**	**TBWT**	**CBWT**	**TBWT**	**CBWT**	**TBWT**	**CBWT**
Backprop	-0.426	-0.358	-0.547	-0.496	-0.422	-0.566	-0.602	-0.639
EWC	-0.409	-0.332	-0.516	0.477	-0.400	-0.521	-0.599	-0.611
Md-IMM	-0.388	-0.296	-0.481	-0.390	-0.355	-0.411	-0.521	-0.429
DT+Md-IMM	-0.342	-0.281	-0.466	-0.355	-0.340	-0.399	-0.491	-0.389
L2+DT+Md-IMM	-0.301	-0.250	-0.422	-0.318	-0.281	-0.355	-0.401	-0.378
HAT	-0.277	-0.252	-0.341	-0.291	-0.200	-0.341	-0.285	-0.328
S-NCN	-0.397	-0.509	-0.507	-0.623	-0.373	-0.252	-0.564	-0.396
Lat1-S-NCN	$\mathbf{-0.048}$	$\mathbf{-0.074}$	$\mathbf{-0.081}$	$\mathbf{-0.048}$	$\mathbf{-0.032}$	$\mathbf{-0.150}$	$\mathbf{-0.086}$	$\mathbf{-0.037}$
Lat2-S-NCN	-0.226	-0.304	-0.270	-0.310	-0.145	-0.215	-0.198	-0.273

Table 6: Generalization metrics (10 trials) for sequence orderings # 1 & #2 (higher values are better).

	Ordering #1: $\{M1, M2, GD1, FM1, FM2, GD2\}$ **(High Color Sim.)**			
	Equal		**Unequal**	
	ACC	**BWT**	**ACC**	**BWT**
Backprop	0.241 ± 0.050	-0.759 ± 0.030	0.185 ± 0.048	-0.791 ± 0.048
Backprop+DO	0.251 ± 0.049	-0.711 ± 0.030	0.178 ± 0.049	-0.733 ± 0.049
EWC	0.280 ± 0.023	-0.714 ± 0.030	0.185 ± 0.046	-0.726 ± 0.039
EWC+DO	0.231 ± 0.029	-0.687 ± 0.029	0.184 ± 0.044	-0.710 ± 0.038
Mean-IMM	0.279 ± 0.019	-0.465 ± 0.024	0.210 ± 0.041	-0.499 ± 0.043
Md-IMM	0.521 ± 0.027	-0.392 ± 0.023	0.480 ± 0.039	-0.240 ± 0.040
DT+Mean-IMM	0.321 ± 0.023	-0.430 ± 0.020	0.300 ± 0.044	-0.471 ± 0.044
DT+Md-IMM	0.530 ± 0.024	-0.387 ± 0.021	0.551 ± 0.042	-0.220 ± 0.042
L2+Mean-IMM	0.301 ± 0.022	-0.443 ± 0.022	0.250 ± 0.038	-0.492 ± 0.046
L2+Md-IMM	0.491 ± 0.020	-0.376 ± 0.023	0.480 ± 0.039	-0.235 ± 0.041
L2+DT+Mean-IMM	0.354 ± 0.029	-0.390 ± 0.021	0.351 ± 0.039	-0.421 ± 0.046
L2+DT+Md-IMM	0.532 ± 0.025	-0.237 ± 0.027	0.520 ± 0.040	-0.240 ± 0.045
HAT	0.550 ± 0.019	-0.211 ± 0.020	0.492 ± 0.031	-0.231 ± 0.036
S-NCN (ours)	0.421 ± 0.022	-0.408 ± 0.026	0.352 ± 0.016	-0.476 ± 0.020
S-NCN-relu (ours)	0.398 ± 0.009	-0.430 ± 0.012	0.352 ± 0.008	-0.470 ± 0.011
Lat1-S-NCN (ours)	$\mathbf{0.716 \pm 0.013}$	$\mathbf{-0.031 \pm 0.017}$	$\mathbf{0.713 \pm 0.011}$	$\mathbf{-0.041 \pm 0.012}$
Lat2-S-NCN (ours)	0.573 ± 0.020	-0.236 ± 0.0258	0.554 ± 0.038	-0.235 ± 0.042

