Exploring Alternatives to Full Neuron Reset for Maintaining Plasticity in Continual Backpropagation

Urtė Urbonavičiūtė[1]

Supervisors: 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: Urtė Urbonavičiūtė
Final project course: CSE3000 Research Project
Thesis committee: Wendelin Böhmer, Laurens Engwegen, Megha Khosla

Abstract

Deep learning systems are typically trained in static environments and fail to adapt when faced with a continuous stream of new tasks. Continual learning addresses this by allowing neural networks to learn sequentially without forgetting prior knowledge. However, such models often suffer from a gradual decline in learning ability, a phenomenon known as loss of plasticity. Recent work introduced Continual Backpropagation (CBP), which restores plasticity by fully reinitializing low-utility neurons. While this approach is effective, it can also disrupt the learning process. This research proposes and tests three less disruptive alternatives to full reinitialization: injecting Gaussian noise into weights, reinitializing weights from the original initialization distribution, and rescaling weights to match their initial variance. We evaluate these strategies using the Permuted MNIST benchmark. The present findings show that noise injection has results similar to original CBP, reinitializing weights from the original distribution shows a better performance, while weight rescaling performs much worse than CBP. This implies that less destructive methods can maintain plasticity effectively, with some alternatives offering better stability-plasticity trade-offs than CBP.

1 Introduction

Deep learning, the foundation of many modern neural network architectures, is based on gradient-based optimization methods such as backpropagation [1]. Backpropagation enables the efficient computation of gradients and allows models to adjust their parameters to minimize prediction error. This technique has achieved remarkable success in domains such as natural language processing [2], robotics [3], and game playing [4]. The standard deep learning systems work in two main stages: a training phase where the model learns information, and a usage phase where the model's parameters are fixed and the model tries to predict. While this design works well in static environments, it limits the system's capacity to adapt to new information over time. [5]

In contrast, continual learning addresses this limitation by enabling models to learn sequentially from a stream of tasks without forgetting previously acquired knowledge [6, 7]. This capability is essential for dynamic real-world applications, such as autonomous systems and lifelong learning agents [8].

However, while continual learning appears to be a significant improvement over traditional deep learning, over time the model loses the ability to adapt effectively to new tasks. This phenomenon is called *loss of plasticity* and it happens because networks become over-specialized in earlier tasks, and their learning capacity decreases. This issue has been extensively examined in recent studies, which highlight how neural networks struggle to maintain adaptability in long-term learning scenarios [9, 10].

To address this, researchers have explored various strategies that also involve reinitializing neurons, weights, or sub-networks of a model to keep the model adaptable. For instance, Sokar et al. [11] proposed ReDo, a method for deep reinforcement learning that identifies dormant neurons (units that are rarely active) and periodically resets their weights to restore the network's ability to learn and respond to new patterns.

Similarly, Dohare et al. [12] conducted a systematic study on loss of plasticity in both supervised and reinforcement learning. The researchers confirmed that standard deep learning models consistently lose their learning ability over time in continual learning settings, regardless of the dataset or architecture. Eventually, these models perform no better than shallow networks in later tasks. To address this issue, they proposed *Continual Backpropagation*, a backpropagation adaptation that maintains plasticity by periodically reinitializing a small fraction of low-utility neurons during training. During this process, the outgoing weights and biases of re-initialized units are set to zero, while incoming weights are reset using Kaiming uniform initialization. In this initialization technique, the weights are sampled from a uniform distribution scaled by the number of input connections, helping to maintain stable activation variance across layers [13].

While full reinitialization has shown promise in restoring plasticity, it also introduces potential downsides. Resetting neurons to zero may discard partially useful features, introduce instability since some existing network contributions are removed, and delay reactivation, as freshly reset neurons cannot immediately participate in learning. These limitations raise an important question:

> *Are there less disruptive alternatives to full neuron reinitialization for maintaining plasticity in Continual Backpropagation?*

This research investigates three such alternative strategies for handling low-utility neurons within the Continual Backpropagation framework.

One approach, inspired by Ash and Adams [14], scales and injects small Gaussian noise into the weights of low-utility neurons instead of completely resetting them. This added randomness is expected to help re-engage inactive neurons without completely discarding their learned information.

Another method reinitializes low-utility neurons by sampling their outgoing weights from the Kaiming uniform distribution instead of resetting them to zero. It is hypothesized that this method could provide a softer reset and allow neurons to learn faster.

A third strategy, based on the work of Niehaus et al. [15], rescales the weights of inactive neurons and restores their original variance. This method is expected to help these neurons become active again by increasing their sensitivity, without completely changing what they have already learned.

Together, these approaches are examined to assess whether they can improve plasticity in continual learning without the drawbacks associated with full neuron resets.

This paper is organized as follows. Section 2 reviews the relevant background and prior work related to this research. In Section 3, we introduce and motivate several alternative strategies to full neuron reset within the Continual Backpropagation framework. Section 4 outlines the experimen-

tal setup, including dataset details, hyperparameter configurations, and evaluation metrics. The empirical results of the proposed methods are presented and analyzed in Section 5. Section 6 briefly reflects on responsible research practices, followed by a discussion of the key findings and their implications in Section 7. Finally, Section 8 concludes the paper and suggests directions for future work.

2 Background and Related Work

This research builds on the work by Dohare et al. [12], who addressed the problem of loss of plasticity in continual learning and proposed a way to solve it. In addition, several other studies are explored to identify alternative neuron reinitialization strategies. This section outlines the key background and related work that form the foundation for this research.

2.1 Continual Backpropagation (CBP)

Dohare et al. [12] proposed *Continual Backpropagation (CBP)*, an effective modification of standard backpropagation designed to mitigate the loss of plasticity in continual learning settings. The main idea behind CBP is to periodically reinitialize a small fraction of neurons that have low utility.

Low utility is an indicator of neurons that have become stale or uninformative over time. It is measured based on both the magnitude of the outgoing weights and the neuron's recent activations. Specifically, in a feed-forward neural network, the contribution utility $u_l[i]$ of the i-th hidden unit in layer l at time t is updated using an exponential moving average:

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

Where:

- $h_{l,i,t}$ is the activation of the i-th unit in layer l at time t,
- $w_{l,i,k,t}$ is the weight connecting the i-th unit in layer l to the k-th unit in layer $l+1$,
- n_{l+1} is the number of neurons in the next layer $l+1$,
- $\eta \in (0,1)$ is the decay rate controlling the moving average.

In the reinitialization process, the outgoing weights and biases of selected neurons are reset to zero, while their incoming weights are re-sampled from the original initialization distribution (Kaiming uniform). This allows the network to reuse underperforming neurons and restore some of its ability to adapt to new tasks.

To ensure that neurons had enough opportunity to learn before being replaced, only those that exceed a maturity threshold are considered eligible for reinitialization. Among these mature neurons, a small fraction with the lowest utility scores is selected.

Although CBP is effective in maintaining long-term plasticity, full neuron reinitialization may discard partially useful features and delay reactivation, thus motivating the exploration of less disruptive alternatives.

