Reaching for Resilience: Understanding How Optimizers Affect the Stability Gap in Continual Learning

Chris Obis[1]

Supervisor(s): Tom Viering[1]**, Gido M. van de Ven**[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: Chris Obis
Final project course: CSE3000 Research Project
Thesis committee: Tom Viering, Gido M. van de Ven, Alan Hanjalic

Abstract

In the context of continual learning, recent work has identified a significant and recurring performance drop, followed by a gradual recovery, upon the introduction of a new task. This phenomenon is referred to as the *stability gap*. Investigating it and the potential solutions is essential, as such findings can reduce both the energy consumption and computational time required to prepare a high-performing agent. Given the strong influence of training procedures on model performance and stability, we analyze how various optimizers –*SGD, NAG, AdaGrad, RMSprop, Adam*– and momentum values affect the stability gap. We expose a deep neural network to a sequence of digit-identification tasks with varying rotations, and track several metrics to capture the components of the stability gap and the overall performance. Our results reveal that increasing momentum amplifies the steepness and depth of the gap, while shortening its duration. Within this simplified setup, *RMSprop* proves most effective in reducing the magnitude and duration of the drop while maintaining high overall performance.

1 Introduction

The concept of *continual learning* arises in both natural and artificial intelligence settings, where an agent undergoes a knowledge accumulation process that shapes its performance to solve pattern recognition problems. The adaptation mechanisms that facilitate continual learning are inherently complex in both settings. They combine information capturing strategies with effective knowledge integration processes that are designed to reduce interference with existing memory.

In the sphere of deep neural networks, several gradient-based optimization techniques (Choi et al., 2019) have been shown to effectively navigate the parameter space and achieve rapid convergence to high performance (Krizhevsky et al., 2012). To the best of our knowledge, most studies on the efficiency of these methods in reaching high accuracy are conditioned by a static nature of the input data distribution. In this sense, continual or lifelong learning changes the paradigm by introducing a temporal variable that influences the range and shape of the training data distribution. Consequently, the model is tasked with adapting to a multitude of scenarios or tasks, changing its inner configuration according to a joint objective.

Potential interference with previously learned tasks has been shown to pose a real challenge when training neural network models, manifesting as the phenomenon known as *catastrophic forgetting* (McCloskey and Cohen, 1989; Ratcliff, 1990). By simply introducing new tasks, parameter updates can drastically overwrite valuable past states that perform well on previous inputs. To prevent this, several strategies that use the concepts of replay, also called data rehearsal, (Robins, 1995; Rolnick et al., 2019) and regularization (Kirkpatrick et al., 2017; Li and Hoiem, 2017) have managed to substantially mitigate regressive behavior. This is achieved by reintroducing representatives of past tasks and adjusting the shape of the loss function, respectively. *Replay* is based on retraining the model on buffered input samples or task-specific data generators that approximate the underlying input distributions. In continual learning, *regularization* techniques help balance knowledge acquisition and retention by protecting weights that are relevant to past tasks from changing, while adapting to new incoming information. Consequently, these techniques require longer training periods and significantly more resources to improve performance.

A clear contrast emerges from these previous studies. On the one hand, humans have an innate ability to incrementally and effectively acquire new skills while distinguishing the scenarios in which they become applicable (Flesch et al., 2018; Kudithipudi et al., 2022). On the other hand, machine learning systems face a struggle when it comes to simultaneously capturing and retaining knowledge, even with state-of-the-art techniques (van de Ven et al., 2014). In the realm of artificial model training, this is known as the *stability-plasticity dilemma* (Grossberg, 1982).

1.1 Related Work and Contributions

This ablation study follows from the findings of Lange et al. (2023), who have set up a series of incremental learning scenarios (van de Ven et al., 2022) and have observed a recurring pattern of temporary forgetting. At the time of transition between training on specific tasks, they could observe a decreasing trend of performance, followed by a gradual recovery phase. They have assigned the term *stability gap* to this novel observation and highlighted how the similarity of consecutive tasks shapes its magnitude. The decision to employ a low evaluation periodicity for proper performance analysis has proven crucial in uncovering these subtle events. Such common training occurrences often go undetected in standard testing settings, as evaluation is performed at the end of particular stages. Figure 1 gives a visual representation of the described phenomenon, which arises when the training focus is switched to a new task.