	Ordering #2: $\{GD2, M1, FM2, M2, GD1, FM1\}$ **(Low Color Sim.)**			
	ACC	**BWT**	**ACC**	**BWT**
Backprop	0.303 ± 0.030	-0.644 ± 0.037	0.287 ± 0.043	-0.671 ± 0.043
Backprop+DO	0.285 ± 0.032	-0.587 ± 0.031	0.266 ± 0.042	-0.610 ± 0.044
EWC	0.303 ± 0.031	-0.643 ± 0.033	0.291 ± 0.039	-0.663 ± 0.047
EWC+DO	0.302 ± 0.033	-0.558 ± 0.032	0.281 ± 0.039	-0.586 ± 0.046
Mean-IMM	0.453 ± 0.026	-0.170 ± 0.031	0.402 ± 0.036	-0.274 ± 0.035
Md-IMM	0.584 ± 0.027	-0.091 ± 0.030	0.533 ± 0.034	-0.230 ± 0.036
DT+Mean-IMM	0.558 ± 0.021	-0.128 ± 0.029	0.510 ± 0.033	-0.254 ± 0.035
DT+Md-IMM	0.591 ± 0.020	-0.088 ± 0.032	0.528 ± 0.036	-0.211 ± 0.039
L2+Mean-IMM	0.465 ± 0.021	-0.156 ± 0.033	0.430 ± 0.039	-0.271 ± 0.032
L2+Md-IMM	0.576 ± 0.028	-0.99 ± 0.038	0.511 ± 0.036	-0.266 ± 0.039
L2+DT+Mean-IMM	0.587 ± 0.025	-0.105 ± 0.033	0.528 ± 0.038	-0.253 ± 0.043
L2+DT+Md-IMM	0.630 ± 0.029	-0.076 ± 0.030	0.551 ± 0.037	-0.201 ± 0.041
HAT	0.596 ± 0.026	-0.114 ± 0.029	0.563 ± 0.031	-0.210 ± 0.044
S-NCN (ours)	0.444 ± 0.017	-0.393 ± 0.0210	0.272 ± 0.013	-0.587 ± 0.014
S-NCN-relu (ours)	0.431 ± 0.009	-0.398 ± 0.010	0.286 ± 0.007	-0.559 ± 0.008
Lat1-S-NCN (ours)	$\mathbf{0.721 \pm 0.014}$	$\mathbf{-0.042 \pm 0.013}$	$\mathbf{0.667 \pm 0.011}$	$\mathbf{-0.097 \pm 0.013}$
Lat2-S-NCN (ours)	0.633 ± 0.028	-0.170 ± 0.033	0.5778 ± 0.035	-0.211 ± 0.042

We compare our S-NCN model (specifically, our best-performing one from the experiment in the last section – the Lat1-S-NCN) to the following approaches that have been proposed over the years to combat catastrophic interference: Naive rehearsal with memory (NR+M1 & NR+M2), EWC, synaptic intelligence (SI), MAS [59], Lwf [60], GEM [61], DGR [86], Rtf [21], ICarl [62], Lucir [63], Bic [87], and Mnemonics [64]. In Table 7, we report model ACC and BWT, averaged over 10 trials, offering not only an extensive and comprehensive comparison of competitive methods, but also demonstrating that, for all three benchmarks, **our proposed S-NCN outperforms all of them**, demonstrating the power afforded by challenging the very assumptions underlying modern-day artificial neural systems and designing models with stronger grounding in neuro-biology. Furthermore, it is astounding to see that the S-NCN outperforms/matches performance with not only the single-head models but also with the multi-head models (except GEM), which enjoy an easier version of the problem since they are permitted to utilize a different classifier per task. Finally, it is critical to note that **our proposed S-NCN is a complementary neural system that learns without explicitly-provided task descriptors**, i.e., in other words, the model learns to compose its own task contexts in a data-dependent manner.

Note that, in the online setting, split FashionMNIST appears to be simpler, given that the S-NCN readily learns to generate rough forms of bigger objects (shirt/shoes/pants) and associate these with labels early whereas digits/characters are a bit more intricate and take longer to learn.

Table 7: Generalization metrics (10 trials) for Split MNIST, Split Fashion MNIST (FMNIST) and Not-MNIST benchmarks. Note for IMM, we employ the best performing variant, *L2+DT+Md-IMM*. Above dashed line are single-head models & below are multi-head models (except the S-NCN).

