Layerwise Perspective into Continual Backpropagation

Replacing the First Layer is All You Need

Augustinas Jučas

Supervisor(s): Wendelin Böhmer[1], Laurens Engwegen[1]

[1]EEMCS, Delft University of Technology, The Netherlands

A Thesis Submitted to EEMCS Faculty Delft University of Technology,
In Partial Fulfilment of the Requirements
For the Bachelor of Computer Science and Engineering
June 22, 2025

Name of the student: Augustinas Jučas
Final project course: CSE3000 Research Project
Thesis committee: Wendelin Böhmer, Laurens Engwegen, Megha Khosla

Abstract

Continual learning faces a problem, known as plasticity loss, where models gradually lose the ability to adapt to new tasks. We investigate Continual Backpropagation (CBP) – a method that tackles plasticity loss by constantly resetting a small fraction of low-utility neurons. We find that resetting neurons in deeper layers gives increasingly worse performance, with exclusively first-layer resets achieving performance very close to regular CBP. We confirm this phenomenon holds across different models. Additionally, we find an underlying reason for this phenomenon: first-layer resets prevent continual growth in weight magnitudes, which is crucial for maintaining plasticity, while not resetting the first layer results in strong weight growth. Additionally, we find that CBP fails under models based on non-ReLU activations, which is a novel result.[1]

1 Introduction

Neural networks perform well in static domain settings, where the data distribution remains fixed across training and evaluation. However, in continual learning scenarios—where the data distribution changes over time—models suffer substantial performance degradation [1–3]. This degradation is primarily related to two core challenges: catastrophic forgetting, a rapid loss of knowledge about previously learned tasks [2], and plasticity loss, a gradual reduction in a model's capacity to learn new information as training progresses [1, 3]. We depict an instance of plasticity loss in Figure 1. While catastrophic forgetting has been widely studied in the literature, plasticity loss remains comparatively underexplored, despite being core to continual adaptation [1].

Many already existing Machine Learning (ML) methods can be used to mitigate the plasticity problem, such as L_2 regularization or the Shrink-and-Perturb method [3]. In our work, we investigate a novel method, introduced by Dohare et al. – Continual Backpropagation (CBP) – a simple yet effective version of stochastic gradient descent (SGD) that periodically resets low-utility neurons *across the whole network*. Our goal is to develop a better understanding of CBP, in particular, from a *layerwise perspective*. We aim to examine at which layers of the model CBP is most effective, as well as the underlying reasons for that, thus uncovering new insights into how continual learning dynamics vary across the depth of a neural network.

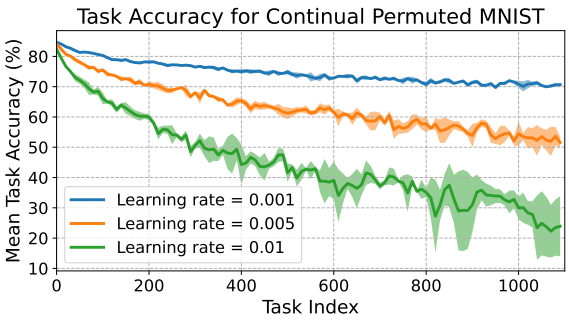

Figure 1: An example of plasticity loss. A model is trained consecutively on different datasets (tasks) of similar complexity, using regular backpropagation. It can be observed that as the model is trained, its accuracy on the current task keeps decreasing due to plasticity loss.

[1]The data and code are available at https://github.com/augustinasjucas/first-layer-cbp/.

Our contributions are the following:

1. We observe that during CBP, resetting neurons in earlier layers leads to increasingly better performance. The earlier a layer is, the more plasticity a neural network can retain by resetting that layer.
2. We show that during CBP, continually resetting neurons only in the first hidden layer achieves nearly the same performance as resetting neurons in all layers. We call this *the first layer phenomenon*.
3. We show that the first layer phenomenon can be explained through the lens of a model's weight magnitudes. In particular, replacement in the first layer is necessary and sufficient to prevent the model's weights (in all layers) from continually increasing, which is necessary for maintaining plasticity.
4. We propose a lightweight variant of Continual Backpropagation that performs resets exclusively in the first layer – such a method is more efficient and results in very similar performance.
5. We find an underlying flaw in Continual Backpropagation: the training algorithm does not reduce plasticity loss for models which use non-ReLU activation functions.

2 Background and Related Work

Continual learning (CL) refers to a model's capacity to learn from a data stream whose distribution changes over time, thus avoiding infeasible or expensive retraining from scratch. This capability is critical in reinforcement learning, where agents must adapt to non-stationary environments while retaining previously acquired policies [4] and in robotics, for achieving lifelong autonomy and incremental skill acquisition [5]. A notorious challenge in the field of CL is catastrophic forgetting – the tendency of models to lose previously learned knowledge when learning new information. For this problem, the ML community has developed solutions such as Elastic Weight Consolidation [6], replay-based methods like Gradient Episodic Memory [7], and others.

However, continually trained neural networks also suffer from another serious issue – a gradual loss of plasticity, in which the ability to learn new information diminishes with each successive task. This decline is observed in supervised settings in the form of a training slowdown after several hundred tasks [3], and in deep RL agents, as an increased number of collapsed activations and vanishing gradients over time [4]. Several hypotheses have been proposed to explain plasticity decline. Lyle et al. claim that inappropriate loss functions

can make the optimization landscape less smooth [8], thus reducing learning capacity. Other work attributes plasticity reduction to a loss of curvature in the loss landscape during continual training [9]. Additional studies report an increase in *dormant* neurons, which do not change much over time and also do not contribute meaningfully to performance as training progresses [4, 10].

Even though the dynamics behind plasticity loss are not fully understood, a variety of methods for its mitigation exist. In supervised settings, a common approach is *Shrink-and-Perturb*, which applies L_2 regularization around the initial weights and periodically perturbs the parameters to prevent converging into regions that slow down learning [3]. Another method, used in reinforcement learning, is related to using Concatenated ReLU (CReLU) activation functions, which expand feature maps to reactivate dormant units, improving adaptability in non-stationary environments [4]. Other techniques include parameter regularization, feature rank regularization, and alternative activation functions [11]. In our work, we focus on Dohare et al.'s method, which introduces *Continual Backpropagation* to address this issue [3, 12].

Dohare et al. observe that plasticity is linked to the weight values being close to the network's initial weight distribution, which typically supports rapid learning and stability [3]. As continual training progresses, many neurons drift away from their initial values, with some becoming dormant. To address both of these problems, the authors propose *Continual Backpropagation*, where a small fraction of the least useful neurons (identified by a layerwise *contribution utility score*) are reinitialized at each step. This targeted reinitialization maintains high learning rates across thousands of tasks, matching the performance of freshly initialized models [3]. However, their work does not include a detailed analysis of the algorithm's internal workings, a gap we aim to address in part in our work.

In both Dohare et al.'s work and most related literature, plasticity is evaluated and investigated at the entire network level [1, 3]. It remains relatively unexplored how important different parts of the network are to the plasticity performance even though some work on this topic exists [13–15]. In this paper, we address this gap by analyzing the effects of differentiating the replacement rates per layer in the CBP setting, thus being able to observe what layers have the largest impact on performance and other metrics.

3 Method

In this section, we present our approach for investigating the layerwise dynamics of Continual Backpropagation. The first subsection presents a modification of CBP that enables our analysis, along with a step-by-step pipeline for training models and collecting the core data. The second subsection describes all metrics tracked during the training process.

3.1 Layerwise Replacement Analysis

It is well known in the literature that earlier layers of neural networks tend to perform lower-level, concrete feature extraction, while deeper layers extract and represent increasingly general and abstract features [16]. Additionally, feature representations learned by deeper layers are directly dependent on those learned in previous layers. Motivated by this asymmetric dependence in the network's hierarchy, we investigate how reinitialising neurons at different depths affects CBP algorithm performance, since the impact may vary by layer depth. Therefore, we modify the CBP algorithm to accommodate layerwise replacement rates, as described in Algorithm 1, which enables selective neuron replacement within specific layers.

Algorithm 1 Layerwise-adjusted CBP algorithm for an MLP with L hidden layers. The algorithm is based on [12], with modifications to the *Set* and the inner *for* loop steps.

Set: step-size α, replacement rate ρ, **a set of layers** $\mathbf{S} = \{\mathbf{l_1}, \mathbf{l_2}, ..., \mathbf{l_k}\}$ decay rate η, and maturity threshold m.
Initialize: Initialize the weights $w_0, \ldots, w_L$. Let w_l be sampled from a distribution d_l
Initialize: Utilities $u_1, \ldots, u_L$, average feature activation $f_1, \ldots, f_t$, and ages $a_1, \ldots, a_L$ to 0
for each input x_t **do**
  Forward pass: pass input through the network, get the prediction $\hat{y}_t$
  Evaluate: Receive loss $l(x_t, \hat{y}_t)$
  Backward pass: update the weights using stochastic gradient descent
  for layer l in S **do**
    Update age: a_l += 1
    Update feature utility: Using Equations 4, 5, and 6 from [12].
    Find eligible features: Features with age more than m
    Features to replace: $n_l * \rho$ of eligible features with smallest utility, let their indices be r
    Initialize input weights: Reset the input weights $w_{l-1}[r]$ using samples from d_l
    Initialize output weights: Set $w_l[r]$ to zero
    Initialize utility, feature activation, and age: Set $u_{l,r,t}$, $f_{l,r,t}$, and $a_{l,r,t}$ to 0
  end for
end for

We further propose a **pipeline** for investigating layerwise CBP replacement rates. This pipeline consists of 3 actionable steps.