2.2 Noise Injection

In their work on *WARM* (Weight Averaged Random Model), Ash and Adams [14] propose a simple yet effective regularization strategy for continual learning: periodically scaling weights by a factor λ and adding small Gaussian noise to the model parameters. They call this method *shrink and perturb*, and the key idea is to introduce controlled noise that encourages ongoing exploration of the weight space and helps the model escape overconfident or stale regions without catastrophic forgetting. In each training round t, the update perturbs each parameter θ_i^t as follows:

$$\theta_i^t \leftarrow \lambda \cdot \theta_i^{t-1} + p^t, \text{ where } p^t \sim \mathcal{N}(0, \sigma^2) \text{ and } 0 < \lambda < 1$$

The researchers demonstrate that this simple yet powerful technique significantly improves generalization while maintaining computational efficiency.

In the context of Continual Backpropagation, this idea is adapted by applying the same scaling and noise injection process to low-utility neurons instead of setting their weights to zero. Incoming weights, outgoing weights, and biases are scaled by a factor λ and perturbed by noise sampled from a Gaussian distribution with standard deviation σ.

2.3 Weight Rescaling for Variance Restoration

Weight modification methods are often developed as regularization techniques aimed at improving the performance of neural networks.

Batch normalization, introduced by Ioffe and Szegedy [16], is one of the main methods in deep network training that improves stability and performance. It reduces internal covariate shift by normalizing the input to each layer across a mini-batch. This technique makes training more stable and faster. Usually, other research builds upon batch normalization and aims to improve it or make the algorithm more efficient.

L2 regularization adds a penalty to the loss function based on the squared magnitude of the weights. This discourages overly large weights and promotes smoother, more generalizable models. It was formally introduced in the context of linear models as "ridge regression" by Hoerl and Kennard [17] and is now a standard regularization strategy in neural networks. L2 regularization is also used alongside CBP in the work of Dohare et al. [12]

Weight normalization, introduced by Salimans and Kingma [18], reparameterizes the weights by decoupling their magnitude and direction. It introduces a learnable scaling parameter, which must be optimized via gradient descent. While this method has proven to be effective, it is tightly coupled with the optimizer and introduces additional parameters, making it less ideal for a continual backpropagation (CBP) setting, where only the reinitialization method is being replaced and neurons are reinitialized individually, rather than through global retraining.

Weight interpolation is a recent approach explored in continual learning [19]. It blends weights from the current and previous models using linear interpolation after task-specific training. To be effective, it requires storing entire model copies and solving permutation matching to align neurons

across tasks. Although it is powerful for merging models and mitigating forgetting, its computational cost and architectural constraints (e.g. batch normalization updates and permutation alignment) make it less compatible with simple continual learning pipelines.

In contrast, **Weight rescaling**, proposed by Niehaus et al. [15], is quite suitable for integration with the CBP setup and is used in this research as one of the alternative methods to neuron reinitialization. Specifically, it is explored as a replacement for the original CBP reinitialization step where outgoing weights are set to zero. This technique does not require additional learnable parameters or separate models, works well with Kaiming initialization, ReLU activation, and the SGD optimizer, and has shown good performance on CIFAR-10 in supervised learning settings.

This method resets the variance of neuron weights without discarding their directional information. The procedure involves standardizing the neuron's weights (computing z-scores) and then rescaling them to match the standard deviation of the original initialization distribution, typically derived from Kaiming or Xavier initialization:

$$w^{(\ell)} \leftarrow \frac{w^{(\ell)} - \mu(w^{(\ell)})}{\sigma(w^{(\ell)})} \cdot \sigma_{\text{init}} + \mu(w^{(\ell)})$$

Where:

- $w^{(\ell)}$ are the weights of a neuron in layer ℓ,
- μ and σ represent the empirical mean and standard deviation of those weights,
- σ_{init} is the standard deviation used by the original weight initialization strategy.

Unlike full reinitialization, Weight rescaling in the CBP setting can preserve the mean and directional structure of the weights while restoring diversity, which is essential for continual learning. This method has the advantage of being non-destructive and not dependent on activation history, making it broadly applicable across architectures.

3 Alternative Strategies for Maintaining Plasticity in Continual Backpropagation

In the research of Dohare et al. [12], neurons with low utility are periodically reinitialized to maintain the plasticity of the model. During the reinitialization, the incoming weights are redrawn from a Kaiming uniform initialization distribution, while the outgoing weights and biases are set to zero.

In this work, the focus is on modifying this reinitialization step itself. Instead of the original reset, three alternative strategies are explored.

1. **Noise Injection.** Inspired by the Shrink-and-Perturb method [14], this approach aims to softly re-engage low-utility neurons. It scales the existing weights and biases by a factor λ and adds small Gaussian noise, thus retaining some memory while encouraging diversity. Two versions are implemented: one where noise is injected only into the incoming weights (leaving biases and outgoing weights as zero), and another where noise is added to incoming weights, outgoing weights, and biases. The noise injection follows the same formula as it was proposed by Ash and Adams [14]:

$$\theta_t = \lambda \cdot \theta_{t-1} + \epsilon, \quad \epsilon \sim \mathcal{N}(0, \sigma^2)$$

2. **Reinitialization from Kaiming Uniform Distribution.** As a more principled alternative to zeroing out weights, this method samples both incoming and outgoing weights from the Kaiming uniform distribution used during model initialization, while keeping the biases at zero. By assigning small random values instead of zeros, it aims to avoid leaving neurons in an inactive state and allows them to start contributing more quickly after reinitialization.

3. **Weight Rescaling.** Based on the method proposed by Niehaus et al. [15], this strategy rescales the incoming weights of a neuron to match the variance of the original initialization, without fully discarding the previously learned information. It uses the same formula as it was proposed in their research paper:

$$\mathbf{w} \leftarrow \left(\frac{\mathbf{w} - \mu(\mathbf{w})}{\sigma(\mathbf{w})} \right) \cdot \sigma_{\text{init}} + \mu(\mathbf{w})$$

All methods are applied within a feedforward neural network trained using stochastic gradient descent (SGD). Further details of the network architecture, training loop, and task-specific configuration are given in the next section.

4 Experimental Setup

This section outlines the experimental setup, including the dataset, model architecture, hyperparameter configurations, and evaluation metrics.

4.1 Permuted MNIST Benchmark

The new methods are evaluated using the **Permuted MNIST** benchmark [20], a widely used and computationally efficient setup for evaluating continual learning algorithms.

A more extensive version of this benchmark, referred to as *Online Permuted MNIST*, is used by Dohare et al. [12]. In that setting, each task applies a unique random pixel permutation to the entire MNIST dataset, presented one sample at a time with no mini-batches or task change indicators. This setup is designed to test long-term plasticity in continual learning setting.

In our experiments, a reduced version of this benchmark is used to allow faster iterations and evaluation. The full MNIST set is shuffled, and the first 10,000 samples are selected for training. For each of the 600 tasks, a new unique pixel permutation is applied to the entire 10,000-sample dataset. This means that the dataset is fully permuted 600 times, resulting in 600 distinct permutations and task transitions. Although this setup is smaller than the 60,000 sample configuration used in Dohare et al. research [12], it preserves key properties of continual learning and remains effective for evaluating plasticity and learning stability over time.

All experiments are repeated 20 times for final evaluations and 10 times during hyperparameter tuning.

4.2 Network Architecture

The architecture used in the Permuted MNIST experiments is a fully connected feedforward network with three hidden layers, each containing 100 ReLU-activated neurons. This reduced architecture, compared to the one used in the original paper of Dohare et al. [12], is sufficient for comparing continual backpropagation with our proposed alternatives, as our focus is not on the final accuracy, but rather a comparison between the CBP and the alternative strategies.