	MNIST		**FMNIST**		**NotMNIST**	
	ACC	**BWT**	**ACC**	**BWT**	**ACC**	**BWT**
EWC	0.760 ± 0.030	-0.210 ± 0.011	0.739 ± 0.020	-0.201 ± 0.011	0.790 ± 0.020	-0.176 ± 0.010
VCL	0.980 ± 0.210	-0.003 ± 0.002	0.980 ± 0.20	-0.002 ± 0.003	0.953 ± 0.003	-0.004 ± 0.002
IMM	0.951 ± 0.018	-0.007 ± 0.003	0.950 ± 0.013	-0.005 ± 0.003	0.925 ± 0.011	-0.006 ± 0.002
HAT	0.972 ± 0.011	-0.040 ± 0.002	0.968 ± 0.011	-0.004 ± 0.002	0.942 ± 0.009	-0.005 ± 0.002
GEM	0.922 ± 0.110	$\mathbf{+0.001 \pm 0.002}$	0.930 ± 0.12	$+0.001 \pm 0.003$	0.919 ± 0.021	-0.003 ± 0.002
DGR	0.911 ± 0.300	-0.011 ± 0.002	0.915 ± 0.25	-0.013 ± 0.001	0.920 ± 0.015	-0.014 ± 0.003
Rtf	0.925 ± 0.200	-0.009 ± 0.003	0.930 ± 0.25	-0.009 ± 0.005	0.922 ± 0.012	-0.011 ± 0.004
EWC	0.190 ± 0.030	-0.357 ± 0.015	0.199 ± 0.06	-0.350 ± 0.012	0.186 ± 0.020	-0.361 ± 0.010
NR+M1	0.906 ± 0.870	-0.050 ± 0.001	0.900 ± 0.810	-0.060 ± 0.003	0.890 ± 0.030	-0.071 ± 0.004
NR+M2	0.950 ± 0.470	-0.100 ± 0.003	0.948 ± 0.380	-0.090 ± 0.003	0.880 ± 0.028	-0.103 ± 0.002
SI	0.197 ± 0.110	-0.367 ± 0.014	0.198 ± 0.10	-0.370 ± 0.013	0.161 ± 0.030	-0.370 ± 0.010
MAS	0.195 ± 0.290	-0.340 ± 0.010	0.180 ± 0.25	-0.340 ± 0.010	0.178 ± 0.060	-0.341 ± 0.011
Lwf	0.846 ± 0.340	-0.120 ± 0.001	0.875 ± 0.30	-0.130 ± 0.003	0.626 ± 0.091	-0.130 ± 0.004
ICarl	0.940 ± 0.410	-0.100 ± 0.004	0.960 ± 0.40	-0.110 ± 0.005	0.887 ± 0.102	-0.109 ± 0.007
Lucir	0.940 ± 0.310	-0.103 ± 0.007	0.950 ± 0.34	-0.110 ± 0.005	0.935 ± 0.093	-0.101 ± 0.006
Bic	0.901 ± 0.860	-0.139 ± 0.009	0.890 ± 0.85	-0.160 ± 0.009	0.851 ± 0.099	-0.155 ± 0.009
Mnem	0.960 ± 0.320	-0.991 ± 0.005	0.968 ± 0.30	$\mathbf{+0.007 \pm 0.006}$	0.950 ± 0.071	-0.011 ± 0.007
S-NCN	$\mathbf{0.981 \pm 0.300}$	-0.005 ± 0.004	$\mathbf{0.982 \pm 0.400}$	-0.003 ± 0.007	$\mathbf{0.957 \pm 0.400}$	$\mathbf{-0.004 \pm 0.005}$

S-NCN Task Accuracy Curves

See Figure 2 (below) for a visual depiction of the S-NCN's task accuracy over time across tasks for all three continual learning benchmarks investigated in this paper.