Step 1: Experiment Specification
The first step of investigating the layerwise dynamics of Continual Backpropagation is specifying the broad experimental settings which will be investigated. This primarily requires selecting several dataset-model pairs for further investigation.

Step 2: Hyperparameter Search
The second step involves performing a hyperparameter search to identify optimal configurations for the regular CBP algorithm and baseline methods across all experimental settings defined in the previous step. We use both unregularized and L_2-regularized backpropagation as baseline algorithms, abbreviated as BP and L_2 throughout this paper.

To be precise, for every experimental setting, this step requires finding the optimal replacement rate and learning rate configuration for regular CBP (denote as $repl_{CBP}$ and lr_{CBP}), the optimal weight decay and learning rate configuration for the L_2 baseline, and the optimal learning rate for

BP. Hyperparameters are compared using the mean online accuracy/error over the last 15% of data points.

Step 3: Running Modifications of CBP
After completing a hyperparameter search for the baselines, the final step is to run the experiments using different versions of CBP. These versions differ only in the choice of the set S, which indicates the layers where CBP replacement should be applied (see Algorithm 1). We list the specific CBP variants to be run, by describing the sets S used in each case, that is, which layers are selected for replacement:

- The sets $\{1\}$, $\{2\}$, ..., $\{L\}$: for every layer, only it gets replaced.
- The set $\{2, 3, \ldots, L\}$: all layers except the first get replaced.
- The set $\{1, 2, \ldots, L\}$: all layers get replaced, equivalent to regular CBP.
- The set $\emptyset$: no layers get replaced, equivalent to regular backpropagation.

All CBP variants are executed for each experimental configuration from Step 1, tracking the metrics defined in Section 3.2. To reduce computational expenses, additional hyperparameter searches are not performed for these CBP variations. Instead, we reuse the optimal replacement rate $repl_{CBP}$ and learning rate lr_{CBP}. Following this step, the collected metric data allows us to perform ad-hoc analysis.

3.2 Metrics

During training, we monitor numerous metrics for individual layers and across the entire network to answer questions and test hypotheses. These tracked metrics include:

- **Global performance.** The online accuracy/error is tracked as training progresses, as is done in the original work by Dohare et. al [3].
- **Weight Norm Statistics.** The L_1 norm for each layer's flattened weight matrix over time is tracked, as well as that of the whole network.
- **Dead Neuron Counts.** The fraction of neurons that output 0 activation across a large sample of the training dataset is measured. Such a neuron is considered *dead*, since the derivative of the loss with respect to the outgoing weights of a neuron that outputs an activation a are proportional to that activation a. Thus, if a is 0, the gradients of the outgoing weights of the neuron are killed and the neuron stops adapting. This number is tracked for every layer separately.
- **Feature Ranks.** When a task changes, a large batch of samples from the next task is sampled, passed through the model, and activation maps for every layer are computed. Then, for each layer, the effective rank of the subspace spanned by the features of that layer is computed.
- **Utility Scores.** During training, the mean utility scores are tracked for every layer, defined by Dohare et al. [3] as:

$$\mathbf{u}_l[i] = \eta \times \mathbf{u}_l[i] + (1 - \eta) \times |\mathbf{h}_{l,i}| \times \sum_{k=1}^{n_{l+1}} |\mathbf{w}_{l,i,k}| \quad (1)$$

i.e., for every layer l, the following is tracked:

$$\overline{\mathbf{u}}_l = \frac{1}{n_l} \sum_{i=1}^{n_l} \mathbf{u}_l[i] \quad (2)$$

Here η denotes an exponentially moving average coefficient, $\mathbf{h_{l,i}}$ denotes activation value of neuron i in layer l, and $\mathbf{w}_{l,i,k}$ denotes the weight value of the weight between layer l and $l + 1$ between neurons i and k.

- **Curvature of the Loss Landscape**. The effective rank of the network's approximate Hessian is tracked, as defined by Lewandowski et al. [17], who attribute plasticity loss to a reduction of directions of curvature in the loss landscape.
- **Probing accuracies.** We present a method we use in parallel of feature ranks, to quantify how useful the features learned by the intermediate layers in a network are. After every task, the network is frozen and a linear classifier is attached at each hidden layer. Then, each linear classifier is trained separately on the **next task's** data, until convergence. The training set accuracies are reported for each layer.

  Using this layerwise data, we can quantify how useful the intermediate features learned in the current task are for the next task. Higher accuracies indicate that the learned features contain some useful information for the subsequent task, while low feature utility suggests less usefulness and may indicate that larger weight changes will be needed when learning the next task.

4 Experimental Setup and Results

To investigate the layerwise effects of CBP, we use a customised Continual Permuted MNIST experimental setting, following Dohare et al. [3, 12]. We first describe the exact experimental setup and explain how we apply the investigation protocol from Section 3.1. We then present our analysis of the results and describe follow-up experiments that provide deeper insights.

We also aimed to perform this analysis for another experimental setting used by Dohare et al. [3, 12], but we were unable to obtain meaningful results with a deeper model. These results are described in Appendix A.

4.1 Experimental Setting

Dataset and Base Model
We construct a continually changing data distribution using the Permuted MNIST dataset, similar to what was done by Dohare et al. [12]. In particular, we take the MNIST dataset, sample T random pixel permutations and construct T tasks, each being a version of the MNIST dataset with the pixels permuted using that task's permutation. An example of this procedure being done for a single digit is depicted in Figure 2. Additionally, due to computational constraints, we

subsample every task to be 6 times smaller than the original MNIST dataset, therefore every task contains exactly 10000 samples. By default, we use $T = 1100$ for full experiments and $T = 600$ for hyperparameter search.

Figure 2: A single sample from MNIST, changing across multiple Continual Permuted MNIST tasks.

As for the neural network to train, while we vary the model across different experiment settings (described later), we set the **base neural network** to be a deeper, 5-hidden-layer MLP (Multi Layer Perceptron) with 100 neurons in every hidden layer and ReLU activations. We use only 100 hidden layers, since such a configuration saves computation time significantly.

Reproducing Core Results

Having changed the experimental setting to a smaller dataset and a smaller (yet deeper) model, we first show that the core results from the original paper [3] are still obtained for the base network. In particular, we care about three training algorithms, as described in Section 3.1: regular CBP, BP and L_2, hence we perform an initial hyperparameter search to find the optimal settings for each algorithm. For BP the optimal learning rate is chosen from the set $LR = \{0.01,\ 0.003,\ 0.001\}$; for CBP the optimal learning rate and replacement rate combination is chosen with learning rate selected from LR and the replacement rate from $\{10^{-5},\ 10^{-4},\ 10^{-3}\}$; for L_2 the optimal learning rate and weight decay combination is chosen with the learning rate selected from LR and weight decay from $\{10^{-4},\ 10^{-3},\ 10^{-2}\}$. The performances are compared by selecting the best accuracy for the last 15% of tasks. Also, all experiments are run with 4 random seeds. The best version for each algorithm is depicted in Figure 3. It can be seen that even with the very best versions of each algorithm, Continual Backprop, as expected, outperforms BP and L_2, by not losing accuracy over multiple tasks, while L_2 and BP algorithms both decrease in accuracy over time. Hence, we get the expected results and conclude that our base setting is valid.

Model Configurations

After verifying that the base network setting is valid, we begin applying our investigative method described in Section 3.1. The protocol's first step requires defining the experimental settings. To ensure any results obtained will generalise across various neural network architectures, we investigate layerwise dynamics across different model sizes and types. Therefore, we examine:

- **Varying model width**. Keeping the base neural network (5 hidden layer MLP, ReLU activation 100 neurons per hidden layer), we only vary the widths of all layers, trying out the following widths: $\{20,\ 50,\ 100,\ 150,\ 200\}$.

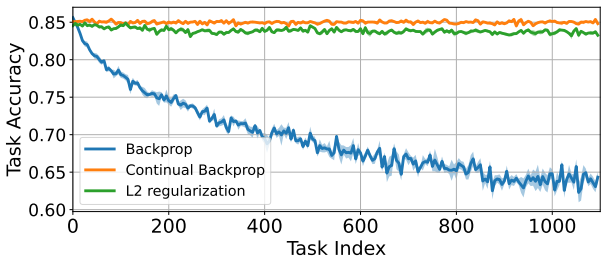

Figure 3: A figure comparing the accuracy evolution of the best-performing version of all 3 training algorithms on the base neural network (defined in Section 4.1) after hyperparameter search. The shaded regions denote a single standard deviation across 4 different runs of the same experiment.

- **Varying model depth**. Keeping the base neural network, we vary the depth of the model, trying out a model with 2, 5 and 8 hidden layers.
- **Varying model shape**. Keeping the same activation function (ReLU), we make the model deeper and shallower, but keep the number of parameters almost equal, hence trying out three options:
  - 2 hidden layers, 86 neurons per layer, inducing 120.7k parameters.
  - 5 hidden layers, 100 neurons per layer, inducing 119.9k parameters.
  - 8 hidden layers, 129 neurons per layer, inducing 119.3k parameters.
- **Varying activation function**. Keeping the base neural network, we vary the activation function between ReLU, hyperbolic tangent (Tanh), and Scaled Exponential Linear Unit (SELU).

As a result, a total of $(5 + 3 + 3 + 3) - 3 = 11$ separate experimental settings are obtained. The 3 is subtracted, since all 4 experiment types share a single experimental setting (i.e., a neural network configuration) – the base setting, described previously in *Dataset and Base Model* subsection.