All models are implemented using the PyTorch framework [21].

4.3 Computational Environment

Due to the large number of tasks and multiple experimental runs across methods and hyperparameter settings, high-performance computational resources were required. Most experiments were conducted on the DelftBlue supercomputer [22] and the Delft AI Cluster (DAIC) [23] at TU Delft. These clusters enabled efficient parallel execution and handling of data-intensive workloads.

4.4 Evaluation Metrics

To assess the performance of each method on the **Permuted MNIST** benchmark, the following metrics are recorded continuously during training:

- **Overall Task Accuracy** - the model's average prediction accuracy across all samples within the task.
- **Initial Task Accuracy** - average accuracy of the first 10% of samples in each task, indicating how quickly the model learns.
- **Per-Task Accuracy** - accuracy evaluated for an individual task, using all data samples from that task.
- **Dead Unit Ratio** - the proportion of neurons with zero utility, indicating under-utilized or non-contributing neurons.
- **Approximate Rank** - a proxy for the representational capacity and plasticity of each layer. It reflects how many principal components (or singular values) are needed to explain most of the variance in the matrix [24].

These metrics together provide a comprehensive view of the model's learning behavior. Among them, **overall task accuracy** is arguably the most important in the context of continual learning, especially when solving the problem of loss of plasticity. This is because plasticity loss is directly linked to a model's inability to acquire new knowledge, which is presented as a drop in accuracy over time. Therefore, our main objective is to maintain high overall accuracy throughout training. In some cases, when a method or configuration shows noticeably lower accuracy, deeper analysis of other metrics might not be even done.

Nevertheless, the remaining metrics still offer valuable insights. **Initial task accuracy** is especially useful for verifying hypotheses regarding learning speed. For example, we hypothesize that when weights are not reset to zero (as in standard CBP), models may start learning faster, which would be reflected in this metric. This trend can often be confirmed further by inspecting the **per-task accuracy** curves.

The **approximate rank** helps evaluate how expressive or flexible the model remains over time. A high rank suggests that the model has diverse internal features and is capable of adapting to new information, while a low rank may show that the model's features have become less flexible.

The **dead unit ratio**, also used by Dohare et al. [12], measures the proportion of neurons whose utility is zero according to the contribution-based utility function. While this metric is tightly coupled to the specific utility function and does not directly measure plasticity or accuracy, it is still useful. It allows comparison of how different methods affect neuron usage, and whether they prevent the inactivity of units.

Together, these metrics support a comprehensive understanding of how each proposed strategy affects the learning and the network.

All line graphs that present the evaluation metrics report the mean value across multiple runs and include shaded areas that represent standard deviation. Lines in the plots are averaged over 5 steps.

4.5 Hyperparameter Configuration

CBP has several key hyperparameters: the *replacement rate*, which determines the probability that a neuron is selected for reset; the *maturity threshold*, which ensures that neurons are not reset too soon after reinitialization; the *decay rate*, which controls the exponential moving average used in utility computation; the *learning rate*, which affects how quickly the model updates its weights; and the *step size*, which controls the magnitude of each weight update during the training process.

For all continual backpropagation variants analyzed in Permuted MNIST experiments, the best-performing hyperparameter settings reported in the Dohare et al. [12] research paper are used: a learning rate of 0.003, a replacement rate of 1×10^{-5}, a decay rate of 0.99, a maturity threshold of 100, and a step size of 0.001.

Noise injection is the only method among our alternatives that introduces additional hyperparameters: the shrinkage factor λ and the standard deviation σ of the Gaussian distribution. Also, two variants of this method are considered, one in which noise is applied only to the incoming weights, and another in which noise is added to incoming weights, outgoing weights, and biases.

The following hyperparameter values were explored:

- $\sigma \in \{0.1, 0.08, 0.07, 0.06, 0.05, 0.04, 0.03, 0.02, 0.01, 0.005, 0.001\}$ - controls the magnitude of the added noise.
- $\lambda \in \{0.2, 0.4, 0.6, 0.8\}$ - controls the degree of weight shrinkage.

The results in Figure 6 (Appendix A) show that applying Noise injection to both weights and biases consistently outperforms Noise injection only to the incoming weights.

Among the standard deviations, $\sigma = 0.03$ and $\sigma = 0.04$ yielded the best accuracy (Appendix A, Figure 7). It can be also seen that the results of different σ make a parabola as accuracy for both very small and very large values of σ decreases. As shown in Figure 8 (Appendix A), higher σ values also lead to a higher approximate rank. In contrast, lower

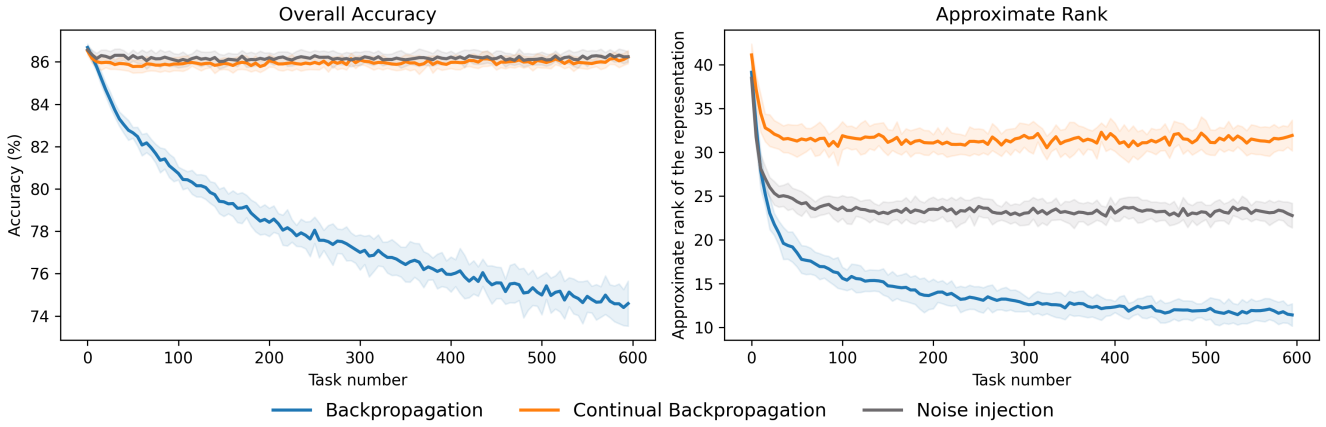

Figure 1: Noise injection method compared to Continual Backpropagation and Backpropagation across two metrics: (a) Overall accuracy, (b) Approximate rank

σ values result in higher percentage of dead neurons (Appendix A, Figure 9). Based on these results, $\sigma = 0.04$ is selected as the optimal value.

As for λ, lower values (e.g., 0.2) resulted in fewer dead neurons (Appendix A, Figure 12), while higher values (e.g., 0.8) had higher approximate rank (Appendix A, Figure 11). In terms of accuracy, $\lambda = 0.2$ performed best, while $\lambda = 0.4$ came as a close second (Appendix A, Figure 10). For the final evaluations, $\lambda = 0.2$ is selected.

