Maintaining Plasticity for Deep Continual Learning
Activation Function-Adapted Parameter Resetting Approaches

Victor Purice[1]

Supervisor(s): Dr. 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: Victor Purice
Final project course: CSE3000 Research Project
Thesis committee: Dr. Wendelin Böhmer, Dr. Megha Khosla, Laurens Engwegen

Abstract

Standard deep learning utensils, in particular feed-forward artificial neural networks and the back-propagation algorithm, fail to adapt to sequential learning scenarios, where the model is continuously presented with new training data. Many algorithms that aim to solve this problem exist, but their performance is heavily influenced by factors such as the properties of the environment, the non-stationarity of the input/output data, and the intrinsic characteristics of the utilised models. In this paper, we design an activation function-adapted framework for reinitializing neurons in continual learning, which aims to preserve the network's ability to learn and adjust to new data. A novel utility measure is introduced, which estimates the activation value of each neuron. The proposed strategy selectively reinitializes neurons exhibiting the lowest and highest activation values, which are typically detrimental to the learning performance, particularly in continual learning contexts. We evaluate the proposed framework across different scenarios using various activation functions and show that simple strategies—when well-matched to the model's activation function—can effectively mitigate plasticity loss in simple supervised learning tasks.

1 Introduction

The field of Continual Learning (CL) focuses on developing practical machine learning approaches for sequentially training machine learning models on new tasks without catastrophically forgetting the preceding task data [1]. Connectionist networks face the problem of gradually forgetting old information when learning from new data [2]. This phenomenon has been defined as the loss of plasticity in Continual Learning. Formally, CL strategies aim to balance the trade-off between plasticity — the ability to learn from new data — and stability — the retention of past information.

There does not exist a solitary definition of plasticity which encapsulates all the underlying aspects of this phenomenon [3]. For example Lyle et al. [4] refers to plasticity as the quality of a particular point in parameter space to serve as a starting point for objective optimization, and Berariu et al. [5] refers to plasticity as "the difference in performance between a pretrained model — e.g. one that has learned a few tasks already — versus a freshly initialized one".

Plasticity loss in deep learning can stem from a plethora of factors operating on different conceptual levels. For example, when analyzing the network properties of the model, factors such as the loss landscape sharpness [6], shifts in the effective rank dynamics [7], issues like vanishing gradients [8], and a high number of dormant/dead neurons [9] can all exacerbate the plasticity loss phenomenon. A perpetrator present in almost all loss of plasticity occurrences is the non-stationarity (change of distribution) of the input data [4]. This aspect closely links CL to the field of deep Reinforcement Learning (RL). In deep RL, distribution shifts occur naturally – even without changing tasks – for example, through changes in state visitation distribution due to an updated policy or changes in an agent's target distribution due to an improved value function in temporal difference learning [3], [10]. Hence, understanding the mechanism behind the plasticity loss phenomenon and mitigating this loss is crucial if we wish to develop deep RL agents which can rise to the challenge of complex and constantly changing environments [6].

Multiple strategies exist which mitigate the plasticity loss in CL. These strategies can be grouped into the following categories [3]:

- **Non-Targeted Weight Resets**: strategies where the weights of some parts of the network are either reset or re-sampled from the initial weight distribution. Popular approaches include *Shrink and Perturb* [11], which performs weight regularization and injects noise into the network.
- **Targeted Weight Resets**: strategies that extend on the previous category, which selectively reinitialize the neurons of a network. Existing approaches include *Continual Backpropagation* [12], which resets neurons based on a heuristic measure of neuron utility.
- **Parameter Regularisation**: strategies that regularise the parameters' norms. Common approaches include ensuring small weights magnitudes through frameworks like L2 regularisation [13].
- **Feature Rank Regularisation**: strategies that discourage feature rank collapse, a metric correlated with high performance in CL. Common approaches include *Direct Singular Value Regularisation* [7], which introduces a loss function that aims to maintain a high feature rank in the underlying network.
- **Custom Activation Functions**: activation functions specifically designed for maintaining plasticity in CL. Existing approaches include Parameterised Exponential Linear Units (PeLU), which introduces learnable parameters that allow the network to respond to distribution shifts in the input data [14].

There exist many strategies that are able to mitigate plasticity loss in CL, but the area of research that explores learning strategy adaptations to the underlying activation function of the network remains underexplored. It is thus worth investigating whether the activation function has enough intrinsic characteristics that can be leveraged for successfully solving the issue of plasticity loss in CL.

In this paper, we present an approach aimed at harmoniously matching a parameter reset algorithm to the activation function of the underlying prediction network, with the main goal of maintaining plasticity in deep CL. A new heuristic utility score measure is introduced, similar to that of the *Continual Backpropagation* model, which assesses the 'activity' of a particular neuron during the training process of the network. We explore the effect of resetting both low and high-utility score neurons during the backpropagation process. This initiative's motivation is that the neurons that reach

these two groups typically adversely affect the mitigation of plasticity loss in CL, because low utility score neurons 'struggle' during the training process, and high utility score neurons 'dominate' it.

The goal of this research is to expand the knowledge on how algorithms that are well-matched to the model's activation function can effectively mitigate plasticity loss in simple supervised learning tasks. The resulting designed framework is capable of mitigating plasticity loss in the proposed experimental setup and outperforms some existing state-of-the-art approaches when applied to networks that use various activation functions on their hidden layers.

In Section 2, the *Continual Backpropagation* algorithm is presented, serving as an inspiration for the framework proposed in the same section. In Section 3, we introduce the experimental setup and in Section 4 present the performance of the proposed framework across different metrics, such as prediction accuracy, model running time, gradient covariance matrix and effective parameter rank. We draw the conclusions on our research in Section 5 and reflect upon the responsible aspects of it in Section 6.

2 Methodology

This section serves as the foundation of the proposed method for maintaining plasticity in deep CL. It presents the Continual Backpropagation algorithm and its limitations, then the theoretical considerations supporting our decision choices and formalises the proposed algorithm.

Preliminaries

The recently proposed Continual Backpropagation algorithm aims to increase the plasticity while also maintaining high stability performance in CL. The algorithm selectively reinitializes neurons in the network. This reinitialization adds a source of variability to the model, restricting this variability to those units that are less frequently used, thereby preserving the past knowledge of the more dominant neurons in the network.

The heuristic utility measure for every neuron is defined as:

$$u_l[i] \leftarrow \eta \cdot u_l[i] + (1-\eta) \cdot |a_{l,i,t}| \cdot \sum_{k=1}^{n_{l+1}} |w_{l,i,k,t}| \quad (1)$$

where $u_l[i]$ - the contribution utility of the ith hidden unit in layer l at time t, $a_{l,i,t}$ - the activation output of the ith hidden unit in layer l at time t,$w_{l,i,k,t}$ - the weight connecting the ith unit in layer l to the kth unit in layer $l+1$ at time t, n_{l+1} - the number of units in the layer $l+1$, and η - the decay factor.