For all 11 of these models, the optimal hyperparameters for regular CBP, BP and L_2 algorithms are found, as described in Step 2 of our protocol (Section 3.1). Due to limited computational resources, the experiments were run for 600 tasks for the hyperparameter search. The sets of hyperparameters which were tried are provided in Appendix C, where all attempted hyperparameters are depicted in Table 4 and the found optimal hyperparameters are shown in Table 3.

Note that for running the experiments, we used CPU nodes at the Delft University of Technology DelftBlue high-performance cluster [18], as well as DAIC cluster [19].

4.2 Layerwise Replacement Results

After finding the optimal hyperparameters, modified versions of CBP with various layerwise replacement rate distributions are run for all 11 models, as defined in Section 3.1, Step 3. The core results are presented below.

Table 1: The effects of varying replacement rates by layer on different neural networks. Every row corresponds to a particular network defined in Section 4.1, under *Model Configurations*. Every column corresponds to a different version of CBP that the neural network was trained on. All numeric values depict accuracies in percentages, hence **higher is better**. The accuracies correspond to an average online accuracy of the last 15% of tasks, averaged over 3 runs. For the errors, we show a single standard deviation. We underline the best mean accuracies for every row, not including the *All Layers* column, to indicate which layer is best for replacement.

	Baseline	Replacing Individual Layers			
	All Layers	**Layer 1**	**Layer 2**	**Layer 3**	**Last Layer**
Varying Widths					
Width 20	*79.2 ± 0.1*	76.1 ± 0.2	59.5 ± 1.3	58.2 ± 0.6	59.1 ± 0.9
Width 50	*82.3 ± 0.1*	80.7 ± 0.2	56.1 ± 2.3	60.2 ± 1.0	52.4 ± 2.3
Width 100	*84.6 ± 0.0*	83.2 ± 0.1	71.0 ± 0.5	67.6 ± 0.4	61.0 ± 1.0
Width 150	*85.5 ± 0.0*	84.2 ± 0.0	75.2 ± 0.1	71.1 ± 0.3	66.6 ± 0.3
Width 200	*86.0 ± 0.0*	84.8 ± 0.0	77.7 ± 0.3	74.0 ± 0.2	69.9 ± 0.4
Varying Activation Functions					
ReLU	*84.6 ± 0.0*	83.2 ± 0.1	71.0 ± 0.5	67.6 ± 0.4	61.0 ± 1.0
SELU	*76.2 ± 0.1*	76.3 ± 0.1	76.2 ± 0.1	76.6 ± 0.2	76.5 ± 0.1
Tanh	*69.2 ± 0.3*	69.1 ± 0.5	69.1 ± 0.1	69.0 ± 0.5	68.6 ± 0.3
Varying Depths					
Depth 2	*87.0 ± 0.0*	86.8 ± 0.1	81.9 ± 0.0	-	81.9 ± 0.0
Depth 5	*84.6 ± 0.0*	83.2 ± 0.1	71.0 ± 0.5	67.6 ± 0.4	61.0 ± 1.0
Depth 8	*82.9 ± 0.0*	82.5 ± 0.1	71.7 ± 0.2	67.6 ± 0.8	65.5 ± 0.2
Varying Depths, Same Number of Parameters					
Depth 2	*87.3 ± 0.0*	87.1 ± 0.1	82.7 ± 0.0	-	82.7 ± 0.0
Depth 5	*84.6 ± 0.0*	83.2 ± 0.1	71.0 ± 0.5	67.6 ± 0.4	61.0 ± 1.0
Depth 8	*82.3 ± 0.1*	82.2 ± 0.1	70.1 ± 0.5	65.7 ± 0.2	64.3 ± 0.6

Benefits of Layer Replacement Decrease with Depth

A key result emerges when each layer is replaced separately. Across all experimental settings, we find that **the deeper a layer is, the less performance benefit is obtained by replacing neurons only in it**. This result is shown in Table 1, where only a certain layer is replaced for each experimental setting. Consequently, the table demonstrates that in almost all experimental settings, replacing the first layer results in the highest accuracy. Furthermore, these accuracies are compared against full-replacement CBP (i.e., regular CBP) to show that replacing only the first layer already achieves surprisingly close performance.

In Figure 4, this exact result is shown in detail for a single model to provide intuition on how the online accuracies diverge during training when individual layers are replaced. Effectively, the figure corresponds to a single line in Table 2. When replacing only the first layer, performance decline is not only small but also very slow, especially compared to replacements in deeper layers. Additionally, exclusive first-layer replacement results in performance very close to L_2 regularisation and comparable to full CBP. From now on, we refer to the property that replacing only the first layer gives especially close results to regular CBP as **the first layer phenomenon**.

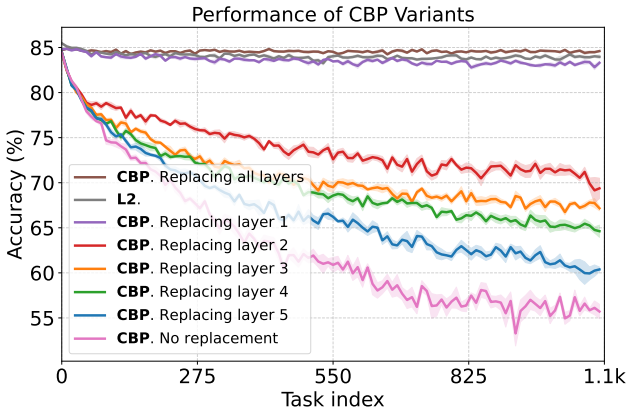

Figure 4: The effects of varying replacement rates by layer on the base neural network: an MLP with 5 hidden layers, 100 neurons each, with ReLU activation function, trained on Continual Permuted MNIST. The shaded areas correspond to a single standard deviation across 3 runs of the same experiment.

CBP Does Not Work on Non-ReLU Activations

Inspecting Table 1, the *SELU* and *Tanh* rows appear to be outliers compared to other models within the table as these two variations perform significantly worse in terms of accuracy even with regular, full-replacement CBP. We investigate this phenomenon further, comparing regular CBP to regular BP and the L_2 baseline. The results are depicted in Figure 5, showing that the performance of CBP on non-ReLU neural networks is identical to the performace of regular backpropagation, hence indicating that CBP only works on ReLU activations. Note that this is a novel result, as the original work by Dohare et al. [3] only investigates different activations on the previously described Bit-Flipping problem, which we have shown to be unreliable for different experimental setups in Appendix A.

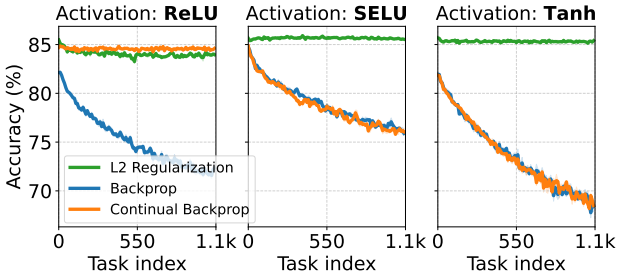

Figure 5: Online accuracy training curves for models with varying activation functions, comparing 3 training algorithms: CBP, BP and L_2. We observe that Continual Backprop only works with ReLU-based MLPs. Shaded areas indicate a single standard deviation over 3 runs.

4.3 Exploring the First Layer Phenomenon

In light of the results seen in the previous section, we focus on understanding the first layer phenomenon in a general setting.

Investigative Setup

To further investigate the first layer phenomenon, we design a more targeted experiment using four versions of CBP that contrast the first layer's contribution against the rest of the network. Therefore, we consider replacing:

1. **All Layers**: regular CBP, where all layers are replaced.
2. **Layer 1**: a version of CBP, where only the first layer is replaced.
3. **Layers 2-L**: a version of CBP where all layers are replaced except the first.
4. **No Layers**: a version of CBP, where no layers are replaced, so exactly equivalent to regular BP.

This experimental design allows the first layer's individual effect to be directly tested against the combined effect of all other layers, determining whether the first layer alone drives the observed phenomenon. And surprisingly, the accuracy results in Table 2 confirm the first-layer phenomenon even more strongly: replacing only the first layer significantly outperforms replacing all other layers combined. Note that SELU and Tanh variations are excluded from the table, since CBP does not work with these activation functions. This result establishes the first-layer phenomenon in its purest form, as the first layer appears to be *more important* than all other layers combined.

Having confirmed this phenomenon under such pure conditions, the metric data collected during these experiments is analysed to understand the mechanisms that make the first layer so important for CBP.

Table 2: Performances of 4 different versions of CBP on different neural networks. Every row corresponds to a different neural network. The numeric values depict average online accuracies over the last 15% of tasks.