5 Results of Alternative Methods to Full Neuron Reinitialization

This section presents the results of the proposed alternative neuron reinitialization strategies, evaluated across multiple criteria, and compares them to standard Continual Backpropagation (CBP) and Backpropagation (BP). The results presented here are based on the Permuted MNIST dataset. Additional results on the Incremental CIFAR dataset are provided in Appendix C.

5.1 Results of Noise Injection

The Noise injection strategy used in this experiment applies Gaussian noise to both incoming and outgoing weights as well as biases, with a standard deviation $\sigma = 0.04$ and a shrinkage factor $\lambda = 0.2$. Compared to standard CBP, this method achieves nearly identical overall accuracy, as shown in Figure 1a. It also maintains a similar percentage of dead neurons and comparable performance in the first 10% samples in each task (Appendix B, Figures 13 and 14). The most noticeable decline appears in the approximate rank metric, where the values fall 10 units lower than those of CBP, but still remain around 10 units higher than BP (Figure 1b).

5.2 Results of Reinitialization from Kaiming Uniform Distribution

The Reinitialization from Kaiming uniform distribution method proved to be better than standard Continual Backpropagation across all evaluated metrics. In term of overall accuracy, it performs slightly better than CBP (Figure 2a). A more notable improvement is observed in accuracy of the first 10% data points of each task, where it performs about 2% better than CBP (Figure 2b). This is further illustrated by the accuracy on an individual task, such as the 400th, where the method shows higher initial accuracy before converging to a similar level as CBP (Figure 2c).

The most significant difference is observed in the approximate rank, which is consistently higher by about 10 units compared to CBP, indicating stronger representational capacity and better plasticity (Figure 2d). In addition, the method results in slightly fewer dead neurons throughout training (Appendix B, Figure 15).

5.3 Results of Weight Rescaling

Weight rescaling was tested by applying it to both incoming and outgoing weights, as well as to incoming weights only. In most cases, the outcomes were similar across both approaches, except in a few cases where rescaling only the incoming weights resulted in slightly better performance. Therefore, the upcoming figures present results where only the incoming weights are rescaled.

Initial experiments in which weight rescaling is applied in the standard continual backpropagation (CBP) setting showed no improvement over standard backpropagation. Upon inspecting the weights before and after rescaling, it was noticed that they did not change much during this process. This led us to hypothesize that CBP's default hyperparameters, particularly the replacement rate, which controls how many neurons are replaced, may limit the degree to which neurons are able to deviate from their original weights.

To test this, further experiments were conducted with lower replacement rates (10^{-6} and 10^{-7}) in order to affect fewer neurons and potentially allow more time for their weights to change before rescaling. However, no improvement in performance was observed, as the model continued to behave similarly to standard backpropagation (Figure 3).

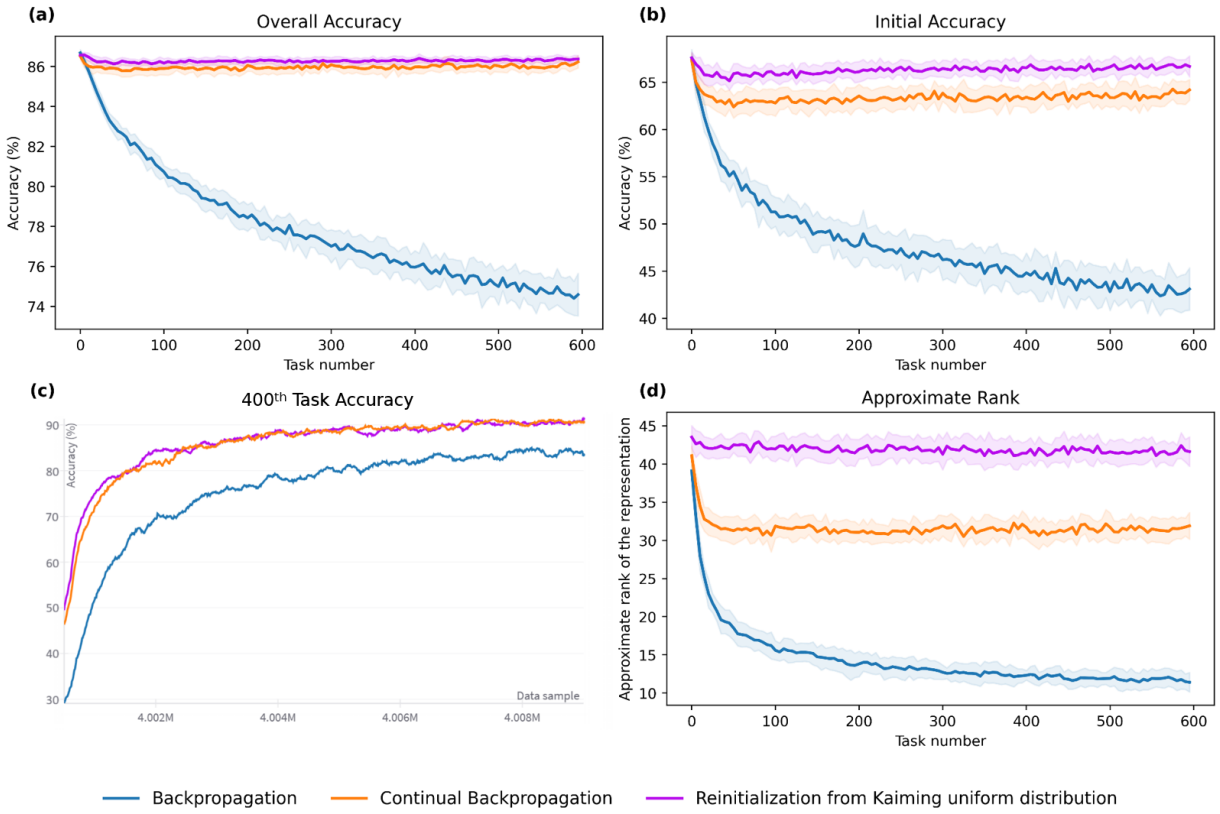

Figure 2: Reinitialization from Kaiming uniform distribution method compared to Continual Backpropagation and Backpropagation across various metrics: (a) Overall accuracy, (b) Initial accuracy, (c) Accuracy of the 400th task, (d) Approximate rank

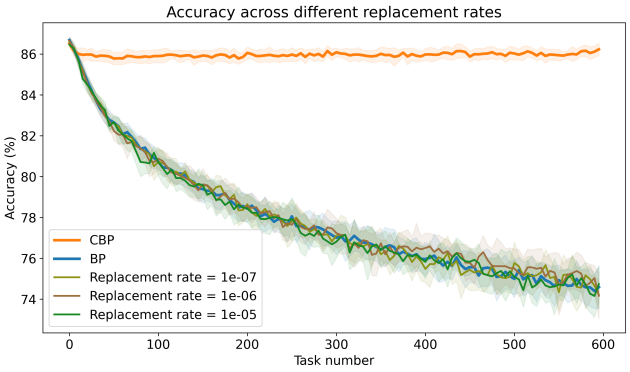

Figure 3: Accuracy of initial Weight rescaling application (replacement rate = 10^{-5}) and Weight rescaling with lower replacement rates, compared to Continual Backpropagation (CBP) and Backpropagation (BP)