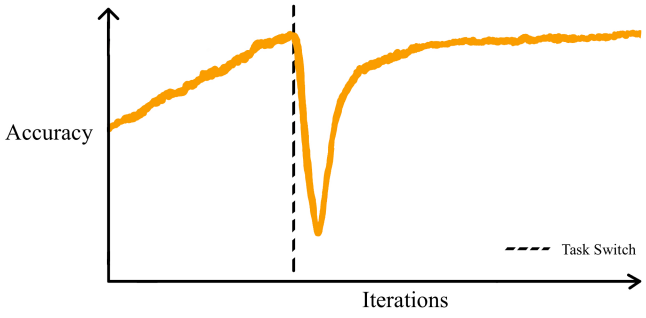


Figure 1: The *Stability Gap* phenomenon exhibited by models at task transitions, as theorized by Lange et al. (2023).

Building on Lange et al. (2023)'s identification and conceptualization of the stability gap, Hess et al. (2024) have continued to demonstrate its persistence in incremental-joint training, also called perfect-replay. In these cases, both the new and the old data are available, meaning that the ideal joint loss on them can be effectively minimized. Their results disprove the hypothesis that this gap could be the consequence of an imprecise approximation of the optimal joint loss shape. This shifts our focus from *what* needs to be optimized to *how* to optimize our trajectory to the target parameter configuration.

We advocate for the necessity of further investigation in this scope, as lowering worst-case accuracy of models, temporary as it may be, and with proven recovery potential, can pose a major and exploitable risk. This is particularly relevant in safety-critical systems and applications that require high reliability. Furthermore, the stability gap can also be perceived as a symptom of inefficient resource consumption.

The contributions of our work revolve around the generation and comparison of several experimental runs that highlight the influence of various industry-standard optimizers on the shape of the stability gap. We choose *SGD with momentum*, *NAG with momentum*, *AdaGrad*, *RMSprop* and *Adam* as our subjects. Various pre-established and newly proposed quantitative metrics are recorded to capture both the stability gap and the general stability-plasticity context. These are complemented by visualizations of the experiments. Our aim is to study and understand the underlying optimization dynamics that shape the stability gap, while contributing to the growing body of research on training efficiency.

For this purpose, we decide to proceed with a *domain-incremental learning* scenario. *Domain-incremental learning* refers to a continual learning process in which consecutive tasks define the same problem but differ in context (van de Ven et al., 2022). We require a setting that can accommodate our study objective. Therefore, our experimental work is based on a code base that was originally developed to support previous studies on mitigating catastrophic forgetting in continual learning (van de Ven and Tolias, 2019; van de Ven et al., 2022).

1.2 Research Questions and Findings

We benchmark the performance of a model undergoing a continual learning process under various optimization configurations to answer our proposed research question: "What is the impact of momentum and different optimizer choices on the stability gap of deep neural networks in continual learning problems?", alongside its sub-questions:

SQ1. How does increasing the **momentum** of the *SGD* and *NAG* optimizers impact the depth and duration of the *stability gap* under optimal hyperparameters?

SQ2. How do **adaptive learning rate** optimizers (*AdaGrad*, *RMSprop*, *Adam*) compare in terms of the *stability gap* shape under optimal hyperparameters?

Following the execution of our experimental runs, we have identified some performance trends that can be linked to optimization-specific components. In the cases of momentum-based optimizers (*SGD* and *NAG*), the amplitude of the stability gap tends to increase with momentum. At the same time, both the decrease and recovery phases become more sudden, leading to a steeper drop and a reduced duration. Ultimately, higher momentum values accelerate convergence at higher overall accuracy, but this comes at the cost of a more pronounced drop during task transitions.

Adam presents a similar stability gap shape as the previous two optimizers, characterized by a large amplitude and increased volatility. The purely adaptive nature of both *AdaGrad* and *RMSprop* attenuates the drop and reduces instability when switching tasks. *AdaGrad* experiences a slower performance recovery and, implicitly, a wider stability gap, compared to *Adam* and *RMSprop*.

1.3 Overview

The remainder of this work is structured as follows. **Section 2** describes the continual learning scenario, the fundamentals of each optimizer, and the measurements recorded for our quantitative analysis. **Section 3** defines the technical aspects of our experimental process. **Section 4** presents a quantitative and qualitative analysis of our findings, discussing the registered trends. **Section 5** summarizes these and explains the causal relationship between each optimization strategy and its observed behavior. **Section 6** revisits our main findings, addresses the questions and hypotheses, and suggests directions for future related work. Finally, **Section 7** discusses the reproducibility and ethical considerations of our research.