This utility function is used for selecting which neurons to reinitialize during the backpropagation process. The intuition behind this utility measure is that the product of the units' activation and outgoing weights gives information about how valuable this connection is to its consumers. Consequently, it makes sense to reinitialize neurons with a low utility measure, giving the network the capacity to learn and at the same time preserve some of its important knowledge gained so far during the training process.

Dohare et al. [12] presents the performance of the Continual Backpropagation algorithm across multiple common setups in CL, such as the online permuted MNIST [6] or Continual Image Net [15]. The algorithm successfully alleviates the plasticity loss of the learning system, but the reported experiments are based solely on learning networks that use ReLU as the activation function of their hidden layers. As we asses in our experiments, in the best case, the Continual Backpropagation algorithm successfully preserves the plasticity of the network, but, in the worst case, it predicts the outcomes no better than a network trained using simple backpropagation.

This indicates a severe limitation of the Continual Backpropagation algorithm, due to its dependence on the ReLU activation function for the network's hidden layers. This motivates the design of a new utility measure and reset approach for CL, which can be easily adapted to the activation function of the network.

Theoretical Considerations

The proposed approach aims at maintaining plasticity in deep CL by having its hyperparameters advantageously matched to the underlying activation function of the network. This method emerged from the simple idea of heuristically increasing the magnitudes of the gradients of the problematic neurons during the training process of the network.

Most families of activation functions suffer from a few common problematic regions. Some of these problems are presented in Figure 1. The majority of the problematic regions are related to the high magnitude post-activation values of the function, both positive and negative. It is thus worth designing reinitialization strategies which target neurons related to these two groups. A more detailed rationale behind the process of the analysis of gradient computations and how one can design an approach aimed at maintaining the plasticity of the network in CL is presented in Appendix A.

As a metric to measure how valuable a neuron is during the training process of the network, the following utility measure is proposed:

$$u_l[i] \leftarrow \eta \cdot u_l[i] + (1-\eta) \cdot a_{l,i,t} \quad (2)$$

where the terms have the same meaning as in Equation 1.

Equation 2 defines the utility value of a neuron as a decaying sum of the activation values of that neuron during the training process of the network. This represents a simplification of Equation 1, since it doesn't consider the outgoing weights of the node for which it calculates the utility value. This utility measure represents a good indicator of whether a neuron relates to the problematic region of low post-activation valued neurons or high post-activation valued neurons. Additionally, we apostolate that the activation value alone encapsulates enough information about the incoming weights of a node, and that it represents a cost-effective indicator for whether the node suffers from the vanishing gradient or the exploding gradient problem [16].

Algorithm Design

For the rest of the paper, we use the terms 'rescaling', 'reinitializing' and 'resetting' interchangeably, as the latter two are the terms usually used in CL literature for denoting

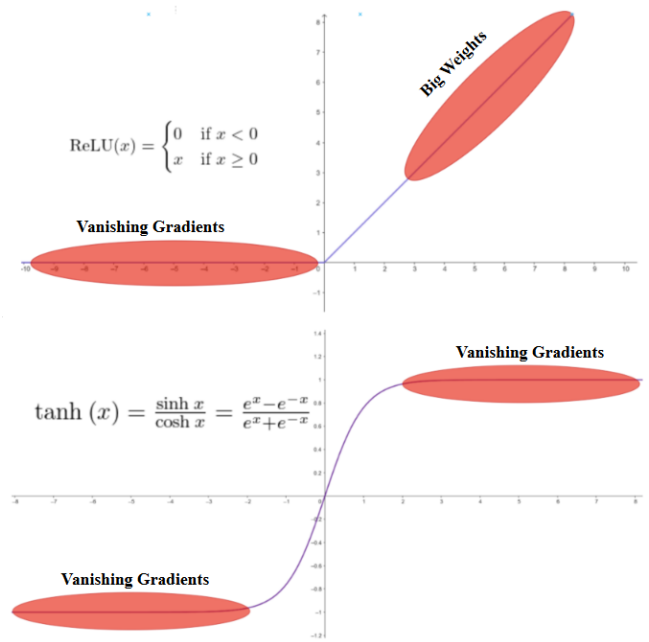

Figure 1: Illustration of some of the problems faced by deep learning activation functions(vanishing gradients and big weights). The top row represents ReLU, and the bottom row represents tanh.

an anthropic change that has been applied to some parts of the network. In this case, this change represents the rescaling of the weights and biases of some neurons of the network.

Firstly, let c_{low} denote the scaling factor of the low-utility score neurons, those neurons that are more likely to be associated with problematic regions that induce vanishing gradients during training, for example. Additionally, let ρ_{low} denote the replacement rate for the low utility score neurons of the network. This constant determines the frequency at which the neurons are rescaled for each particular layer of the network. A subtle but important consideration is the introduction of a table $age_l{}_{l=0}^{L-1}$, which measures the number of steps since the last time that a neuron's weights and biases have been rescaled. By introducing a maturity threshold m, we can protect the neurons from being rescaled too often, since it might take time for the utility measure of a neuron to calibrate after it has been rescaled. The variable $cnt_{low,l}{}_{l=0}^{L-1}$ serves as a counter, which counts the number of neurons that should be reinitialized at the current time during the training process of the network. Additionally, let d_l denote the initial distribution of the weights of the hidden layers.

We introduce analogous variables for the high utility score neurons of the network: c_{high}, ρ_{high} and $cnt_{high,l}{}_{l=0}^{L-1}$. The reason behind having two different strategies is that the optimal strategy for rescaling the weights and biases of the low-utility score neurons and ameliorating their plasticity loss symptoms is likely distinct from the optimal strategy for rescaling the weights and biases of the high-utility score neurons of the network. Intuitively, the low-utility score neurons of the network should be stimulated, and the high-utility score neurons of the network should be decelerated during the training process. The formalised pseudo-code of the proposed algorithm is presented in Algorithm 1.

The values of $c_{low}, c_{high}, \rho_{\text{low}}, \rho_{\text{high}}$ represent constant hyperparameters that should be chosen before the start of the training process. A rational approach for finding their optimal values is to perform a grid search on the corresponding problem whenever possible.