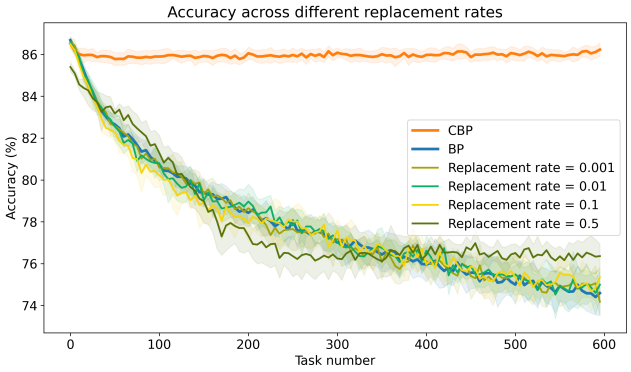

Figure 4: Accuracy of Weight rescaling applied once per task compared to Continual Backpropagation (CBP) and Backpropagation (BP)

Another approach, inspired by the original weight rescaling paper [15], was explored. In that research, weight rescaling is applied once per epoch in a supervised learning setting, therefore we decided to apply weight rescaling less frequently. A sample counter was introduced to trigger rescaling once per task (every 10,000 samples). Also, the replacement rate was increased to compensate for the reduced frequency, allowing more neurons to be rescaled at once.

Several replacement rates were tested under this setup (0.5, 0.1, 0.01, 0.001), as shown in Figure 4. While most configurations again showed no benefit, a high replacement rate of 0.5 resulted in slightly more stable accuracy over time, although performance remained low overall.

The final results, in which the best performing version of each alternative method is presented, can be seen in Figure 5.

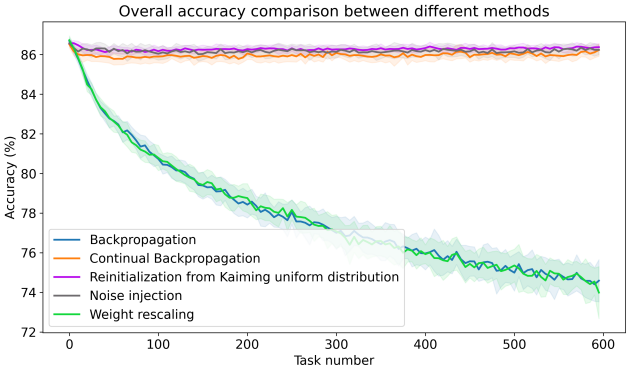

Figure 5: Final overall accuracy comparison between CBP, BP, Noise injection, Weight rescaling and Reinitialization from Kaiming uniform distribution

6 Responsible Research

This section reflects on the ethical aspects of the research, scientific integrity, and reproducibility of the results.

6.1 Ethical Issues

This research does not use any personal, sensitive, or identifiable data. The experiments are conducted on standard benchmark datasets Permuted MNIST and Incremental CIFAR-100, which consist of synthetic or anonymized image data. The research focuses purely on methods aimed at improving plasticity in continual learning models.

No direct deployment or real-world integration of the models was carried out. However, it is worth noting that since the extensive experiments are only done on a narrow synthetic benchmark (Permuted MNIST), the results are not necessarily representative of real-world applications and therefore may not generalize broadly. This limitation should be considered when interpreting the findings or applying them to real systems.

6.2 Scientific Integrity

Throughout the research, ChatGPT[1] was used only for grammar correction, synonym suggestions, and sentence rephrasing. Example prompts used for these purposes are provided in the Appendix D for transparency.

All methodological and experimental contributions were independently developed and implemented by the author. All references to existing methods (e.g., Continual Backpropagation) are cited.

6.3 Reproducibility of the Results

The results in this research are designed to be reproducible, as the detailed description of the experimental setup is provided in Section 4. It includes all relevant hyperparameters, the model architecture, and the evaluation metrics. Each experiment was run multiple times to ensure statistical robustness, and the number of runs is explicitly reported.

[1]ChatGPT by OpenAI, https://chat.openai.com

Additionally, the full source code is publicly available in a *GitHub repository*. The repository includes instructions for setup, installation, and running experiments.

It is important to note, that fixed random seeds were not used across all experiments since the goal was to evaluate the robustness of the results. However, the variability introduced by this is minimal and accounted for using multiple runs and standard deviation reporting.

All code was written in Python 3.8 using standard machine learning libraries such as PyTorch [21].

7 Discussion

This section reflects on the results obtained from the Online Permuted MNIST benchmark and provides further interpretation. The proposed alternative methods are compared with Continual Backpropagation (CBP) and discussed in the broader context of maintaining plasticity in continual learning.

Among the experimented methods, sampling both incoming and outgoing weights from the Kaiming uniform distribution shows a consistent improvement over standard CBP across all metrics. From the single-task accuracy and initial accuracy graphs, it can be concluded that a network in which outgoing weights are not periodically set to zero is able to learn new information faster, consequently improving overall accuracy. This likely happens because outgoing weights initialized from the Kaiming distribution provide a meaningful gradient path for learning from the start, unlike zeroed weights, which need several updates before becoming effective. Furthermore, the higher approximate rank indicates that the network flexibly responds to new data and is more plastic. The reduction in the number of dead neurons also supports this, suggesting that reinitializing outgoing weights with non-zero values increases the chance that these neurons will remain useful and active over time, thus the likelihood of them becoming dead again reduces.

As for Noise injection, it performs similarly to CBP across almost all metrics, with the exception of approximate rank, where its performance shows a noticeable decline. This may be explained by the fact that Noise injection perturbs existing representations rather than reinitializing them completely, making neurons less sensitive to new input patterns.

Additionally, while the best results are observed with a shrinkage factor $\lambda = 0.2$, it could be argued that this value causes a substantial loss of prior knowledge since the weights are strongly scaled down before noise is added. From the hyperparameter tuning results (Figure 10), it can be seen that $\lambda = 0.4$ could be considered as a strong alternative. It offers a better trade-off between preserving previously learned information and maintaining accuracy similar to that of CBP. Moreover, if the goal is to retain even more information from prior tasks while still introducing plasticity, using a smaller noise standard deviation, such as $\sigma = 0.03$, could be beneficial. Our tuning experiments suggest that this standard deviation yields similar accuracy as $\sigma = 0.04$ (Figure 7) while introducing less disruption to the network.

Finally, the Weight rescaling strategy appeared to be ineffective in the continual learning setting used in this research.

This is likely due to the differences between continual learning and the static, supervised learning conditions under which weight rescaling was originally proposed and shown to be effective.

The main goal of Weight rescaling is to restore the variance of a neuron's weights while preserving its directional structure. This can be beneficial in supervised learning, where data distributions are static and previously useful directions remain relevant. In contrast, continual learning involves task shifts and non-stationary data, where older directions may become not useful anymore.

Thus, rescaling stale neurons in continual learning changes their magnitude but not their usefulness. The preserved directions may continue to align with outdated features and prevent the neuron from learning representations relevant to new tasks. Without a change in direction, such neurons are unlikely to reactivate or meaningfully contribute to learning.

These findings of the experiments suggest that while Weight rescaling offers a non-destructive alternative to reinitialization in theory, it may not be sufficiently forceful to restore plasticity under a continual learning setting.