2 Methodology

Subsection 2.1 gives the context of the proposed continual learning environment. **Subsection 2.2** describes the fundamentals of the chosen optimizers, while **subsection 2.3** defines several hypotheses based on these details. Lastly, **Subsection 2.4** addresses the measurement methods used to capture the stability gap and the overall performance.

2.1 Continual Learning Setting

In our domain-incremental learning study, we adopt a problem in which task switches are induced by applying a fixed transformation to the input distribution, while the output space remains constant.

To formulate this, we define a sequence of tasks $\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_N$, where the first one is given by a base dataset, and every task instance $\mathcal{T}_i$ is generated by rotating the dataset corresponding to its predecessor $\mathcal{T}_{i-1}$.

Similarly to the experiments conducted by Lange et al. (2023), we use the MNIST dataset (LeCun et al., 1998) as our base. Therefore, the goal consists of identifying handwritten grayscale digits. This simplified setting allows the model to reach high performance and exhibit the stability gap phenomenon. We measure accuracy as the proportion of correctly classified digits in one evaluation phase, computed per task from the moment it is introduced to the model. Task-specific training samples are shown in Figure 2. We opt for a four-task setup with rotations $\phi \in \{0°, 50°, 100°, 150°\}$, respectively.

To alleviate forgetting, we retrain the model according to the *incremental joint training* method. According to it, the model is continuously exposed to all previously seen contexts (van de Ven et al., 2022).

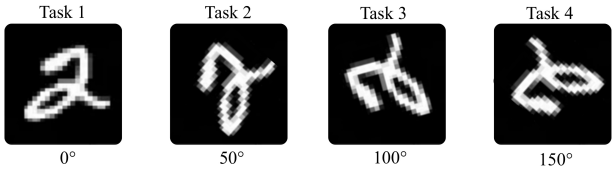


Figure 2: Task-specific training samples, with rotational degrees applied. To avoid purely situational misjudgments in distinguishing 6 from 9, rotations do not exceed 180°.

We aim to confirm the existence and measure the magnitude of the stability gap even in this simplified and universally optimal setting.

2.2 Optimization Strategies

Our experiments involve five different first-order gradient-based optimizers. We are going to dive into their specifics in chronological order of their development. They have all benefited and improved on the contributions of their predecessors. We refer to **Appendix A.1** for the detailed mathematical equations that underlie each optimizer referenced in this study.

Stochastic Gradient Descent with Momentum. The first optimizer (Polyak, 1964) of the series pioneered a "velocity" component, which is an accumulation of past gradients. This helps guide the search of the parameter space in directions of consistent descent and minimized fluctuations, according to the loss functions. The momentum coefficient μ controls the impact of historical updates on the next change of weights. The higher the momentum, the more influence the velocity of the past has on the next step. An analogy can be made here with a ball rolling down a hill and accumulating inertia, thus overcoming small bumps, in our context local minima, more easily.

Nesterov Accelerated Gradient (NAG). The next optimizer, *NAG* (Nesterov, 1983), is a close variant of *SGD with momentum* that aims to indicate more informed gradient trajectories, by computing them at look-ahead positions. In doing so, it calculates the gradient after the velocity component is applied. The goal is to indicate the next trajectory after predicting where the accumulated inertia leads. Figure 3 shows a side-by-side comparison of how the two optimizers determine their update step.

Adaptive Gradient Algorithm (AdaGrad). Theorized by Duchi et al. (2011), *AdaGrad* was among the first algorithms to introduce the concept of per-parameter adaptive learning rates. Essentially, it maintains a sum of squared gradients that, inversely proportional to its value, determines the volatility of each parameter. This results in smaller changes to parameters with frequent or large gradients, and larger ones for those with infrequent or smaller gradients. However, a limitation of this approach is the monotonically decreasing trend of the learning rates, which can reduce the adaptability in later stages of the learning process.