Algorithm 1 Modified Continual Backpropagation

Input: low/high replacement rates $\rho_{\text{low}}, \rho_{\text{high}}$; low/high coefficients $c_{\text{low}}, c_{\text{high}}$; decay rate η; maturity threshold m
Initialise weights $\{w_\ell\}_{\ell=0}^{L-1}$ with $w_\ell \sim d_\ell$
Initialise utilities $\{u_\ell\}_{\ell=0}^{L-1} \leftarrow 0$
Initialise ages $\{age_\ell\}_{\ell=0}^{L-1} \leftarrow 0$
Initialise counters $cnt_{\text{low},\ell}{}_{\ell=0}^{L-1}, cnt_{high,\ell}{}_{\ell=0}^{L-1} \leftarrow 0$
for each input x_t **do**
  $\hat{y}_t \leftarrow f(x_t; w)$ ▷ forward pass
  $\mathcal{L} \leftarrow \ell(y_t, \hat{y}_t)$ ▷ evaluate
  Update w with SGD (or variant) ▷ back-prop
  for $\ell = 1$ **to** $L-1$ **do**
    $age_\ell \leftarrow age_\ell + 1$ ▷ age each unit
    Update u_ℓ using (2)
    $n_{\text{eligible}} \leftarrow \#\{i \mid age_{\ell,i} > m\}$
    $cnt_{\text{low},\ell}$ += $\rho_{\text{low}}\, n_{\text{eligible}}$
    if $cnt_{\text{low},\ell} \geq 1$ **then**
      $r \leftarrow \arg\min_i u_{\ell,i}$ ▷ least-useful unit
      $w_{\ell-1}[:, r] \leftarrow c_{\text{low}} \cdot w_{\ell-1}[:, r]$ ▷ reset inputs
      $age_{\ell,r} \leftarrow 0$
      $cnt_{\text{low},\ell}$ −= 1
    end if
    $cnt_{\text{high},\ell}$ += $\rho_{\text{high}}\, n_{\text{eligible}}$
    if $cnt_{\text{high},\ell} \geq 1$ **then**
      $r \leftarrow \arg\max_i u_{\ell,i}$ ▷ most-useful unit
      $w_{\ell-1}[:, r] \leftarrow c_{\text{high}} \cdot w_{\ell-1}[:, r]$ ▷ reset inputs
      $age_{\ell,r} \leftarrow 0$
      $cnt_{\text{high},\ell}$ −= 1
    end if
  end for
end for

3 Experiments

In this section, we introduce the online permuted MNIST problem. We continue by presenting the model that will be used throughout the experiments, and describing the metrics that the models will be assessed on.

Problem Description

A computationally cheap problem based on the MNIST dataset [17] was chosen for this research. The MNIST dataset consists of 60000 (scaled down to 10000), 28×28, greyscale images of handwritten digits from 0 to 9, together with their correct labels. The small number of classes and simple image structures reduce the computational cost for training models on this dataset, which helps carry out many more experiments under a larger variety of different conditions, enabling us to perform more extensive and profound

research on the topic.

Scenarios commonly used in generating CL tasks involve permuting and/or splitting the available dataset [18]. In our case, we use the former one. The training process is divided into multiple tasks. Each different task is defined by the way we permute the pixels within the context of the original image's shape. Such an example is presented in Figure 2. For each new task, we draw a new permutation of the original image's structure, permute the whole dataset based on this permutation, and consider the result as the next task that the model will be trained on.

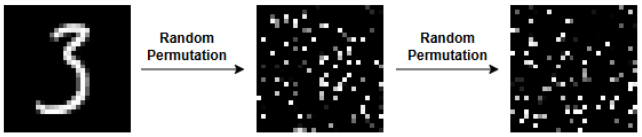

Figure 2: The process of generating different input configurations for each training task in the online permuted MNIST experiment. Each particular image of the dataset is altered according to a unique pre-determined random permutation at the beginning of a new task.

Model Setup

As a base model, we use feed-forward artificial neural networks with 3 hidden layers and 100 neurons per layer. We use Kaiming Initialization [19] for initializing the weights of the model, and predict the classification output of each of the 10 classes using the Softmax activation function [13]. The network is trained on a single pass through the data. In all the experiments, we use a batch size of 1 datapoint. For each batch, the network estimates the probabilities of each of the 10 classes, compares them to the correct label and performs stochastic gradient descent on the cross-entropy loss. The learning rate used for this experiment is constant and set to 0.003. The reason behind this choice is that the learning rate represents a hyperparameter that greatly influences the behaviour of the model. It is beyond our scope of research to compare the performance of models that use different learning rates during the training process.

Each experiment was performed by using 3 different seeds. For each particular task, the average accuracy of the model was measured across all 10^3 datapoints that the task consists of. In the presented plots, the performance of the model represents the mean and standard deviation calculated by considering a running window of size $n = 5$ over the results of individual tasks. The primary experiments were carried out on two network architectures: one with hidden layers employing ReLU activation functions, and another which uses tanh activations. A more general experiment involving a deeper architecture with interleaved activation functions is presented in Section 4.

Evaluation and Scoring

The primary metric used to evaluate the models' performance is their prediction accuracy on the online permuted MNIST problem. Additionally, due to the increased burden of resetting weights during the backpropagation process, the runtime of the model will also be measured. To gain better insights into the optimisation behaviour and generalisation abilities of the neural network, we will inspect the gradient covariance matrix and effective rank of the neural network.

The gradient covariance matrix measures the dot product of the gradients of different data point samples of the input dataset [3]. In particular, given sample indices i and j, the gradient covariance matrix is defined as:

$$C_k[i,j] = \frac{\langle \nabla_\theta L(\theta, x_i), \nabla_\theta L(\theta, x_j) \rangle}{\|\nabla_\theta L(\theta, x_i)\| \cdot \|\nabla_\theta L(\theta, x_j)\|} \tag{3}$$

The gradient covariance matrix gives insights into whether updates between a pair of datapoints generalise. One can think of a positive covariance as an indicator of generalisation for a pair of inputs, and a negative covariance as an indicator of interference [6]. Additionally, a pronounced block structure of the gradient covariance matrix indicates a sharp and unstable loss landscape [3]. The gradient covariance matrix represents an expensive-to-compute metric, as a model can have a significant number of parameters. It is thus usually calculated by sampling a limited number of data points from the input space of the problem.

The effective rank assesses how evenly distributed the singular values of the Singular Value Decomposition (SVD) of a matrix are. Intuitively, the singular values of the diagonal term of the SVD represent the magnitudes of stretching and scaling of different bases in the matrix's transformation context. A low effective rank of the features of a model implies that the model maps dissimilar inputs to similar prediction embeddings, thus making it more complicated to fit a classifier to these predictions [20]. The effective rank is a measure commonly used to asses the quality of the learned features of a model, as higher effective rank usually correlates to a higher performance in CL [21].

Roy et al. [22] introduces the following formula for measuring $erank(A)$, the effective rank of a matrix A of dimension $M \times N$:

$$erank(A) = exp(H(p_1, p_2, \ldots, p_Q)) \tag{4}$$

where $H(p_1, p_2, \ldots, p_Q)$ is the (Shannon) entropy given by:

$$H(p_1, p_2, \ldots, p_Q) = -\sum_{k=1}^{Q} p_k \cdot \ln(p_k)$$

with $Q = min(N, M)$, $p_k = \frac{\sigma_k}{\|(\sigma_1, \sigma_2, \ldots \sigma_Q)^T\|_1}$, and $\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_Q \geq 0$ representing the singular values of the diagonal matrix of the SVD of A.

4 Results

In this section, we present and discuss the results of the conducted experiments. We start with the hyperparameter search for designing the ReLU and tanh strategies. We continue by presenting the results of the experiments and introducing a more general, deeper network experiment.

Hyperparameters

We have performed a hyperparameter search for c_{low}, c_{high}, ρ_{low}, ρ_{high} used by the proposed algorithm. The found results are indicated in Table 1. The search space for c_{big} and c_{low}

consists of several values in the range $[-2, 2]$, as having scaling coefficients with greater magnitude can lead to exploding weights. Additionally, we consider several values in the interval $[0, 1]$ for ρ_{low} and ρ_{high}, as larger rates would reinitialize neurons too often.