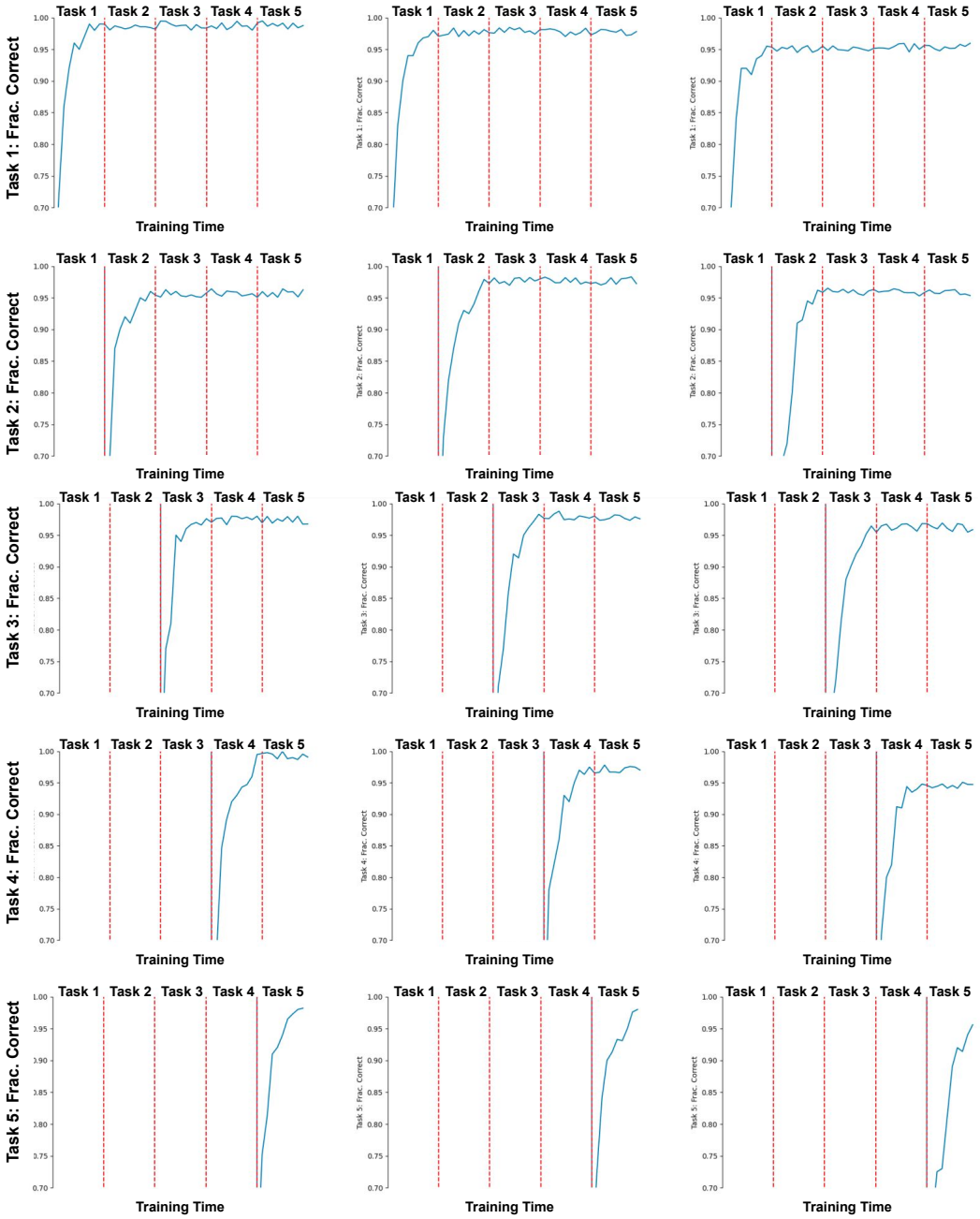

Figure 2: Task accuracy curves for the S-NCN measured across the five tasks within the (Left column) Split MNIST, (Middle column) Split Fashion MNIST, and (Right column) Split NotMNIST benchmarks (red vertical lines indicate actual task boundaries). Y-axis depicts the fraction correct while the x-axis depicts (online) training iteration (or one single epoch/pass through each task). Each row, as indicated by the Y-axis label, represents the perspective of a different task, e.g., row 1 corresponds to performance on Task 1 as the S-NCN learns across all tasks, row 2 corresponds to performance on Task 2 as the S-NCN learns across all tasks, etc.