Root Mean Square Propagation (RMSprop). Building upon *AdaGrad*'s contributions, *RMSprop* (Tieleman and Hinton, 2012) addresses the irreversibly decreasing learning rate schedule. It does so by adopting an exponentially decaying

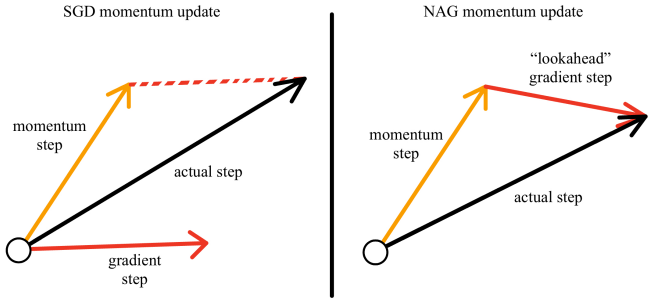


Figure 3: *SGD* and *NAG* momentum update comparison. *NAG* includes the momentum-based velocity in calculating the gradient. Visualization inspired from Li et al. (2025).

average approach to accumulating past gradients, rather than summing them. Ultimately, this leads to positive and negative oscillations in the step sizes.

Adaptive Moment Estimation (Adam). The final and most recently developed among all presented optimizer choices, *Adam* (Kingma and Ba, 2015) combines the features of momentum and dynamically scaled learning rates. It uses two gradient moments and includes bias correction to account for initialization at zero.

2.3 Hypotheses

We formulate the following hypotheses and run our experiments to confirm or disprove their validity.

H1. Higher momentum values in *SGD* and *NAG* lead to more instability when the input domain changes, increasing the steepness and depth of the stability gap.

H2. Higher momentum values in *SGD* and *NAG* lead to faster accumulation of joint knowledge, resulting in a quicker and earlier recovery.

H3. The adaptive step size of *AdaGrad* and *RMSprop* determines a more gradual and limited performance decrease.

H4. We project *Adam* to outperform the other optimizers in most stability gap and general metric categories, due to its hybrid nature.

2.4 Quantifying the Stability Gap

The following section discusses the methods used to analyze key trends of the accuracy curves under evaluation. There is a close connection between our experimental starting point and goal, and those of the domain-incremental learning exercise presented in the work of Lange et al. (2023). Therefore, the measurements recorded include a series of approaches first enunciated by Lange et al. (2023), which give a more general stability-plasticity context. Several novel ways to capture the shape of the gap are also proposed.

Pre-established metrics. We report the **Average Accuracy (ACC)**, a standard metric in continual learning, as the average performance of the model across all tasks after the last training iteration. The **Average Forgetting (FORG)** coefficient is defined as the average difference between the model's final accuracy on each task and its accuracy immediately after training on that task was completed. The **Average Minimum Accuracy (min-ACC)** is calculated as the average

absolute minimum accuracy achieved by the model across all tasks after they have been learned. This is a worst-case indicator of knowledge retention.

We refer to the study by Lange et al. (2023) for computational equations and an in-depth discussion on these preliminary metrics.

Proposed metrics. In addition to the previous metrics, we define the following novel ways to capture the components of the stability gap. All of them operate on the same task- and window-based averaging principle, where windows correspond to the introduction of a new task. This is based on intuitive and empirical proof that the stability gap occurs when transitioning between tasks. Therefore, every metric is an average across the model's performance on all tasks, where each task-specific component is the average across all fixed-size windows post-training on that specific one. Naturally, the metrics differ in terms of the inner-window measurement. We will refer to this operation as *Window-Task Averaging (WTA)* for simplicity. We include Figure 4 to illustrate these metrics.

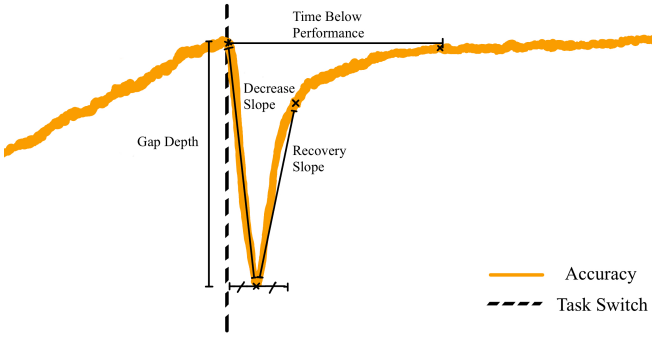


Figure 4: Visualization of the proposed metrics, applied to a stability gap. These are represented by **Gap Depth, Time Below Performance, Decrease Slope** and **Recovery Slope**. The highlighted segments at the bottom of the drop illustrate the symmetry used in our calculations for the **Recovery Slope**.