	Layers Replaced			
	All Layers	**Layer 1**	**Layers 2-L**	**No Layers**
Varying Widths				
Width 20	79.2 ± 0.1	76.1 ± 0.2	62.9 ± 0.7	58.4 ± 1.0
Width 50	82.3 ± 0.1	80.7 ± 0.2	72.6 ± 0.1	45.3 ± 1.1
Width 100	84.6 ± 0.0	83.2 ± 0.1	78.3 ± 0.1	56.2 ± 0.8
Width 150	85.5 ± 0.0	84.2 ± 0.0	80.9 ± 0.0	61.0 ± 0.3
Width 200	86.0 ± 0.0	84.8 ± 0.0	82.3 ± 0.1	66.2 ± 0.2
Varying Depths				
Depth 2	87.0 ± 0.0	86.8 ± 0.1	82.0 ± 0.0	75.1 ± 0.4
Depth 5	84.6 ± 0.0	83.2 ± 0.1	78.3 ± 0.1	56.2 ± 0.8
Depth 8	82.9 ± 0.0	82.5 ± 0.1	75.5 ± 0.1	65.3 ± 0.5
Varying Depths, Same Number of Parameters				
Depth 2	87.3 ± 0.0	87.1 ± 0.1	82.6 ± 0.0	76.6 ± 0.2
Depth 5	84.6 ± 0.0	83.2 ± 0.1	78.3 ± 0.1	56.2 ± 0.8
Depth 8	82.3 ± 0.1	82.2 ± 0.1	74.2 ± 0.1	63.1 ± 0.6

Primary Cause: CBP Regularises Weight Magnitudes

To understand the underlying cause of the first layer phenomenon, we test various hypotheses with respect to metrics

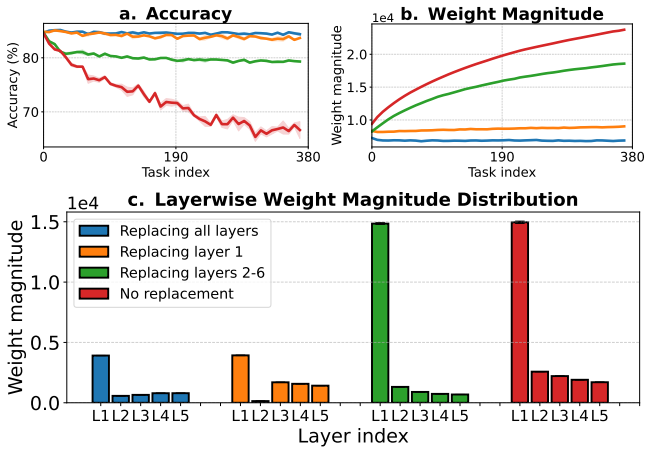

Figure 6: Results of training the base model (defined in Section 4.1) with 4 versions of CBP: 1. all layers are replaced, 2. only the first layer is replaced, 3. all except the first are replaced, or 4. no layers are replaced (BP). Subplot *a* shows how the average online training accuracies evolve over training. *b* depicts how the cumulative total magnitude of the whole model changes as training progresses. *c* shows the layerwise weight magnitudes for the 4 versions of CBP, at the end of training. Each bar in plot *c* represents the mean magnitude for some layer for the last 15% of the tasks, averaged over 3 runs. All experiments were run with 3 seeds; a single standard deviation is depicted as error.

such as the curvature of the loss space, feature ranks and utilities, the number of dead neurons, weight magnitudes and others. Most of these hypotheses proved false. For instance, no correlation is found between metrics that approximate feature usefulness (such as feature ranks or probing accuracies introduced in Section 3.2) and the performance results obtained in Table 2. In fact, for the vast majority of metrics, the CBP variant which replaces layers 2-L performs moderately to significantly better than the first-layer version of CBP, even though the actual accuracy differences strongly favour the first-layer replacement version of CBP. These failed results are described in detail in Appendix B, while only the results that do explain the phenomenon are presented here.

It is well known within CL literature that if a model's weights are not regularised while training, its weight magnitudes tend to increase as it gets trained on new tasks [20, 21]. And a model with inflated weight magnitudes tends to adapt more slowly to new tasks [21]. A potential reason for that, is that for increased weights, the gradient updates need to move the weights more to adjust to new tasks, therefore adaptation slows down. As a result, not allowing weight magnitudes to grow while continually training is necessary for maintaining plasticity. This is well justified by how much more slowly a model loses plasticity when simple L_2 regularisation is introduced, compared to regular backpropagation (e.g., see Figure 3).

A natural side effect of regular CBP is that it tends to act as a regularizer for weight magnitudes. This happens because CBP continually either resets certain weights to 0, or resam-

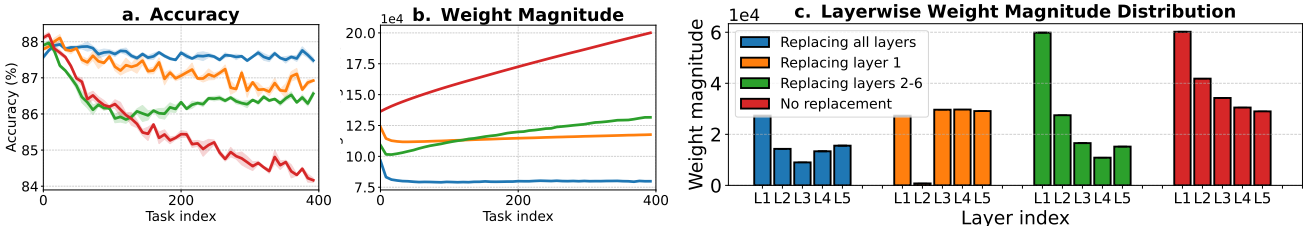

Figure 7: Results of training a variation of the base model (model defined in Section 4.1, only with 784 hidden neurons in every layer), with 4 versions of CBP: 1. all layers are replaced, 2. only the first layer is replaced, 3. all except the first are replaced, or 4. no layers are replaced (BP). Subplot *a* shows how the average online training accuracies evolve over training. *b* depicts how the cumulative total magnitude of the whole model changes as training progresses. *c* shows the layerwise weight magnitudes for the 4 versions of CBP, at the end of training. Each bar in plot *c* represents the mean magnitude for some layer for the last 15% of the tasks, averaged over 3 runs. All experiments were run with 3 seeds; a single standard deviation is depicted as error.

ples them from a 0-meaned distribution, hence reducing the total weight size. Conversely, if a layer and its immediate neighbours are not regularised, CBP does not *directly* provide a clear regularisation effect, as no weights for that layer are regularised. Therefore, rather expectedly, we find that **replacing all layers except the first results in a stable weight magnitude increase over time**, mainly driven by the weight magnitude increase in the first layer. However, surprisingly, we show the inverse is also true – **replacing neurons only in the first layer stabilises weight magnitudes in all layers**, which we suspect is the core reason behind the first layer phenomenon.

We depict the results described in the previous paragraph in Figure 6, for the base model. Part *a* of the figure depicts the differences in accuracy for all 4 versions of CBP, again showing that first-layer CBP performs better than 2-L layer CBP. Furthermore, part *b* depicts how the weight magnitudes increase significantly over time for the 2-L version of CBP, while staying relatively constant for the layer 1 version of CBP, which is a significant result. Finally, the reason behind this fact is directly motivated by analysing the *layerwise* weight magnitudes (part *c* of Figure 6) – it can be seen that the first layer's weight magnitudes dominate the weight distribution. Hence, controlling the first layer's magnitudes is key, which is exactly what layer 1 version of CBP does, and the 2-L layer CBP fails to accomplish, as seen in part *c*.

However, we also point out, that while all hidden layers of the base model contain 100 neurons, the first layer's weight matrix contains $784 \cdot 100$ parameters, while the consecutive layers all contain $100 \cdot 100$ parameters, hence the first layer is ≈ 8 times larger. To account for this potential bias, we additionally perform the same experiment with a different MLP, which has the width of 784, hence making all hidden layers the same size ($784 \cdot 784$) and removing the potentially unfair layer size bias. In doing so, we still find the same results, although less extreme, which can be seen in the Figure 7.

5 Discussion

Our results show a direct relation between the depth of replacement in a neural network, and a performance increase. Since we have empirically shown that replacement only in the first layer results in comparable performance to regular CBP, we propose a new, more efficient variant of CBP, which we call *CBP-1*, which differs from CBP in the set of layers that are replaced – *CBP-1* only continually replaces the first layer of a neural network.

An important question still remains: why does replacement in the first layer stabilise weight magnitudes of deeper layers? While this requires deeper investigation, we hypothesise that this effect stems from the randomness in earlier layers effectively trickling down to deeper layers as the earlier ones are replaced. This occurs because the more abstract features learned in deeper layers of a neural network are directly dependent on the concrete features learned earlier in the network. Therefore, the randomness induced by replacing neurons in the first layer may similarly affect later layers, resulting in a regularisation effect. However, this remains a hypothesis requiring further investigation.

Furthermore, while our work mainly agrees with previous literature, certain results could be seen as contradicting other works. For instance, Lewandowski et al. claimed that a decrease in curvature in the loss landscape causes plasticity loss [17], however, we did not find this to be the case, as it did not correlate with plasticity performance (we describe this in Section B.4). Additionally, Berariu et al. [15] found that complete resets of the *last* layers tend to lead to a full regain in plasticity performance, which may be seen as contradicting our core finding that the first layers are optimal for replacement. However, we consider *soft* replacements where we do not replace the full layers from scratch, while also considering significantly longer task horizons compared to the mentioned work hence our findings may be tangential instead of contradictory.

6 Conclusions

When neural networks are trained on continually changing data distributions, they tend to suffer from the *plasticity loss* problem, where a model gradually loses its capacity to learn new information. In our work, we investigated the layerwise dynamics of Continual Backprop (CBP) algorithm [3, 12], which addresses this problem by gradually reinitialising certain weights of a neural network. We first examined which layers are most important for maintaining plasticity. Through experiments on the Continual Permuted MNIST dataset, we

showed two key results:

1. The deeper a layer is in the network, the less plasticity performance is regained by resetting that layer.
2. Resetting only the first hidden layer achieves performance very close to regular Continual Backprop.

The second finding led us to propose a more efficient version of CBP, which we called *CBP-1*: a training algorithm that performs Continual Backpropagation only on the first layer of a neural network. This observation also raised an important question: why does resetting just the first layer produce such strong results?