The hyperparameter values selected for the ReLU-based strategy correspond to those yielding the highest accuracy on a ReLU-activated network. Similarly, the parameters selected for the tanh-based strategy were chosen based on their optimal performance on a tanh-activated network. Additionally, we have performed a hyperparameter search for the Continual Backpropagation algorithm and the L2 regularisation framework. The results of the search and the values for the other constant hyperparameters of the models are presented in Appendix B.

The results of the grid search for the c_{low}, c_{high}, ρ_{low}, ρ_{high} can be interpreted as follows:

- For the ReLU strategy, the c_{low} of -0.5 flips the sign of the node's input, which automatically activates the gradient in case it was 0 before, while also shrinking the weights of the node. We can interpret c_{high} of 0.5 as a factor which reduces the weight's magnitude, not allowing the network to commit to some particular feature, while also pulling the neuron away from the problematic high activation value region of the activation function.
- For the tanh strategy, we can interpret both the c_{high} of 0.2 and the c_{low} coefficient of 0.5 as steps taken in the activation function towards the place where the gradient is the greatest, that is, the 0 point. The approach is different from L2 regularisation due to its use of replacement rates ρ_{low} and ρ_{high} which only shrink the weights occasionally.

An insightful result is the fact that for both strategies, the values of ρ_{high} and ρ_{low} are higher than the lowest explored value for these parameters, 10^{-7}. This means that indeed, reinitializing the low-utility and high-utility neurons does help with maintaining plasticity, and that these groups adversely affect performance of the model in CL. Additionally, the values of the mentioned parameters are also lower than the maximum possible inspected value, most probably because reinitializing neurons too often adversely affects the stability of the model during the training process.

Accuracy and Effective Rank

As shown in Figure 3, the accuracy of the simple backpropagation model decreases with each subsequent task, which showcases its lack of plasticity. Moreover, the effective rank of the backpropagation algorithm also decreases, which is expected. The Continual Backpropagation algorithm preserves its plasticity if the underlying activation function is ReLU, but loses its plasticity and displays similar performance both for the accuracy and effective rank when the underlying activation function is tanh, which exhibits a significant limitation of the algorithm. The L2 framework maintains its plasticity in both scenarios, thus serving as a baseline for preserving plasticity in CL.

The designed ReLU strategy outperforms the other strategies, achieving a performance of $(88.65 \pm 0.4\%)$ (mean and std) across all tasks, compared to $(86.23 \pm 0.4\%)$ of L2 regularisation and $(84.98 \pm 0.7\%)$ of tanh strategy when used on a ReLU-activated network. The tanh strategy outperforms the other strategies, achieving a performance of $(87.60 \pm 0.32\%)$ compared to $(86.45 \pm 0.37\%)$ of L2 regularisation and $(84.12 \pm 0.5\%)$ of ReLU when used on a tanh-activated network. This demonstrates that the chosen parameters, when tuned to the underlying activation function, are able to preserve the plasticity of the network while also maintaining a high classification accuracy. The drop in performance of 1.05 p.p of the tanh strategy compared to the ReLU strategy in their native scenarios can be attributed to multiple reasons and indicates some possible limitations of the proposed framework, but further research is needed to make more accurate claims.

Interestingly enough, even though the performance of the strategies decreases when used on the opposite activation function scenario, the algorithms still preserve some plasticity of the model, as they perform better than simple backpropagation. Additionally, when inspecting the efficient rank of the ReLU strategy, we can see that it is lower than that of the Continual Backpropagation model for the ReLU activation scenario, even though the accuracy of the model is higher, and drops drastically in the tanh scenario, even though its accuracy is significantly higher than that of backpropagation and Continual Backpropagation. Gulcehre et al. [23] argues that the association between effective rank and agent performance is not as straightforward as previously assumed in offline RL scenarios. Our empirical analysis shows that while a low-prediction accuracy usually indicates a significant drop in the effective rank of the feature matrix, the implication doesn't usually work the other way around.

One might notice a slight increase in performance for the majority of the models for the first few tasks of the training process. This result is consistent with deep-learning applications in which the learning system is first trained on a large dataset and then fine-tuned on a smaller, more relevant dataset [12]. In the case of ReLU strategy applied on the ReLU activated network, this increase is maintained throughout the training process, as the network might still remember some relevant patterns of the previous training task, while a network trained from scratch can not benefit from any prior memory component.

Running Time

Rough estimates for the running time statistics of the backpropagation model, Continual Backpropagation model and the designed ReLU strategy are presented in Table 2. All experiments were conducted on the DAIC - the TU Delft High Performance Computing (HPC) Cluster [24], using one CPU per task from the available types of CPUs[1]. The experiments were run using 3 batch sizes $(1, 10, 100)$. As it turns out, the batch size represents an important hyperparameter which heavily influences the performance of the models in CL. It is especially important for the Continual Backpropagation and ReLU strategy models, most probably due to the granularity

[1] https://daic.tudelft.nl/docs/system/compute-nodes/#list-of-all-nodes

Strategy	ρ_{high}	ρ_{low}	c_{big}	c_{small}
ReLU	10^{-2}, 10^{-3}, 10^{-4}, $\mathbf{10^{-5}}$, 10^{-6}, 10^{-7}	10^{-2}, 10^{-3}, $\mathbf{10^{-4}}$, 10^{-5}, 10^{-6}, 10^{-7}	-2, -1, -0.5, -0.2, 0, 0.2, $\mathbf{0.5}$, 1, 2	-2, -1, $\mathbf{-0.5}$, -0.2, 0, 0.2, 0.5, 1, 2
tanh	10^{-2}, 10^{-3}, 10^{-4}, $\mathbf{10^{-5}}$, 10^{-6}, 10^{-7}	10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}, $\mathbf{10^{-6}}$, 10^{-7}	-2, -1, -0.5, -0.2, 0, $\mathbf{0.2}$, 0.5, 1, 2	-2, -1, -0.5, -0.2, 0, 0.2, $\mathbf{0.5}$, 1, 2

Table 1: Values used for the grid searches to find the best set of hyperparameters for the proposed algorithm tested on the permuted MNIST. The best-performing set of values for each activation strategy is in **bold**.

Algorithm	Size (#params)	Batch Size	Inference time (ns)	Backprop (ns)	Epoch time (s)
Backpropagation	99710 (784 inputs, 3 hidden layers of 100 nodes, 10 outputs)	1	12.6 ± 0.33	41.56 ± 2.10	8.11 ± 0.15
		10	29.94 ± 0.06	21.09 ± 0.13	1.54 ± 0.01
		100	59.56 ± 6.8	44.44 ± 0.12	0.30 ± 0.03
Continual Backpropagation		1	17.04 ± 4.28	199.58 ± 84.40	20.77 ± 5.02
		10	19.13 ± 1.15	235.59 ± 21.08	1.77 ± 0.02
		100	42.88 ± 0.37	291.45 ± 23.86	0.30 ± 0.01
ReLU		1	12.44 ± 0.12	230.13 ± 22.11	16.05 ± 0.15
		10	19.18 ± 0.40	233.95 ± 7.57	1.88 ± 0.04
		100	51.51 ± 9.97	293.16 ± 6.43	0.39 ± 0.08

Table 2: Running time statistics for different models and varying batch sizes applied on a ReLU-based network for the online permuted MNIST problem. We report the average time and standard deviation for a sample size of $n = 3$ seeds.

that defines the way that we compute the utility function of the reinitialization approaches. We omit presenting the accuracy performance based on the batch size as this is beyond the scope of this research.