8 Conclusions and Future Work

To conclude this research, three alternative reinitialization strategies were proposed and evaluated in the Continual Backpropagation (CBP) framework: Noise injection, Reinitialization from Kaiming uniform distribution, and Weight rescaling. All methods were tested on the Online Permuted MNIST benchmark.

The most effective method was Reinitialization from Kaiming uniform distribution. It consistently outperformed the standard CBP approach across all metrics, including overall accuracy. Unlike full reinitialization, where outgoing weights are reset to zero, this method initializes outgoing weights with values sampled from the Kaiming uniform distribution, allowing neurons to remain immediately trainable. This reduces the delay in reactivation, allows faster integration of new information, and improves plasticity.

The Noise injection method also showed promising results, performing comparably to the original CBP and preserving some existing network contributions. This preservation of past knowledge makes the method more suitable for scenarios where stability is as important as plasticity.

In contrast, Weight rescaling did not result in any improvements and performed similarly to standard backpropagation. This is likely because Weight rescaling was originally developed for static supervised learning, where resetting variance improves convergence. In continual learning, however, merely restoring variance does not redirect neurons toward new tasks, and may not be sufficient to overcome loss of plasticity.

Future Work. In the future, testing could be performed on a separate test set after the model has finished learning a task. Instead of averaging accuracy over all training steps, the accuracy of test set could be used. This would provide a better understanding of how well the model performs after training, without taking into account the learning process itself.

Additionally, the runtime of each alternative method could be measured and ways to make them more efficient could be identified.

Furthermore, the current experiments use the best performing hyperparameters from the original CBP setup. However, it is possible that other hyperparameter configurations, which are suboptimal for CBP, might work better with the proposed reinitialization methods. Therefore, tuning the CBP hyperparameters specifically for these alternative methods could potentially improve performance.

Finally, evaluating these alternatives on more complex continual learning benchmarks could provide more insights into their scalability and general applicability.

References

[1] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

[2] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate, 2016.

[3] Elia Kaufmann, Leonard Bauersfeld, Antonio Loquercio, Matthias Mueller, Vladlen Koltun, and Davide Scaramuzza. Champion-level drone racing using deep reinforcement learning. *Nature*, 620:982–987, 08 2023.

[4] David Silver, Aja Huang, Christopher Maddison, Arthur Guez, Laurent Sifre, George Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of go with deep neural networks and tree search. *Nature*, 529:484–489, 01 2016.

[5] Gary Marcus. Deep learning: A critical appraisal, 2018.

[6] German I. Parisi, Ronald Kemker, Jose L. Part, Christopher Kanan, and Stefan Wermter. Continual lifelong learning with neural networks: A review. *Neural Networks*, 113:54–71, 2019.

[7] Ioannis Prapas, Babak Derakhshan, A. Rahman Mahdiraji, Oliver Hinz, and Sebastian Voss. Continuous training and deployment of deep learning models. *Datenbank-Spektrum*, 21:203–212, 2021.

[8] Khadija Shaheen, Muhammad Abdullah Hanif, Osman Hasan, and Muhammad Shafique. Continual learning for real-world autonomous systems: Algorithms, challenges and frameworks, 2022.

[9] Martial Mermillod, Aurélia Bugaiska, and Patrick Bonin. The stability-plasticity dilemma: Investigating the continuum from catastrophic forgetting to age-limited learning effects. *Frontiers in psychology*, 4:504, 08 2013.

[10] Clare Lyle, Zeyu Zheng, Evgenii Nikishin, Bernardo Avila Pires, Razvan Pascanu, and Will

Dabney. Understanding plasticity in neural networks, 2023.

[11] Ghada Sokar, Rishabh Agarwal, Pablo Samuel Castro, and Utku Evci. The dormant neuron phenomenon in deep reinforcement learning, 2023.

[12] Shibhansh Dohare, J. Fernando Hernandez-Garcia, Qingfeng Lan, Parash Rahman, A. Ruapm Mahmood, and Richard S. Sutton. Loss of plasticity in deep continual learning. *Nature*, 632:768—774, 2024.

[13] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

[14] Jordan T. Ash and Ryan P. Adams. On warm-starting neural network training, 2020.

[15] Lukas Niehaus, Ulf Krumnack, and Gunther Heidemann. Weight rescaling: Applying initialization strategies during training, 06 2024.

[16] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift, 2015.

[17] Arthur E. Hoerl and Robert W. Kennard. Ridge regression: Biased estimation for nonorthogonal problems. *Technometrics*, 12(1):55–67, 1970.

[18] Tim Salimans and Diederik P. Kingma. Weight normalization: A simple reparameterization to accelerate training of deep neural networks, 2016.

[19] Jedrzej Kozal, Jan Wasilewski, Bartosz Krawczyk, and Michał Woźniak. Continual learning with weight interpolation, 2024.

[20] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[21] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library, 2019.

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

[23] Delft AI Cluster (DAIC). The delft ai cluster (daic), rrid:scr_025091, 2024.

[24] Yuzhe Yang, Guo Zhang, Zhi Xu, and Dina Katabi. Harnessing structures for value-based planning and reinforcement learning, 2020.

[25] Alex Krizhevsky. Learning multiple layers of features from tiny images. pages 32–33, 2009.

[26] Sylvestre-Alvise Rebuffi, Alexander Kolesnikov, Georg Sperl, and Christoph H. Lampert. icarl: Incremental classifier and representation learning, 2017.

A Hyperparameter Tuning Results

This section presents results from hyperparameter tuning experiments for the Noise injection method on the Permuted MNIST benchmark. It includes comparisons between different noise injection variants and explores the effects of varying the standard deviation σ and the scaling factor λ.

A.1 Noise Injection Methods

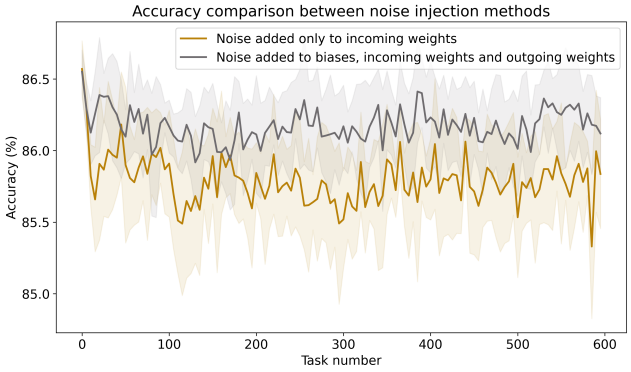

Figure 6: Overall accuracy comparison between two Noise injection methods: injecting noise to only incoming weights and injecting noise to incoming weights, outgoing weights, and biases

A.2 Effect of Standard Deviation on Overall Accuracy, Approximate Rank, and Percentage of Dead Neurons