To investigate this question, we tracked several layerwise metrics over time, including weight norm statistics, dead neuron counts, feature space rank, and neuron utility scores. We found that replacement only in the first layer stabilises the overall *weight norm* of the entire network, while replacing all layers except the first results in fast weight magnitude increase as training progresses. Since gradually increasing weight magnitudes are known to be linked to plasticity loss [20, 21], we concluded that weight magnitude inflation provides a valid explanation for this phenomenon.

Finally, while obtaining results for different model types, we discovered that regular CBP does not reduce plasticity loss on models that use non-ReLU activation functions. This is a new finding, as previous work only evaluated CBP on non-ReLU models with a very simple dataset, which we showed does not scale to larger settings. Our work used a more challenging, Permuted MNIST dataset to reach an opposite conclusion.

7 Future Work

As our compute resources were limited, we were unable to test our hypotheses and validate the results on larger-scale datasets and models. As a result, we had to stick with the Continual Permuted MNIST dataset and a regular MLP model, as that was the most appropriate experimental setting which could still run on CPUs. As a result, a key point for future work is reproducing our results on larger datasets and models, for instance, using a large convolutional neural network, trained on a realistic dataset, such as ImageNet.

Furthermore, even though we found the first layer phenomenon to be directly related to an increase in weight magnitudes, we were unable to figure out why the magnitudes of deeper layers do not increase when training a model using *CBP-1* algorithm. Our current hypothesis is that randomness from early layer replacements propagates to deeper layers, however, this requires theoretical or empirical validation. Hence, future work could look into exactly that – figure out why full-network weight magnitudes remain stable under *CBP-1* algorithm.

Finally, the relationship between activation functions and CBP effectiveness is another direction for future work. Our finding that CBP fails on non-ReLU activations directly shows that CBP acts differently for different activation functions. Therefore, further research could investigate which properties of activation functions make them compatible with CBP to gain further insight into the inner workings of CBP, as well as to find new algorithms that work across all activation types.

8 Responsible Research

In this section, we discuss the ethical considerations and reproducibility concerns of our work and provide a statement on the use of Large Language Models.

8.1 Research Sensitivity

Our work focuses on non-applicative machine learning – it investigates mostly fundamental concepts in continual learning instead of designing machine learning models for real-world scenarios. Given this focus, it was sufficient for us to use simplistic datasets, such as MNIST or the artificial Bit-Flipping dataset, both containing no sensitive or personal information. As a result, our work has few direct ethical concerns in terms of data privacy, data sensitivity or direct misuse of our contributions.

While our work does not raise direct ethical concerns, there exist certain second-order considerations. For instance, continual learning research is related to reinforcement learning (RL), since both fields deal with learning from non-stationary data distributions [1]. And RL is used in many real-world technologies, such as robotics, large language model training, and autonomous vehicles, which have both benefits and risks. Therefore, our contributions could indirectly influence applications with ethical implications, though these effects are outside our control.

Beyond these indirect ethical concerns, we also acknowledge that directing research resources toward continual learning, rather than other areas, is an ethical choice on its own. We justify our focus because continual learning is as an important and even growing building block of artificial intelligence research. And artificial intelligence has been consistently showing potential to solve humanity's most pressing problems, such as medical breakthroughs or poverty reduction. Therefore, our work contributes to a field with significant potential for positive impact.

8.2 Reproducibility and Replicability

We attempt to present our work in a way that allows for full replicability: all code is shared, data is made publicly available, experiments are documented, and optimal hyperparameters are listed. However, despite our work likely being fully replicable, it does have limitations with respect to reproducibility. For instance, following the experimental implementation of Dohare et al. [12], we do not fix random seeds across individual experimental runs, and instead rely on the Central Limit Theorem to ensure that averages over multiple runs result in sufficiently low standard deviations for our results. And although our experiments do demonstrate small standard deviations across runs, indicating valid findings, this approach leaves space for variations in results across different runs. Therefore, we cannot claim that our work is completely reproducible.

8.3 Use of Large Language Models

We used Large Language Models (LLMs) for the following tasks:

- **Code Development**: We used Github Copilot's auto-complete functionality for speeding up programming tasks.
- **Data Visualisations**: We used ChatGPT and DeepSeek to generate skeleton codes for data visualisation scripts and LaTeX figures. After initial generation, the generated codes were always further modified manually to match our exact requirements.
- **Writing Assistance**: Claude AI was used to provide suggestions for text rephrasing and style improvements during the paper writing process.

We provide prompts of our interactions with chatbots in Appendix D.

References

[1] Liyuan Wang, Xingxing Zhang, Hang Su, and Jun Zhu. A comprehensive survey of continual learning: Theory, method and application. *IEEE Trans. Pattern Anal. Mach. Intell.*, 46(8):5362–5383, August 2024.

[2] Michael McCloskey and Neal J Cohen. Catastrophic interference in connectionist networks: The sequential learning problem. In *Psychology of Learning and Motivation*, The psychology of learning and motivation, pages 109–165. Elsevier, 1989.

[3] Shibhansh Dohare, J Fernando Hernandez-Garcia, Qingfeng Lan, Parash Rahman, A Rupam Mahmood, and Richard S Sutton. Loss of plasticity in deep continual learning. *Nature*, 632(8026):768–774, August 2024.

[4] Zaheer Abbas, Rosie Zhao, Joseph Modayil, Adam White, and Marlos C. Machado. Loss of plasticity in continual deep reinforcement learning. In Sarath Chandar, Razvan Pascanu, Hanie Sedghi, and Doina Precup, editors, *Proceedings of The 2nd Conference on Lifelong Learning Agents*, volume 232 of *Proceedings of Machine Learning Research*, pages 620–636. PMLR, 22–25 Aug 2023.

[5] Boyi Liu, Lujia Wang, and Ming Liu. Lifelong federated reinforcement learning: A learning architecture for navigation in cloud robotic systems. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, November 2019.

[6] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, Demis Hassabis, Claudia Clopath, Dharshan Kumaran, and Raia Hadsell. Overcoming catastrophic forgetting in neural networks. *arXiv [cs.LG]*, December 2016.

[7] David Lopez-Paz and Marc' Aurelio Ranzato. Gradient episodic memory for continual learning. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[8] Clare Lyle, Zeyu Zheng, Evgenii Nikishin, Bernardo Avila Pires, Razvan Pascanu, and Will Dabney. Understanding plasticity in neural networks. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 23190–23211. PMLR, 23–29 Jul 2023.

[9] Alex Lewandowski, Haruto Tanaka, Dale Schuurmans, and Marlos C Machado. Directions of curvature as an explanation for loss of plasticity. *arXiv [cs.LG]*, November 2023.

[10] Ghada Sokar, Rishabh Agarwal, Pablo Samuel Castro, and Utku Evci. The dormant neuron phenomenon in deep reinforcement learning. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett, editors, *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 32145–32168. PMLR, 23–29 Jul 2023.

[11] Timo Klein, Lukas Miklautz, Kevin Sidak, Claudia Plant, and Sebastian Tschiatschek. Plasticity loss in deep reinforcement learning: A survey. *arXiv [cs.AI]*, November 2024.

[12] Shibhansh Dohare, Richard S. Sutton, and A. Rupam Mahmood. Continual backprop: Stochastic gradient descent with persistent randomness, 2022.

[13] Dongwan Kim and Bohyung Han. On the stability-plasticity dilemma of class-incremental learning. In *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, June 2023.

[14] Vladimir Araujo, Julio Hurtado, Alvaro Soto, and Marie-Francine Moens. Entropy-based stability-plasticity for lifelong learning. In *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. IEEE, June 2022.

[15] Tudor Berariu, Wojciech Czarnecki, Soham De, Jörg Bornschein, Samuel L. Smith, Razvan Pascanu, and Claudia Clopath. A study on the plasticity of neural networks. *CoRR*, abs/2106.00042, 2021.

[16] Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *Computer Vision – ECCV 2014*, Lecture notes in computer science, pages 818–833. Springer International Publishing, Cham, 2014.

[17] Alex Lewandowski, Haruto Tanaka, Dale Schuurmans, and Marlos C. Machado. Directions of curvature as an explanation for loss of plasticity, 2024.

[18] Delft High Performance Computing Centre (DHPC). DelftBlue Supercomputer (Phase 2). https://www.tudelft.nl/dhpc/ark:/44463/DelftBluePhase2, 2024.

[19] Delft AI Cluster (DAIC). The delft ai cluster (daic), rrid:scr$_0$25091, 2024.

[20] Mohamed Elsayed, Qingfeng Lan, Clare Lyle, and A. Rupam Mahmood. Weight clipping for deep continual and reinforcement learning. *Reinforcement Learning Journal*, 5:2198–2217, 2024.

[21] Clare Lyle, Zeyu Zheng, Khimya Khetarpal, Hado van Hasselt, Razvan Pascanu, James Martens, and Will Dabney. Disentangling the causes of plasticity loss in neural networks, 2024.

A Bit-Flipping Problem

This section describes our effort to perform a layerwise investigation on the Bit-Flipping problem, a toy experimental setting introduced by Dohare et al. [12, 12]. Since the original experiment uses only a single-hidden-layer MLP (Multi Layer Perceptron), it does not allow performing layerwise analysis. Therefore, adapting this setting for deeper architectures while maintaining the core results shown by Dohare et al. [12, 12] was necessary. Unfortunately, the setting did not generalise to deeper models. The following subsections first outline the experimental framework, then present the issues that prevented the scaling to deeper networks.