As expected, the inference time is almost the same for all three models, since the introduced reinitialization approach doesn't change the way predictions are inferred. The average backpropagation time is significantly higher for the Continual Backpropagation and ReLU strategy models. This is due to the fact that the neurons' reinitialization procedure is considered a part of the backpropagation process. The average backpropagation time is similar between the Continual Backpropagation model and the ReLU strategy model, whereas the latter slightly outperforms the former in average epoch time for a batch size of 1. The backpropagation algorithm, as expected, outperforms the other two algorithms, achieving a speed-up factor of 1.97 compared to the ReLU strategy model for a batch size of 1.

The results suggest that incorporating a plasticity preservation mechanism imposes a significant computational overhead. However, as this overhead is highly sensitive to the algorithm's hyperparameters — particularly the reinitialization rates ρ_{low} and ρ_{high} — it is challenging to provide a precise estimate of the additional cost incurred.

Gradient Covariance

The gradient covariance matrices were measured right at the beginning of a new task, to asses how the gradients of the model behave when encountering the 'stress' of having to adapt to a new data distribution. Due to the increased complexity of computing the gradient covariance matrix, we have chosen to sample only 50 arbitrary data points from the MNIST dataset, adapted to the distribution of the underlying task. For this experiment, the training process was extended to 700 tasks, with the hope of being able to draw more conclusions from the behaviour of the gradients of the network. The covariance matrices are presented in Figure 4.

When visually inspecting the ReLU-based gradient covariance matrices, one can note that the ReLU backpropagation algorithm suffers from pronounced magnitudes of the gradients, which is a bad sign, especially since there is a high number of both positive and negative covariances of the gradients, which interfere with each other. For the Continual Backpropagation algorithm, the covariance magnitudes are smaller, which is a good indicator for the lack of interference between gradients, both positive and negative. Similarly, the ReLU strategy has slightly higher magnitudes, but also the majority of them are positive. These results are consistent with the accuracy of the predictions of these strategies.

For the tanh-based gradient covariance matrices, one can note the drastic increase in negative covariance values, both in number and magnitude, of the Continual Backpropagation model, which negatively influences the optimisation properties of the network. On the other hand, the gradients of the proposed tanh strategy are slightly positively correlated, which means that during training, the gradients of different data points generalise, leading to faster convergence and, on average, higher classification accuracy. This experiment once again has shown the loss-of-plasticity phenomenon of the backpropagation algorithm, and the low adaptability of the Continual Backpropagation algorithm to different activation functions. It also gave insights into the gradient behaviour of the analysed models early into the training process of a new task.

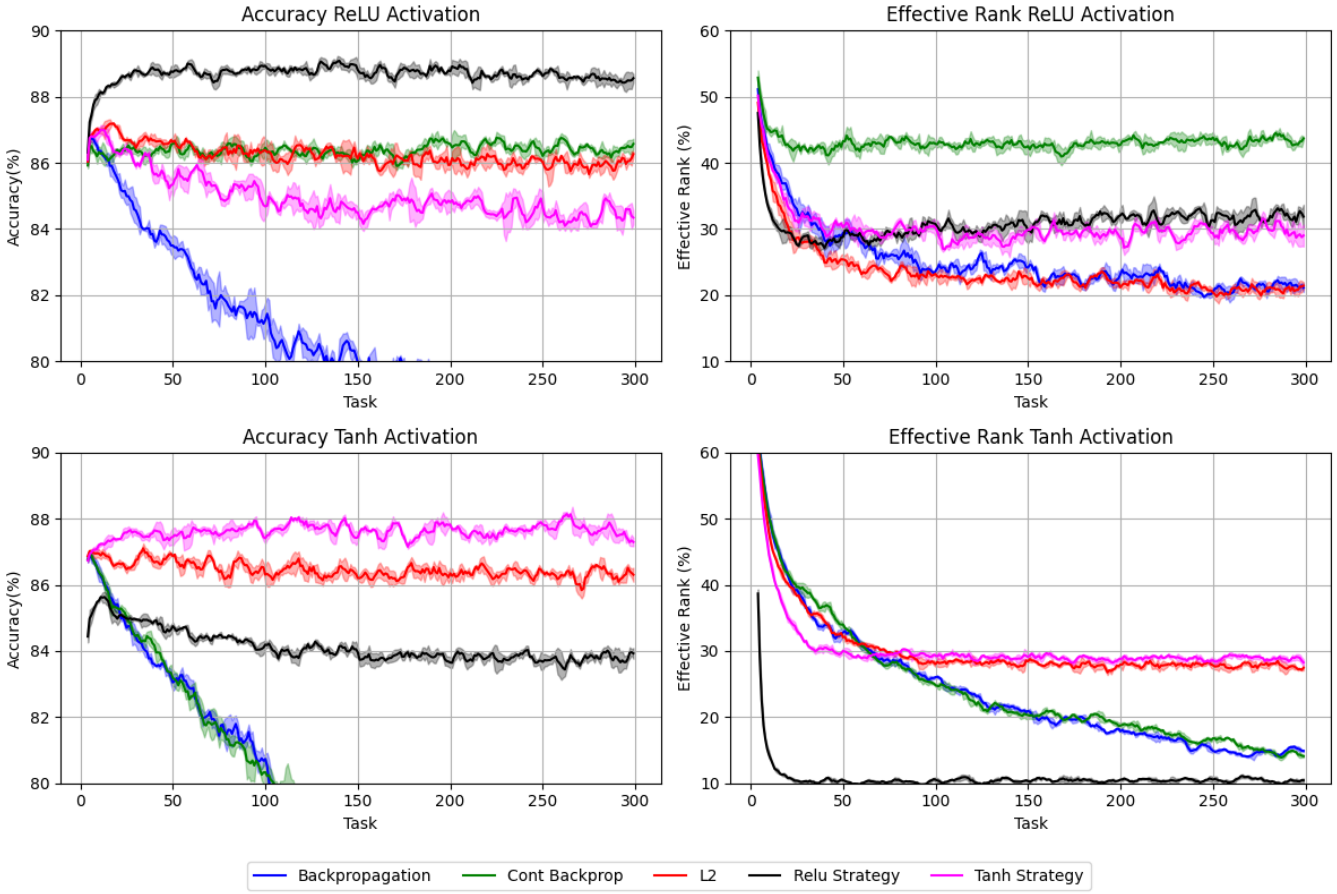

Figure 3: Online MNIST accuracy of models using ReLU and tanh activation strategies. Top row represents the accuracy and the effective rank for networks using the ReLU activation function. The bottom row represents the accuracy and the effective rank for networks using the tanh activation function. The results represent the average and standard deviation taken over 3 seeds for a running window of size ($n = 5$).