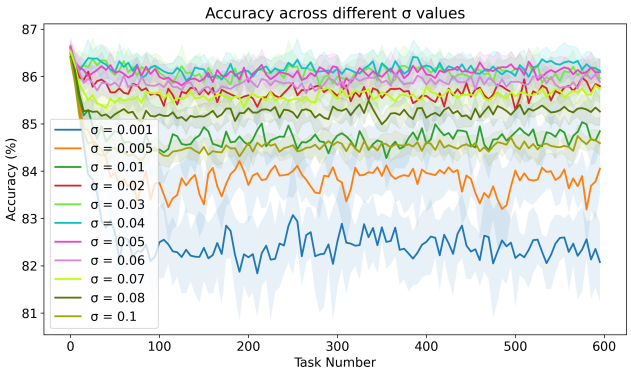

Figure 7: Overall accuracy across different noise standard deviations σ

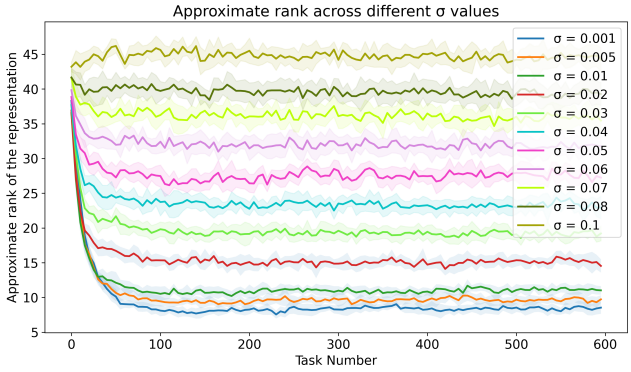

Figure 8: Approximate rank across different standard deviations σ in Noise injection

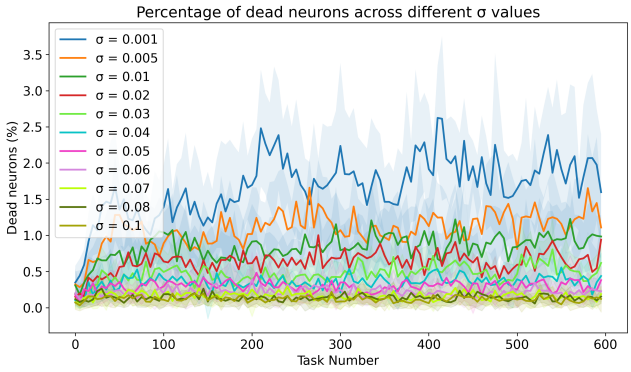

Figure 9: Percentage of dead neurons across different standard deviations σ in Noise injection

A.3 Effect of Lambda on Overall Accuracy, Approximate Rank, and Percentage of Dead Neurons

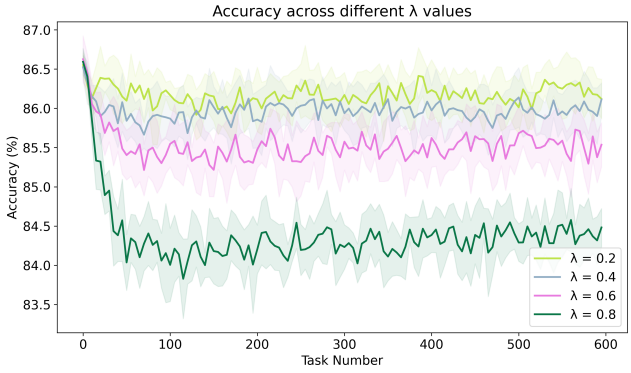

Figure 10: Overall accuracy across different λ values in Noise injection

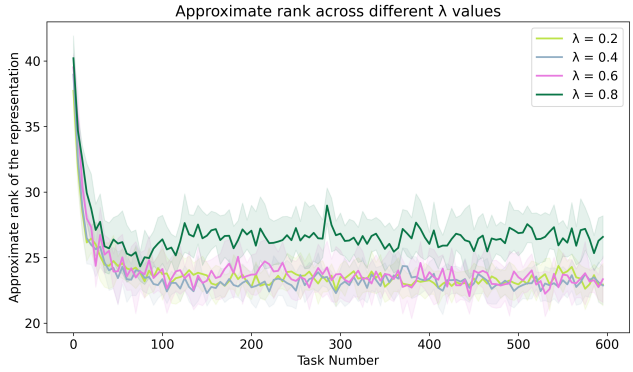

Figure 11: Approximate rank across different λ values in Noise injection

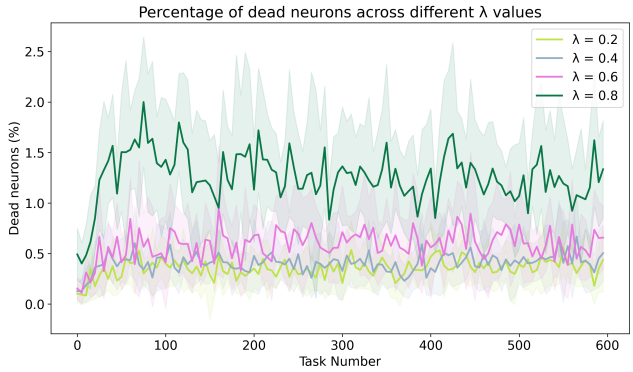

Figure 12: Percentage of dead neurons across different λ values in Noise injection

B Additional Experimental Results on the Permuted MNIST Benchmark

This section presents further experimental results on the Permuted MNIST benchmark that were not included in the main text.

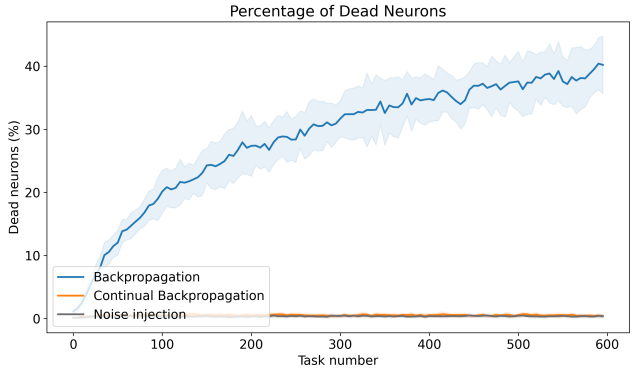

Figure 13: Percentage of dead neurons of the Noise injection method compared to Continual Backpropagation and Backpropagation

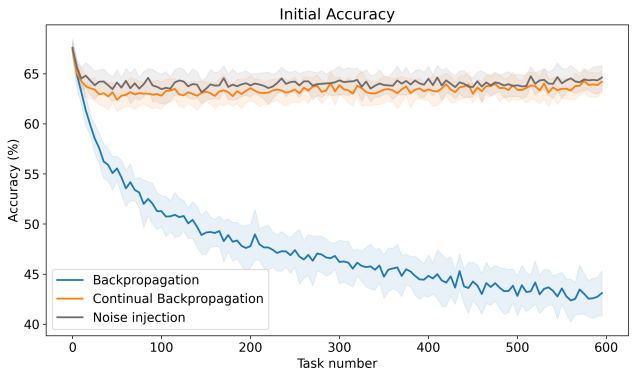

Figure 14: Initial accuracy of the Noise injection method compared to Continual Backpropagation and Backpropagation

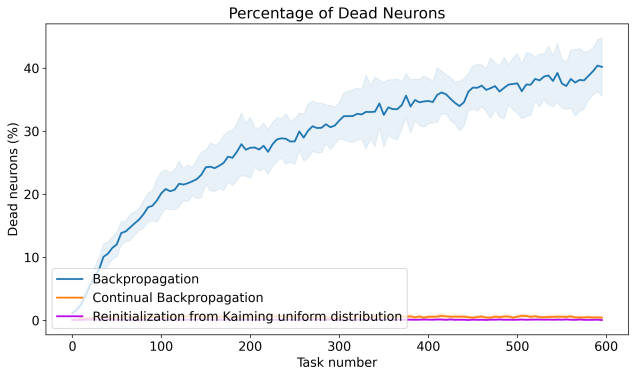

Figure 15: Percentage of dead neurons of the Reinitialization from Kaiming uniform distribution method compared to Continual Backpropagation and Backpropagation

C Class-Incremental CIFAR Results

This appendix presents the evaluation of the proposed neuron reinitialization strategies on the more challenging Incremental CIFAR benchmark, including a description of the benchmark, the evaluation setup, and the corresponding results.