A.1 Experimental Setting

Dataset. The Bit-Flipping dataset consists of a series of consecutive tasks, each task being a small dataset. A single task consisting of T samples, where each sample is a pair (x_t, y_t), such that $x_t \in \{0,1\}^m$, the first f bits $(x_t)_{1:f}$ are constant for the same task, and the last $m-f$ bits $(x_t)_{f:m}$ are sampled from $U\{0,1\}$ for every sample. Therefore, every task is uniquely defined by the f bits (called *flipping bits*) that all its samples have as the first f bits. The original work by Dohare et al. [12] have $m = 20$, $f = 15$ and $T = 10^4$. Additionally, for all samples, we have $y_t = F(x_t)$, where F is a neural network that is more complex than the one being trained on the described dataset. In particular, we follow the original work and set the target network to be a randomly initialised single hidden layer MLP (Multi Layer Perceptron), with the LTU (Linear Threshold Unit) activation function. The size and complexity of the target network F is discussed in the later paragraphs. The described dataset induces a regression problem, therefore, the loss being used for training the learner is regular MSE loss.

Making the model deeper. The original learner and target networks proposed by Dohare et al [12], are both single hidden layer MLPs. However, a single-layer learner network is suboptimal for investigating layerwise dynamics, as there are not enough layers for investigation. Therefore, we opt to make the learner network deeper, in particular, at least 4 hidden layers deep. However, making the model deeper was proven to be non-trivial, as even basic results do not reproduce under slight deviations. In particular, for a meaningful analysis, we aimed to identify a model–dataset configuration that satisfies the following **four properties**:

1. The learner model is at least 4 hidden layers deep – a sufficiently deep model is needed to investigate layerwise dynamics.
2. The error rates increase over time under regular Backpropagation, indicating the dataset is complex enough to see a plasticity decline.
3. The error rates under CBP remain relatively stable. If this condition was not met, it would suggest that CBP is ineffective in the given setting, thus making further analysis uninformative.
4. Both standard backpropagation and the L_2 baseline perform worse than the proposed variant of CBP. Similar to the previous point, analysing CBP in a setting where it is underperforming would not yield useful insights.

However, we were unable to find an experimental setting where all 4 of these settings were met, potentially indicating that the original dataset proposed by Dohare et al. was too specific for small and shallow models.

A.2 Inconsistencies Within the Original Work

In attempting to scale up the experimental setting to work on a deeper neural network, we found two large inconsistencies between what was reported in the core paper we base our work on [3], and the respective reported codebase.

Differing Utility Score Definitions

The publication that provides the codebase we base our work on [3], defines contribution utility scores as shown in Equation 1. However, most experiments from the reported codebase, use a different utility scoring equation, which in the code is called *adaptable utility*. This equation likely comes from an earlier work by Dohare et al., i.e., the Continual Backprop paper [12], where they define utility as $\hat{\mathbf{u}}$:

$$y_{l,i} = \frac{\left|h_{l,i} - \hat{f}_{l,i}\right| \times \sum_{k=1}^{n_{l+1}} |w_{l,i,k}|}{\sum_{j=1}^{n_{l-1}} |w_{l-1,j,i}|} \tag{3}$$

$$\mathbf{u}_l[i] = \eta \times \mathbf{u}_l[i] + (1-\eta) \times y_{l,i} \tag{4}$$

$$\hat{\mathbf{u}}_l[i] = \frac{\mathbf{u}_l[i]}{1-\eta^{a_{l,i}}}. \tag{5}$$

With $\hat{f}_{l,i}$ denoting the running average of hidden activations for neuron i at layer l; and $a_{l,i}$ denoting the time since last replacement of neuron i at layer l.

Through initial experimentation with both of these utility score definitions, we did not find considerable differences in performance between them. As a result, it was decided to use the definition from Equation 1 since it matches the claim in the core publication we base our work on [3].

Flipping One Bit vs. Flipping All Bits

As we describe in Section A.1, for any two consecutive tasks in the Bit-Flipping dataset, their *flipping bits* differ by exactly one bit. However, the codebase allows for another option – one where the flipping bits are completely resampled from $U\{0,1\}$ for every task, thus allowing any number of bit flips for consecutive tasks. We refer to the original dataset version as the *flip-one* dataset, and to the other dataset as *flip-all*. Strangely, most of the experimental setups in the codebase use the *flip-all* version instead of the *flip-one*, described in the publication.

To confirm which version of the dataset was actually used in the original paper, we ran the respective experiments of the Bit-Flipping problem with both *flip-one* and *flip-all* datasets, using regular backpropagation. Each experiment is run on a ReLU-based model with 14 random seeds. As no plasticity-preserving technique is used, we expect to observe errors

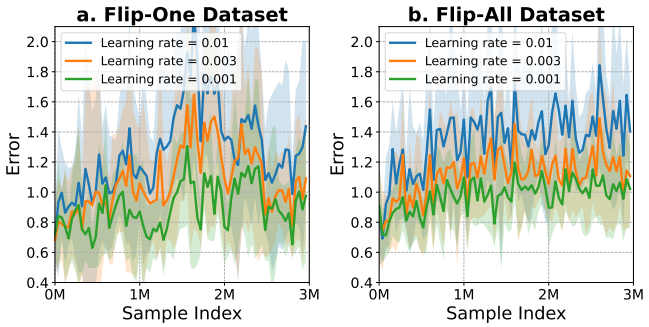

Figure 8: Performance of a network trained using **regular backpropagation** on two versions of the Bit-Flipping dataset. Lower is better. The curves are made up of bins of size 40.000. Shaded areas denote a single standard deviation over 14 runs.

steadily increase, as this was shown to happen in the original work for the Bit-Flipping dataset [3, 12]. However, it can be seen from Figure 8 that only one of the two datasets induced stable plasticity loss – the *flip-all* version of the Bit-Flipping dataset. This suggests that the *flip-all* version was used to obtain results in the original work [3], contrary to what the publication claims. As a result, it was decided to use the *flip-all* version for further experiments.

A.3 Problems With Scaling Up the Model

The original Bit-Flipping experiment is based on a model with 1 hidden layer. However, to investigate layerwise dynamics, it is first crucial to obtain a working setup which reproduces the results from the original publications [3, 12] with a deeper model. We aim for a model which is at least 4 hidden layers deep. However, after initial experimentation, we found that trivially changing the learner model's depth results in the dataset being so simple for the model that even regular backpropagation does not lose plasticity.

As a result, we performed a hyperparameter search to find a model-dataset setting which would satisfy the 4 conditions outlined in Section A.1. Throughout this process, we tuned the following hyperparameters:

- Learner model depths.
- Target model sizes.
- Activation types.
- Learning rates.
- Training algorithms (L_2, Backprop and CBP).
- Weight decay values (L_2) and replacement rates (CBP).
- Task sizes T.
- Total number of samples.

We do not describe the full process of tuning these parameters in order to obtain the 4 desirable conditions. However, we show the results of the last iteration of the hyperparameter search, after which it was decided to stop working on this dataset. In this step, we used a neural network with 5 hidden layers, and had already found it to be beneficial to set $T = 2500$, while using the *flip-all* version of the Bit-Flipping dataset, the regular contribution utility score equation (see Equation 1) and a 500 neuron wide target network. The goal of this iteration was to find the best hyperparameters for ReLU, Sigmoid and Tanh activation functions. So we swept over the following:

1. **Activation functions**. Considered ReLU, Sigmoid and Tanh.
2. **Training algorithms**. Considered CBP, BP and L_2.
3. **Learning rates**. For all algorithms, considered 0.005, 0.001, 0.0005, 0.0001.
4. **Weight decay / Replacement rate**. For L_2 we considered weight lambdas of 10^{-4}, 10^{-3}, 10^{-2}, and for CBP we considered replacement rates of 10^{-4}, 10^{-3}, 10^{-2}.
5. **Seeds**. Every experimental setting was repeated 10 times.

We effectively took the cross product of the 5 items above, hence resulting in over 800 experimental runs.

After obtaining the results, we chose the optimal hyperparameters for every activation function and training algorithm pair, which are depicted in Figure 9. The figure directly shows that in no single configuration all 4 necessary conditions are fulfilled, as for instance, for ReLU, regular backpropagation does not result in increasing errors, and for Sigmoid and Tanh activations, CBP fails to outperform L_2.

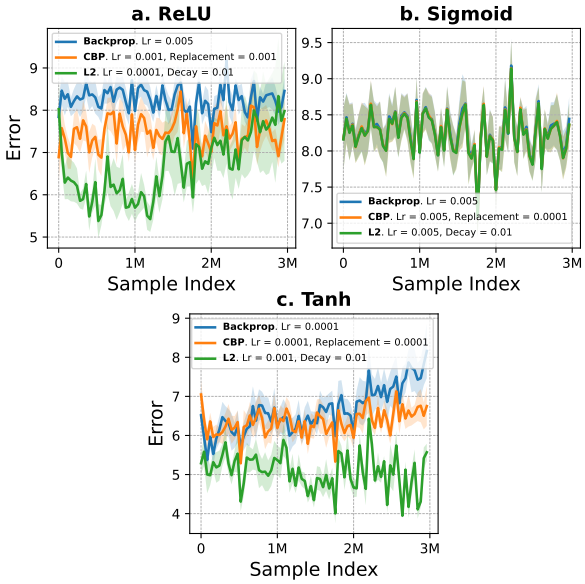

Figure 9: Optimal training algorithms for every activation function. Each line is chosen to be the best among variations of the same algorithm with different hyperparameters. The shaded areas depict a single standard deviation across 10 runs. Lower is better.

B Explaining First Layer Phenomenon

There are multiple lenses that could potentially explain the first-layer phenomenon. In this section, we present some approaches that failed. All figures in this section correspond to

training the base model (see Section 4.1) with CBP on Continual Permuted MNIST for 400 tasks, while tracking a metric discussed in the respective subsection. The layerwise metrics from the figures are computed by averaging values over the particular layer and over the final 15% of tasks.