Twisted Activations

Employing various activation functions can be beneficial for the performance of deep models. For example, Hochreiter et al. [25] introduces the Long Short-Term memory, which leverages a combination of different activation methods that induce a memory component to the network, whereas Howard et al. [26] shows that by varying different activation functions, one can improve the latency of the model while maintaining similar performance. This gives us motivation for the next experiment, which uses as a base model a network that employs different activation functions for different layers of the network. The new network contains 5 hidden layers, instead of 3, and the corresponding activation functions of the hidden layers are as follows: ReLU, ReLU, tanh, tanh, ReLU. This combination was chosen randomly among all the possible $2^5 = 32$ combinations of hidden layers that make use of these 2 functions. We run the simple backpropagation algorithm as a baseline for a network that loses its plasticity during training and an L2 model, which acts like a baseline for a network that preserves its plasticity in CL. We also run the following three models:

- A network which employs the designed ReLU strategy for all the hidden layers of the network (denoted as `strategy_relu`).
- A network which employs the designed tanh strategy for all the hidden layers of the network (denoted as `strategy_tanh`).
- A network which alternates the 2 strategies based on the corresponding hidden layer of the network (denoted as `strategy_combined`).
- A network which alternates the 2 strategies based on the corresponding hidden layer of the network, but employs the suboptimal strategy (denoted as `strategy_opposite`).

The obtained results are presented in Figure 5. The backpropagation algorithm displays a big loss of plasticity, even starting from the first tasks of the training process. Next, we notice that the strategy that combines the best layer-wise strategies, depending on each layer's activation function, outperforms the other approaches. The performance is comparable to that of the tanh strategy when applied on a native tanh network and lower than that of the ReLU strategy when applied on a native ReLU network, as shown in Figure

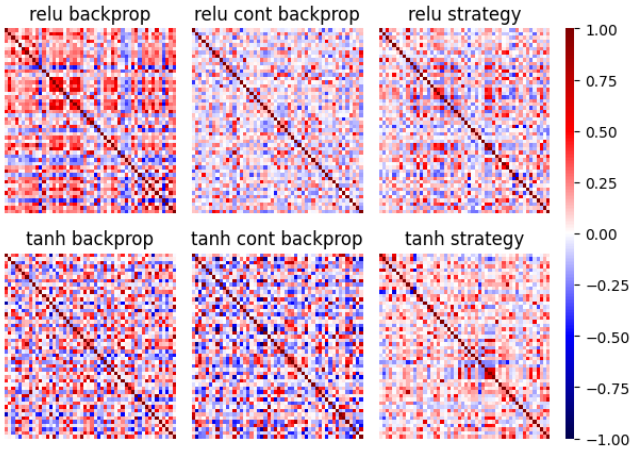

Figure 4: Gradient Covariance Matrices computed at the beginning of the 700th task for 50 datapoints. The top row represents the matrices of the backpropagation, Continual Backpropagation and designed ReLU strategy when applied on a ReLU-based network. The bottom row represents the matrices of the backpropagation, Continual Backpropagation, and designed tanh strategy when applied on a tanh-based network.

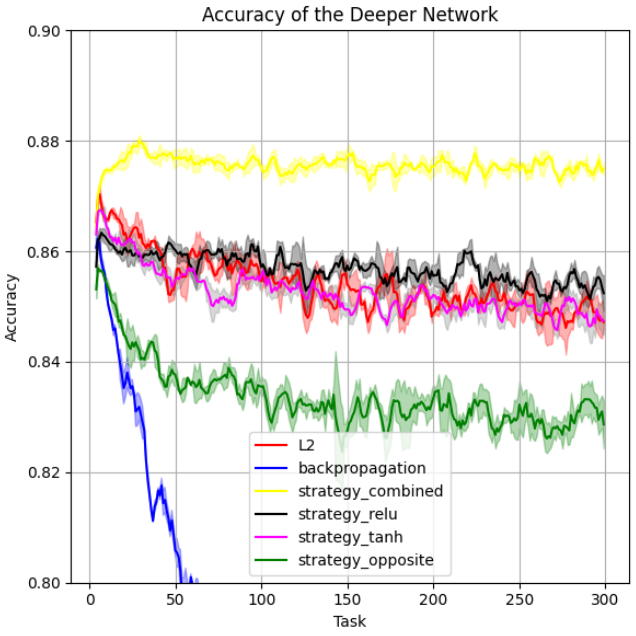

Figure 5: Online MNIST accuracy for the deeper network experiment. The results represent the average and standard deviation taken over 3 seeds for a running window of size ($n = 5$).

3. Further research is needed to explain this phenomenon. Additionally, even though the L2 regularisation doesn't dramatically decrease in prediction accuracy, it still loses some plasticity during the training process. We also notice that the ReLU and tanh strategies are comparable in performance with L2 regularisation. We assume that this is because the strategies are still correctly employed on some layers that use the same activation function as the ones for which the strategies were designed. Finally, the strategy that uses opposite activations compared to the corresponding network layer performs the worst among our designed strategies, as anticipated. This experiment has reinforced our belief that there is a strong correlation between classification accuracy and selecting the most suitable reinitialization algorithm for each layer of the network.

5 Conclusions and Future Work

This research focused on developing a framework for maintaining plasticity in CL that adapts its corresponding reinitialization strategy to the underlying activation function of the network. We have introduced a custom utility measure that estimates the activation value of each neuron. The strategy reinitializes neurons that have either low or high activation values, groups that are usually problematic during deep learning, especially in CL. We have identified that existing approaches, such as *Continual Backpropagation*, do not generalise well to various activation functions, and assessed that the proposed algorithm can adapt to ReLU and tanh, which come from two different families of activation functions. We have presented the prediction accuracy performance of the proposed model on the online permuted MNIST problem, along with other common metrics in CL, such as effective rank, gradient covariance matrix, and analysed its running time statistics. To the best of our knowledge, the concept of reinitializing neurons with high utility scores is novel, distinguishing our approach from prior work such as the Continual Backpropagation algorithm, which focuses solely on reinitializing low-utility neurons.

Two techniques related to the proposed reinitialization algorithm are LayerNorm [27] and BatchNorm [28]. LayerNorm normalizes each feature within a sample, while BatchNorm normalizes each feature across a batch of samples. It is worth investigating the BatchNorm approach in particular, as it also aims to reduce the magnitudes of post-activation values, aligning with the core principle of the newly introduced strategy. The main difference between the two approaches is that the proposed strategy only selectively reinitializes neurons, whereas BatchNorm does so for every single batch of data. For a performance evaluation of BatchNorm on the permuted MNIST problem, one can consult Appendix C.

Several directions for further investigating the usefulness of the proposed model remain. Firstly, it should be assessed whether the framework can mitigate plasticity loss in more general scenarios, including various supervised learning and deep RL popular benchmarks. Second, the model's ability to generalise to various activation functions — such as SeLU, GeLU, and others — should be assessed. Finally, evaluating the rate at which the model forgets previously acquired knowledge when exposed to new data distributions would provide valuable insights into its stability–plasticity trade-off.