C.1 Incremental CIFAR Benchmark

To further assess the generalization of our proposed methods, we evaluate them on the Incremental CIFAR benchmark, derived from the CIFAR-100 dataset [25]. Compared to Permuted MNIST, CIFAR-100 features higher-dimensional input and more complex visual patterns, making it a significantly more challenging continual learning scenario [26].

Following prior work [12], the dataset is split into a sequence of tasks, where each task introduces a disjoint subset of new classes. In the default configuration, five new classes are introduced every 200 epochs, resulting in 100 classes in total. In our setup, we run only the first half of the experiment (introducing 50 classes in total), which is sufficient to observe performance trends while also reducing computational demands. Samples are presented in a continual learning setting, without explicit task boundaries.

A ResNet-18 architecture with ReLU activations and batch normalization is used for all experiments, consistent with Dohare et al. research [12]. The test accuracy is computed after every epoch using held-out examples from all classes the model has encountered so far.

All experiments, including hyperparameter tuning and final evaluations, were repeated 5 times with different random seeds. The presented line graphs are smoothed by averaging every five steps, and the shaded areas represent the standard deviation across runs.

C.2 Hyperparameter Configuration

For all experiments, the best reported Continual Backpropagation (CBP) hyperparameters in Dohare et al. research [12] were used: a replacement rate of 0.00001 and a maturity threshold of 1000. A learning rate schedule is applied at the start of each new task increment:

- Epochs 0–60: learning rate = 0.1
- Epochs 60–120: learning rate = 0.02
- Epochs 120–160: learning rate = 0.004
- Epochs 160–200: learning rate = 0.0008

This schedule helps to achieve more stable convergence and preserve previously learned knowledge.

Training is performed using SGD with a momentum of 0.9, weight decay of 0.0005, and mini-batch size of 90.

Among proposed alternative methods, only the top two performers from the Permuted MNIST benchmark are evaluated due to time and computational constraints: **Noise injection** and **Reinitialization from Kaiming uniform distribution**. For Noise injection, a brief hyperparameter search is performed over four values of λ (0.2, 0.4, 0.6, 0.8) and three values of standard deviation σ (0.02, 0.04, 0.06). The results of this tuning process are shown in Figures 16 and 17.

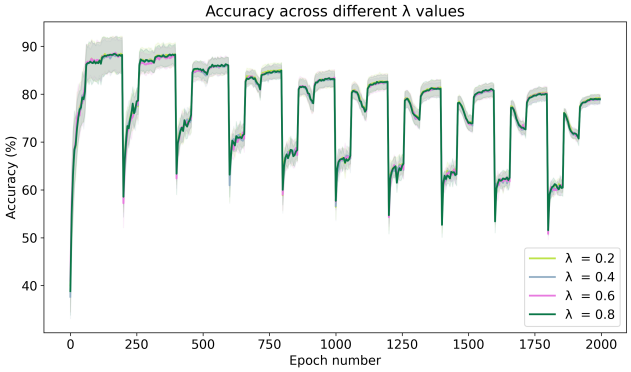

Figure 16: Test accuracy across different values of λ in Noise Injection

Interestingly, all tested combinations of λ and σ yield very similar final test accuracy. Therefore for final evaluations, we use the same setting as in the Permuted MNIST benchmark ($\lambda = 0.2$ and $\sigma = 0.04$).

These hyperparameter tuning results suggest that the performance of Noise injection on the Incremental CIFAR-100

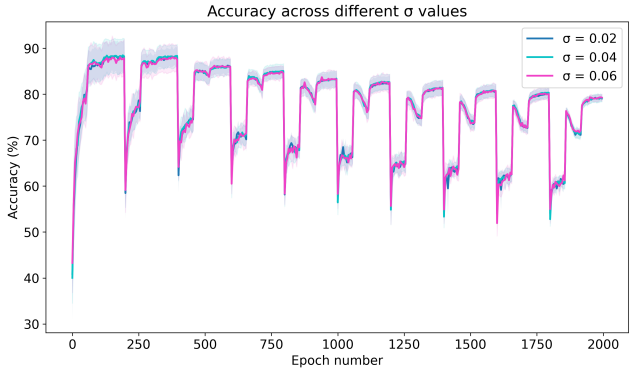

Figure 17: Test accuracy across different noise standard deviations σ in Noise Injection

benchmark is relatively robust to hyperparameter variations within the explored range.

C.3 Results and Discussion

Figure 18 shows that, compared to CBP, Noise Injection maintains similarly high test accuracy on the Incremental CIFAR benchmark, consistent with observations from the Permuted MNIST experiments.

As for the Reinitialization from Kaiming uniform distribution method, it performs well at the beginning of the task and reaches a similar accuracy to CBP. However, its performance slightly declines toward the end of the task. This drop suggests that certain CBP hyperparameters, such as the learning rate schedule, may not be optimal for this method. Hyperparameter tuning could potentially stabilize performance and lead to better results.

Overall, these findings suggest that the proposed alternative methods may generalize across different continual learning benchmarks. While preliminary results on Incremental CIFAR support the potential robustness of these strategies, a more comprehensive evaluation involving broader datasets and additional tuning is needed to confirm generalization. A full investigation of this is left for future work and is beyond the scope of the current study.

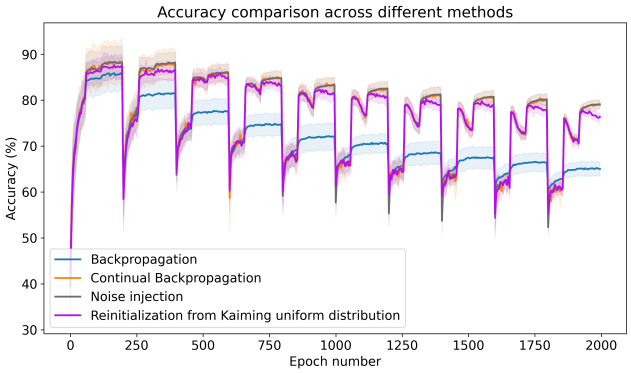

Figure 18: Test accuracy comparison between Continual Backpropagation, Backpropagation, Noise injection, and Reinitialization from Kaiming uniform distribution

D Example Prompts for Language Model Assistance

This appendix provides example prompts used when interacting with the language model. These examples illustrate the type of language-related assistance requested. In some cases, slight variations of these prompts may have been used. Importantly, the responses from the model were not copied directly, but rather suggestions were carefully reviewed and integrated with existing text as appropriate. Since the prompts were applied across multiple different parts of the text, specific before and after examples are not provided.

- Make this sentence clearer and syntactically correct: "..."
- Fix grammar errors: "..."
- Improve the flow and readability of this text: "..."