B.1 Dead Neurons

We initially hypothesised that the reason behind the first layer phenomenon is that replacement in the first layer disproportionately decreases the number of dead neurons throughout the **whole** model. However, we found that to be false. Exclusive first-layer replacement results in a large number of dead neurons. This is shown in Figure 10. Furthermore, the figure shows that replacing layers 2-L nearly solves the dead neuron issue, and this outcome is incompatible with our core performance result. It is incompatible becaus,e according to the dead neuron metric, the 2-L CBP version should perform very well, however we have shown that the exact opposite happens (see Figure 6a).

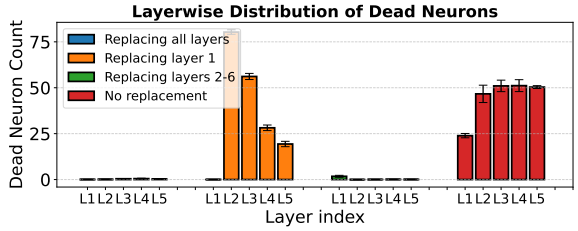

Figure 10: Distribution of dead neurons within the network for different versions of CBP. Lower is better. Error bars correspond to a single standard deviation across 3 runs.

B.2 Feature Ranks

Another hypothesis was that the randomness induced by first-layer replacement results in more useful features for further layers, as measured by the effective rank of the feature maps. That was found to be false. Exclusive first-layer replacement version of CBP results in lower feature ranks compared to other methods which we know to perform worse. This result is shown in Figure 11.

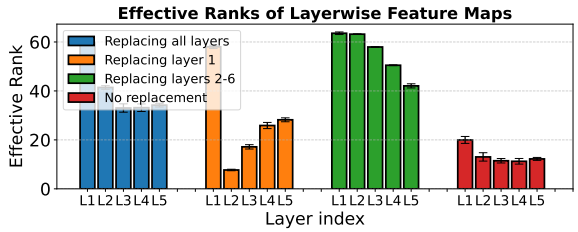

Figure 11: Layerwise feature ranks for different versions of CBP. Higher is better. Error bars correspond to a single standard deviation across 3 runs.

B.3 Feature Usefulness

After disproving the previous hypothesis, it was decided to quantify the usefulness of features using a more sophisticated metric than feature map ranks. Therefore, we attempted to quantify the usefulness of features using the layer probing metric described in Section 3.2. The results, shown in Figure 12, proved to be very similar to those of feature ranks, again suggesting that the first layer phenomenon is not directly related to intermediate network activations.

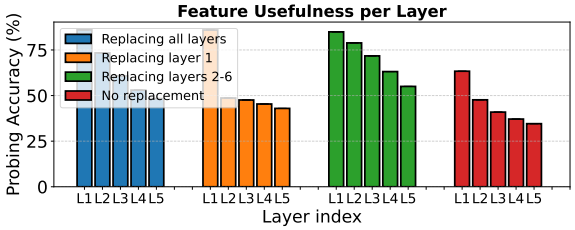

Figure 12: Layerwise probing accuracies for different versions of CBP. The exact way probing accuracies are calculated is described in Section 3.2. Higher is better. Error bars correspond to a single standard deviation across 3 runs.

B.4 Curvature of the Loss Landscape

Lewandowski et al. [9] suggested that plasticity loss is caused by a decrease in curvature of the loss space. Since the curvature can be measured by approximating the rank of the Hessian of a neural network [9], we tracked this metric to investigate if it could be causing plasticity decline. However, we found this to not correlate with actual performance, as the first-layer replacement version of CBP resulted in relatively high loss of curvature, even though it achieves very high performance. We depict the layerwise Hessian ranks for CBP variants in Figure 13.

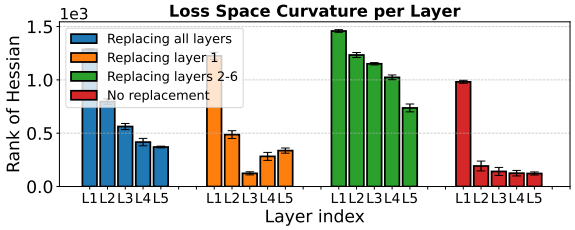

Figure 13: Approximate ranks of the Hessian of a neural network for different versions of CBP. Higher is better. Error bars correspond to a single standard deviation across 3 runs.

C Hyperparameter Search

See Table 4 for all hyperparameters tried and Table 3 for the best ones for every model.

D Use of Large Language Models

In this section, we provide examples of our chatbot usage for code generation and text rewriting.

D.1 Text Rephrasing and Style Improvements

We used Claude AI for text rephrasing suggestions. Here we do not paste the full prompts, since most of them include entire paragraphs or subsections with the goal of informing Claude of the broader context. Therefore, we trim most prompts, indicating that with "..." symbols, for the purposes of readability.

Prompt 1

```
Smooth this out

To investigate the layerwise effects of CBP,
we ...
```

Prompt 2

```
Rephrase this text to make it smooth. Keep the
same style!

\section{Responsible Research}
In this section, we discuss the ethical
considerations, reproducibility concerns,
and transparency issues of ...
```

Prompt 3

```
nono, you are not keeping the same style. You
are changing some words for no reason. For
instance, I said "focuses" you say "centers
on"...
no need to change this. So KEEP MY STYLE
do not change words which do not need to be
changed.
```

Prompt 4

```
Rephrase to smooth:

In this section, we describe our attempts to
scale up...
```

Prompt 5

```
This is a conclusion of my paper. Please write
it in my style, just phrase it more smoothly.
But plz, dont add too fancy words and stuff

When neural networks are trained on
continually changing data...
```

We used this same format for multiple other section or paragraph rewrites – basically asking Claude to *smooth out* some text, while keeping the style the same. If Claude failed to keep the style, we asked it again to do so. After obtaining rewritten text, it was always edited to match our standards and style.

D.2 Generating Scripts for Figures

We use Python's *matplotlib* library for generating figures in this paper, and LaTeX for writing the paper itself. Therefore, we employed chatbots for generating entire Python scripts (as well as parts of them) for rendering basic plots. After initial generation, we would edit those scripts manually to match the exact needs of the paper. Additionally, we used Large Language Models for LaTeX code generation, inputting certain LaTeX figure code and asking the model to transform it to some other LaTeX figure. We now give representative examples of different types of code generation prompts.

Prompt 1. Generating a Python script for figure plotting.

```
I want to do ax.bar plot, where I plot
multiple bar sequences on the same plot, by
stacking the plots next to each other. I.e.,
say for x=2 value, I want to have n bars next
to each other with different colors. same for
x=3, etc
```

Prompt 2. Generating LaTeX code for a figure.

```
Take this:

\begin{block}{Experiment: Continual Permuted
MNIST}
\begin{figure}[htbp]
      \centering
      \includegraphics[width=0.95\textwidth]
    ...  skipped multiple code lines
\end{block}

and turn into a two-part figure, where
these two figures are next to each other
horizontally
```

Prompt 3. Generating partial Python code for figure plotting.

```
sometimes matplotlib shows the y labels in
some simlified format, like 1.0, 2.0, ... with
a note that everything in the scale of, say,
1e7 next to the axis. How do I do this for
1e4?
```

Table 3: The found optimal hyperparemeters for each experimental setting. Every row indicates a version of the model, corresponding to an experimental setting. The columns of that row show the best hyperparameter set for that particular training algorithm.

Setting	BP	CBP	L_2
Varying Widths			
Width 20	Lr = $5 \cdot 10^{-4}$	Lr = $1 \cdot 10^{-3}$, Repl = $1 \cdot 10^{-3}$	Lr = $1 \cdot 10^{-3}$, Decay = $1 \cdot 10^{-2}$
Width 50	Lr = $5 \cdot 10^{-4}$	Lr = $5 \cdot 10^{-3}$, Repl = $1 \cdot 10^{-4}$	Lr = $1 \cdot 10^{-3}$, Decay = $1 \cdot 10^{-2}$
Width 100	Lr = $5 \cdot 10^{-4}$	Lr = $5 \cdot 10^{-3}$, Repl = $1 \cdot 10^{-4}$	Lr = $5 \cdot 10^{-3}$, Decay = $1 \cdot 10^{-3}$
Width 150	Lr = $5 \cdot 10^{-4}$	Lr = $5 \cdot 10^{-3}$, Repl = $1 \cdot 10^{-4}$	Lr = $1 \cdot 10^{-3}$, Decay = $1 \cdot 10^{-3}$
Width 200	Lr = $1 \cdot 10^{-3}$	Lr = $5 \cdot 10^{-3}$, Repl = $1 \cdot 10^{-4}$	Lr = $1 \cdot 10^{-3}$, Decay = $1 \cdot 10^{-3}$
Varying Activation Functions			
ReLU	Lr = $5 \cdot 10^{-4}$	Lr = $5 \cdot 10^{-3}$, Repl = $1 \cdot 10^{-4}$	Lr = $5 \cdot 10^{-3}$, Decay = $1 \cdot 10^{-3}$
SeLU	Lr = $5 \cdot 10^{-4}$	Lr = $5 \cdot 10^{-4}$, Repl = $1 \cdot 10^{-4}$	Lr = $1 \cdot 10^{-3}$, Decay = $1 \cdot 10^{-2}$
Tanh	Lr = $5 \cdot 10^{-4}$	Lr = $5 \cdot 10^{-4}$, Repl = $1 \cdot 10^{-6}$	Lr = $5 \cdot 10^{-3}$, Decay = $1 \cdot 10^{-2}$
Varying Depths			
2 Layers	Lr = $1 \cdot 10^{-3}$	Lr = $5 \cdot 10^{-3}$, Repl = $1 \cdot 10^{-4}$	Lr = $5 \cdot 10^{-3}$, Decay = $1 \cdot 10^{-3}$
5 Layers	Lr = $5 \cdot 10^{-4}$	Lr = $5 \cdot 10^{-3}$, Repl = $1 \cdot 10^{-4}$	Lr = $5 \cdot 10^{-3}$, Decay = $1 \cdot 10^{-3}$
8 Layers	Lr = $5 \cdot 10^{-4}$	Lr = $1 \cdot 10^{-3}$, Repl = $1 \cdot 10^{-4}$	Lr = $1 \cdot 10^{-3}$, Decay = $1 \cdot 10^{-3}$
Varying Depth, Same Number of Parameters			
2 Layers	Lr = $1 \cdot 10^{-3}$	Lr = $5 \cdot 10^{-3}$, Repl = $1 \cdot 10^{-5}$	Lr = $5 \cdot 10^{-3}$, Decay = $1 \cdot 10^{-3}$
5 Layers	Lr = $5 \cdot 10^{-4}$	Lr = $5 \cdot 10^{-3}$, Repl = $1 \cdot 10^{-4}$	Lr = $5 \cdot 10^{-3}$, Decay = $1 \cdot 10^{-3}$
8 Layers	Lr = $5 \cdot 10^{-4}$	Lr = $1 \cdot 10^{-3}$, Repl = $1 \cdot 10^{-4}$	Lr = $1 \cdot 10^{-3}$, Decay = $1 \cdot 10^{-3}$