6 Responsible Research

In this section, we reflect on the ethical aspects, address the broader implications and discuss the reproducibility of the

conducted research. Additionally, we also outline the specific ways in which LLMs assisted our research.

Transparency and Reproducibility
We have made sure that the results of our research are reproducible and transparent. We provided a public Github[2] repository, which is forked from the repository that was introduced in [12] and contains the implementation of the Continual Backpropagation algorithm. Our codebase contains the implementation of the designed algorithms, models and configuration files of the performed experiments. Additionally, it is capable of downloading the public MNIST dataset on which the experiments were performed. All the experiments, except for the measurement of the gradient covariance matrix, were performed by using 3 different seeds to ensure robust and reliable estimation of the obtained results. In the case of the gradient covariance matrix, only one seed was used, as this metric can't be averaged across multiple runs.

Limitation and Scope
The corresponding designed algorithm can mitigate plasticity loss in the online permuted MNIST experiment for networks which use ReLU or tanh as their activation function. However, this algorithm wasn't tested on other common CL benchmarks. Hence, the model should be properly investigated before being used in scenarios involving important decision-making or interaction with humans.

Unreported Research
Other utility functions and algorithms were tested as well during the research, but the majority of them didn't yield positive results and have not been described in this report due to the space constraints of the report and the time constraints of the project. This research initially started with the scope of exploring the effect of reinitializing high-utility score neurons on the performance of the Continual Backpropagation algorithm, but, after the reveal of some limitation of the Continual Backpropagation algorithm, it was reconceptualized to designing a more robust algorithm for resetting both high and low utility score neruons for solving the plasticity loss problem in CL.

Use of LLMs
LLMs have been used during this research for improving the grammar, style and spelling of the paper contents and for improving the visual figures of the obtained results. Some of the most representative prompts are presented in Appendix D.

7 Acknowledgments

We acknowledge and thank Mr. Laurens Engwegen for his guidance and the generously offered time throughout the duration of the research. We would also like to express our gratitude to Dr. Wendelin Böhmer for his support and insightful expertise. Research reported in this work was partially or completely facilitated by computational resources and support from the Delft AI Cluster (DAIC) at TU Delft [24], but remains the sole responsibility of the authors, not the DAIC team.

[2] https://github.com/flea2003/Loss-of-plasticity

References

[1] Liyuan Wang et al. *A Comprehensive Survey of Continual Learning: Theory, Method and Application*. 2024. arXiv: 2302.00487 [cs.LG]. URL: https://arxiv.org/abs/2302.00487.

[2] Robert M. French. "Catastrophic forgetting in connectionist networks". In: *Trends in Cognitive Sciences* 3.4 (1999), pp. 128–135. ISSN: 1364-6613. DOI: https://doi.org/10.1016/S1364-6613(99)01294-2. URL: https://www.sciencedirect.com/science/article/pii/S1364661399012942.

[3] Timo Klein et al. *Plasticity Loss in Deep Reinforcement Learning: A Survey*. 2024. arXiv: 2411.04832 [cs.AI]. URL: https://arxiv.org/abs/2411.04832.

[4] Clare Lyle et al. *Disentangling the Causes of Plasticity Loss in Neural Networks*. 2024. arXiv: 2402.18762 [cs.LG]. URL: https://arxiv.org/abs/2402.18762.

[5] Tudor Berariu et al. *A study on the plasticity of neural networks*. 2023. arXiv: 2106.00042 [cs.LG]. URL: https://arxiv.org/abs/2106.00042.

[6] Clare Lyle et al. *Understanding plasticity in neural networks*. 2023. arXiv: 2303.01486 [cs.LG]. URL: https://arxiv.org/abs/2303.01486.

[7] Aviral Kumar et al. *Implicit Under-Parameterization Inhibits Data-Efficient Deep Reinforcement Learning*. 2021. arXiv: 2010.14498 [cs.LG]. URL: https://arxiv.org/abs/2010.14498.

[8] Zaheer Abbas et al. *Loss of Plasticity in Continual Deep Reinforcement Learning*. 2023. arXiv: 2303.07507 [cs.LG]. URL: https://arxiv.org/abs/2303.07507.

[9] Ghada Sokar et al. *The Dormant Neuron Phenomenon in Deep Reinforcement Learning*. 2023. arXiv: 2302.12902 [cs.LG]. URL: https://arxiv.org/abs/2302.12902.

[10] Majid Ghasemi et al. *A Comprehensive Survey of Reinforcement Learning: From Algorithms to Practical Challenges*. 2025. arXiv: 2411.18892 [cs.AI]. URL: https://arxiv.org/abs/2411.18892.

[11] Jordan T. Ash et al. *On Warm-Starting Neural Network Training*. 2020. arXiv: 1910.08475 [cs.LG]. URL: https://arxiv.org/abs/1910.08475.

[12] Shibhansh Dohare et al. "Loss of Plasticity in Deep Continual Learning". In: *Nature* 632 (2024), pp. 768–774.

[13] Ian Goodfellow et al. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[14] Ludovic Trottier et al. *Parametric Exponential Linear Unit for Deep Convolutional Neural Networks*. 2018. arXiv: 1605.09332 [cs.LG]. URL: https://arxiv.org/abs/1605.09332.

[15] Shibhansh Dohare et al. *Maintaining Plasticity in Deep Continual Learning*. 2024. arXiv: 2306.13812 [cs.LG]. URL: https://arxiv.org/abs/2306.13812.

[16] George Philipp et al. *The exploding gradient problem demystified - definition, prevalence, impact, origin, tradeoffs, and solutions*. 2018. arXiv: 1712.05577 [cs.LG]. URL: https://arxiv.org/abs/1712.05577.

[17] Yann Lecun et al. "Gradient-Based Learning Applied to Document Recognition". In: *Proceedings of the IEEE* 86 (Dec. 1998), pp. 2278–2324. DOI: 10.1109/5.726791.

[18] Yen-Chang Hsu et al. *Re-evaluating Continual Learning Scenarios: A Categorization and Case for Strong Baselines*. 2019. arXiv: 1810.12488 [cs.LG]. URL: https://arxiv.org/abs/1810.12488.

[19] Kaiming He et al. *Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification*. 2015. arXiv: 1502.01852 [cs.CV]. URL: https://arxiv.org/abs/1502.01852.

[20] Andrew McCallum et al. "Reinforcement learning with selective perception and hidden state". In: 1996. URL: https://api.semanticscholar.org/CorpusID:60716402.

[21] Minyoung Huh et al. *The Low-Rank Simplicity Bias in Deep Networks*. 2023. arXiv: 2103.10427 [cs.LG]. URL: https://arxiv.org/abs/2103.10427.

[22] Olivier Roy et al. "The effective rank: A measure of effective dimensionality". In: *2007 15th European Signal Processing Conference*. 2007, pp. 606–610.

[23] Caglar Gulcehre et al. *An Empirical Study of Implicit Regularization in Deep Offline RL*. 2022. arXiv: 2207.02099 [cs.LG]. URL: https://arxiv.org/abs/2207.02099.

[24] The DAIC team. *The Delft AI Cluster (DAIC)*. 2015.

[25] Sepp Hochreiter et al. "Long Short-Term Memory". In: *Neural Computation* 9 (Nov. 1997), pp. 1735–1780. DOI: 10.1162/neco.1997.9.8.1735.

[26] Andrew Howard et al. *Searching for MobileNetV3*. 2019. arXiv: 1905.02244 [cs.CV]. URL: https://arxiv.org/abs/1905.02244.

[27] Jimmy Lei Ba et al. *Layer Normalization*. 2016. arXiv: 1607.06450 [stat.ML]. URL: https://arxiv.org/abs/1607.06450.

[28] Sergey Ioffe et al. *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*. 2015. arXiv: 1502.03167 [cs.LG]. URL: https://arxiv.org/abs/1502.03167.

A Computing Gradients

The standard equations for calculating the outputs, activations, and the gradients of the weights of a layer in a neural network are as follows:

$$z^{(L)} = w^{(L)} \cdot a^{(L-1)} + b^{(L)} \tag{5}$$

$$a^{(L)} = \sigma(z^{(L)}) \tag{6}$$

$$\frac{\partial C}{\partial w^{(L)}} = \frac{\partial z^{(L)}}{\partial w^{(L)}} \cdot \frac{\partial a^{(L)}}{\partial z^{(L)}} \cdot \frac{\partial C}{\partial a^{(L)}} \tag{7}$$

where C - the cost function, $w^{(L)}$ - the weight tensor of the L-th hidden layer, $z^{(L)}$ - the inputs of the L-th layer, $a^{(L)}$ - the activation value of the L-th layer, and σ - the activation function of the L-th hidden layer.

The second term of Equation 7 can be artificially easily manoeuvred by scaling the weights and biases of the layer by some constant factor. Let us denote such a scale factor by c. In this case, the old and new outputs of that node satisfy the following equation: $z^{(L)}_{new} = c \cdot z^{(L)}_{old}$, and $a^{(L)}_{new} = \sigma(z^{(L)}_{new}) = \sigma(c \cdot z^{(L)}_{old})$. Plugging these relation in Equation 7 leads to:

$$\frac{\partial C_{new}}{\partial w^{(L)}_{new}} = \frac{\partial z^{(L)}_{old}}{\partial w^{(L)}_{old}} \cdot \frac{\partial \sigma(c \cdot z^{(L)}_{old})}{c \cdot \partial(z^{(L)}_{old})} \cdot \frac{\partial C_{new}}{\partial \sigma(c \cdot z^{(L)}_{old})} \tag{8}$$

The first term of Equation 8 is exactly the same as the first term of the pre-scaling backpropagation product. The last term of Equation 8 depends on the way the loss function is measured in the network. The second term, however, can 'steer up' the activation function value of a particular neuron. In the case of a vanishing gradient problem, by carefully choosing a value for the scaling coefficient c, one can shift the post-activation value of the corresponding neuron in a more favourable region that is capable of catalysing the behaviour of the gradient of that neuron.

B Hyperparameters

The hyperparameter search results for the Continual Backpropagation algorithm and the L2 regularisation framework are indicated in Table 3 and Table 4, respectively. The constant hyperparameters of the models are presented in Table 5.

Algorithm	Weight Decay
ReLU Strategy	**0.001**, 0.0001, 0.00001
tanh Strategy	**0.001**, 0.0001, 0.00001

Table 3: Values used for the grid searches to find the best weight decay for **L2 regularisation** tested on the permuted MNIST. The chosen values are shown in **bold**.

Algorithm	Replacement Rate
ReLU Strategy	10^{-2}, 10^{-3}, **10^{-4}**, 10^{-5}, 10^{-6}, 10^{-7}
tanh Strategy	10^{-2}, 10^{-3}, **10^{-4}**, 10^{-5}, 10^{-6}, 10^{-7}

Table 4: Values used for the grid searches to find the best replacement rate for the **Continual Backpropagation** algorithm tested on the permuted MNIST. The chosen values are shown in **bold**.

Name	Value
`decay_rate` (η)	0.99
`maturity_threshold` (m)	100
`step_size`	0.003
`num_features`	100
`num_hidden_layers`	3

Table 5: Constant hyperparameters used across all runs for all the models.

C BatchNorm

BatchNorm computes the mean and standard deviation for each feature across a particular mini-batch of data. It does so by applying the following formula:

$$y = \frac{x - E[x]}{\sqrt{Var[x] + \epsilon}} \cdot \gamma + \beta \tag{9}$$

where x - initial distribution of a particular feature of the input data, y - computed normalised distribution, γ, β - learnable parameter vectors, and ϵ - small constant added to the variance to avoid division by zero.

For this experiment, we have employed 4 standard backpropagation models, with batch sizes of 5, 10, 50, and 100 data points, respectively. The base network was augmented by adding a BatchNorm layer between every hidden layer and activation layer of the network. The resulting prediction accuracies are shown in Figure 6. While all of the models maintain an almost stable performance throughout this experiment, their overall accuracy is notably lower than that achieved by the proposed strategy in our paper. This suggests that selectively reinitializing neurons during CL has the potential to be more effective for preserving the plasticity of the network than constantly normalising the activation values of the neurons for each data point.

D LLM Prompts Used

Some of the most representative prompts are presented in Table 6.

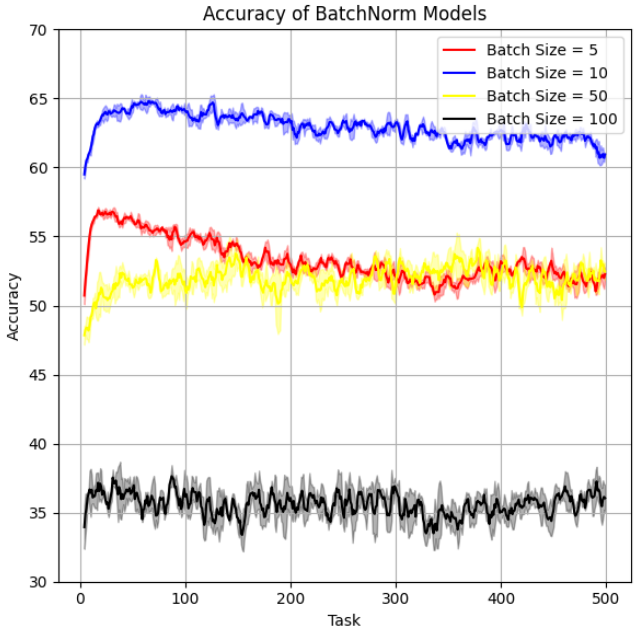

Figure 6: Online MNIST accuracy measured for the BatchNorm models. The results represent the average and standard deviation taken over 3 seeds for a running window of size ($n = 5$).

Scope	Prompt
Grammar, Style and Spelling	Improve the scientific tone and readability of this passage: ... Give suggestions on how to improve this passage:... Rewrite this paragraph into another one that is easier to understand: ...
Visual Figures	How to leave more space between these figures: [image], [code] Refactor this table such that there is no separator between adjacent columns: [LaTex] Improve the styling of this table: [LaTeX]

Table 6: Representative LLM prompts used for the scope of the research.