Table 4: The list of all hyperparameters tried for layerwise experiments. In every cell, we investigate all combinations of the provided hyperparameters.

Experiment Setting	CBP	BP	L_2
Varying Widths			
Width = 20	Lr: $5 \cdot 10^{-4}, 10^{-3}, 5 \cdot 10^{-3}, 10^{-2}$ Repl: $10^{-4}, 10^{-3}, 10^{-5}$	Lr: $5 \cdot 10^{-4}, 10^{-3}, 5 \cdot 10^{-3}, 10^{-2}$	Lr: $5 \cdot 10^{-4}, 10^{-3}, 5 \cdot 10^{-3}, 10^{-2}$ Decay: $10^{-4}, 10^{-3}, 10^{-2}$
Width = 50	Lr: $5 \cdot 10^{-4}, 10^{-3}, 5 \cdot 10^{-3}, 10^{-2}$ Repl: $10^{-4}, 10^{-3}, 10^{-5}$	Lr: $5 \cdot 10^{-4}, 10^{-3}, 5 \cdot 10^{-3}, 10^{-2}$	Lr: $5 \cdot 10^{-4}, 10^{-3}, 5 \cdot 10^{-3}, 10^{-2}$ Decay: $10^{-4}, 10^{-3}, 10^{-2}$
Width = 100	Lr: $5 \cdot 10^{-4}, 10^{-3}, 5 \cdot 10^{-3}, 10^{-2}$ Repl: $10^{-4}, 10^{-3}, 10^{-5}$	Lr: $5 \cdot 10^{-4}, 10^{-3}, 5 \cdot 10^{-3}, 10^{-2}$	Lr: $5 \cdot 10^{-4}, 10^{-3}, 5 \cdot 10^{-3}, 10^{-2}$ Decay: $10^{-4}, 10^{-3}, 10^{-2}$
Width = 150	Lr: $5 \cdot 10^{-4}, 10^{-3}, 5 \cdot 10^{-3}, 10^{-2}$ Repl: $10^{-4}, 10^{-3}, 10^{-5}$	Lr: $5 \cdot 10^{-4}, 10^{-3}, 5 \cdot 10^{-3}, 10^{-2}$	Lr: $5 \cdot 10^{-4}, 10^{-3}, 5 \cdot 10^{-3}, 10^{-2}$ Decay: $10^{-4}, 10^{-3}, 10^{-2}$
Width = 200	Lr: $5 \cdot 10^{-4}, 10^{-3}, 5 \cdot 10^{-3}, 10^{-2}$ Repl: $10^{-4}, 10^{-3}, 10^{-5}$	Lr: $5 \cdot 10^{-4}, 10^{-3}, 5 \cdot 10^{-3}, 10^{-2}$	Lr: $5 \cdot 10^{-4}, 10^{-3}, 5 \cdot 10^{-3}, 10^{-2}$ Decay: $10^{-4}, 10^{-3}, 10^{-2}$
Varying Activation Functions			
ReLU	Lr: $5 \cdot 10^{-4}, 10^{-3}, 5 \cdot 10^{-3}, 10^{-2}$ Repl: $10^{-4}, 10^{-3}, 10^{-5}$	Lr: $5 \cdot 10^{-4}, 10^{-3}, 5 \cdot 10^{-3}, 10^{-2}$	Lr: $5 \cdot 10^{-4}, 10^{-3}, 5 \cdot 10^{-3}, 10^{-2}$ Decay: $10^{-4}, 10^{-3}, 10^{-2}$
SeLU	Lr: $5 \cdot 10^{-4}, 10^{-3}, 5 \cdot 10^{-3}, 10^{-2}$ Repl: $10^{-4}, 10^{-3}, 10^{-5}, 10^{-6}$	Lr: $5 \cdot 10^{-4}, 10^{-3}, 5 \cdot 10^{-3}, 10^{-2}$	Lr: $5 \cdot 10^{-4}, 10^{-3}, 5 \cdot 10^{-3}, 10^{-2}$ Decay: $10^{-4}, 10^{-3}, 10^{-2}$
Tanh	Lr: $5 \cdot 10^{-4}, 10^{-3}, 5 \cdot 10^{-3}, 10^{-2}$ Repl: $10^{-4}, 5 \cdot 10^{-4}, 10^{-3}, 5 \cdot 10^{-3}, 10^{-5}, 10^{-6}, 5 \cdot 10^{-5}$	Lr: $5 \cdot 10^{-4}, 10^{-3}, 5 \cdot 10^{-3}, 10^{-2}$	Lr: $5 \cdot 10^{-4}, 10^{-3}, 5 \cdot 10^{-3}, 10^{-2}$ Decay: $10^{-4}, 10^{-3}, 10^{-2}$
Varying Depths			
2 Layers	Lr: $5 \cdot 10^{-4}, 10^{-3}, 5 \cdot 10^{-3}, 10^{-2}$ Repl: $10^{-4}, 10^{-3}, 10^{-5}$	Lr: $5 \cdot 10^{-4}, 10^{-3}, 5 \cdot 10^{-3}, 10^{-2}$	Lr: $5 \cdot 10^{-4}, 10^{-3}, 5 \cdot 10^{-3}, 10^{-2}$ Decay: $10^{-4}, 10^{-3}, 10^{-2}$
5 Layers	Lr: $5 \cdot 10^{-4}, 10^{-3}, 5 \cdot 10^{-3}, 10^{-2}$ Repl: $10^{-4}, 10^{-3}, 10^{-5}$	Lr: $5 \cdot 10^{-4}, 10^{-3}, 5 \cdot 10^{-3}, 10^{-2}$	Lr: $5 \cdot 10^{-4}, 10^{-3}, 5 \cdot 10^{-3}, 10^{-2}$ Decay: $10^{-4}, 10^{-3}, 10^{-2}$
8 Layers	Lr: $5 \cdot 10^{-4}, 10^{-3}, 5 \cdot 10^{-3}, 10^{-2}$ Repl: $10^{-4}, 10^{-3}, 10^{-5}$	Lr: $5 \cdot 10^{-4}, 10^{-3}, 5 \cdot 10^{-3}, 10^{-2}$	Lr: $5 \cdot 10^{-4}, 10^{-3}, 5 \cdot 10^{-3}, 10^{-2}$ Decay: $10^{-4}, 10^{-3}, 10^{-2}$
Varying Depth, Same Number of Parameters			
2 Layers	Lr: $5 \cdot 10^{-4}, 10^{-3}, 5 \cdot 10^{-3}, 10^{-2}$ Repl: $10^{-4}, 10^{-3}, 10^{-5}$	Lr: $5 \cdot 10^{-4}, 10^{-3}, 5 \cdot 10^{-3}, 10^{-2}$	Lr: $5 \cdot 10^{-4}, 10^{-3}, 5 \cdot 10^{-3}, 10^{-2}$ Decay: $10^{-4}, 10^{-3}, 10^{-2}$
5 Layers	Lr: $5 \cdot 10^{-4}, 10^{-3}, 5 \cdot 10^{-3}, 10^{-2}$ Repl: $10^{-4}, 10^{-3}, 10^{-5}$	Lr: $5 \cdot 10^{-4}, 10^{-3}, 5 \cdot 10^{-3}, 10^{-2}$	Lr: $5 \cdot 10^{-4}, 10^{-3}, 5 \cdot 10^{-3}, 10^{-2}$ Decay: $10^{-4}, 10^{-3}, 10^{-2}$
8 Layers	Lr: $5 \cdot 10^{-4}, 10^{-3}, 5 \cdot 10^{-3}, 10^{-2}$ Repl: $10^{-4}, 10^{-3}, 10^{-5}$	Lr: $5 \cdot 10^{-4}, 10^{-3}, 5 \cdot 10^{-3}, 10^{-2}$	Lr: $5 \cdot 10^{-4}, 10^{-3}, 5 \cdot 10^{-3}, 10^{-2}$ Decay: $10^{-4}, 10^{-3}, 10^{-2}$