Time Below Performance (TBP) quantifies the average duration, expressed in iterations, that the model's accuracy remains below the threshold set at the end of the previous training window. From now on, we will call this the *pre-window* value for conciseness. We compute and average the task- and window-specific recovery times of the model according to the *WTA* method.

Equation 1 defines the computation of **TBP**, where N is the total number of tasks and W_i is the number of windows corresponding to task i. $t^{i,j}_{recovery}$ represents the first iteration number within a window j at which the model's performance on task i recovers to the pre-window value.

$$TBP = \frac{1}{N}\sum_{i=1}^{N}\frac{1}{W_i}\sum_{j=1}^{W_i} t^{i,j}_{recovery} \tag{1}$$

We propose the **Decrease Slope (DS)** to measure the rate of degradation experienced by the model in response to the introduction of new tasks. This is an indication of the percentages of accuracy lost per iteration during the declining phase. We incorporate the same *WTA* method to average the slopes between the pre-window and minimum performance points.

Equation 2 applies the computational idea, where $t^{i,j}_{\text{min}}$ and $t^{i,j}_{\text{pre-window}}$ are the iteration numbers corresponding to the pre-window performance $P^{i,j}_{\text{pre-window}}$ and the minimum inner-window performance $P^{i,j}_{\text{min}}$, respectively. The rest of the terms convey the same information as previously.

$$\text{DS} = \frac{1}{N}\sum_{i=1}^{N}\frac{1}{W_i}\sum_{j=1}^{W_i}\frac{P^{i,j}_{\text{min}} - P^{i,j}_{\text{pre-window}}}{t^{i,j}_{\text{min}} - t^{i,j}_{\text{pre-window}}} \tag{2}$$

We define the **Recovery Slope (RS)** to capture the initial sharpness of the ascending trend, mirroring **DS**. We decide to calculate this from the inner-window minimum performance point and the point that occurs x iterations past this minimum. x represents the inner-window iteration number of the minimum. This symmetry ensures less sensitivity of the metric to recovery plateaus while being a solid indicator of the immediate recovery trajectory. The same *WTA* principle is respected.

In case the model recovers to the pre-window performance before the fixed recovery range x, the slope is capped at the actual recovery index. We make this computational choice to avoid underestimating the speed of recovery. A maximum iteration range is also set to the length of the window to ensure robustness.

Equation 3 covers the three potential situations. In the first case, $P^{i,j}_{\text{recovery}}$ is the model's performance on task i in window j at iteration $t^{i,j}_{\text{recovery}}$, given that the pre-window accuracy is reached sooner than x iterations away from the minimum. In the second case, if continuing for x iterations past the minimum point exceeds the window range, $P^{i,j}_{\text{final}}$ gives the performance on task i at the final iteration $t^{i,j}_{\text{final}}$ of window j. In the third scenario, $P^{i,j}_{\text{min}+x}$ represents the performance recorded x iterations past the minimum accuracy point. The interpretation of the rest of the terms remains unchanged.

$$\text{RS} = \frac{1}{N}\sum_{i=1}^{N}\frac{1}{W_i}\sum_{j=1}^{W_i}\begin{cases}\frac{P^{i,j}_{\text{recovery}} - P^{i,j}_{\text{min}}}{t^{i,j}_{\text{recovery}} - t^{i,j}_{\text{min}}} & \textit{if recovery is before } x \textit{ iterations}\\ \frac{P^{i,j}_{\text{final}} - P^{i,j}_{\text{min}}}{t^{i,j}_{\text{final}} - t^{i,j}_{\text{min}}} & \textit{if } x \textit{ iterations exceed the window}\\ \frac{P^{i,j}_{\text{min}+x} - P^{i,j}_{\text{min}}}{x} & \textit{otherwise}\end{cases} \tag{3}$$

The final measurement reported is the **Gap Depth (GD)**, which quantifies the magnitude of the accuracy drop during task switches. This is computed as the *WTA* of the maximum accuracy decrease within a window, with the pre-window value as our reference point. Note that the minimum accuracy point is the same as in **DS**.

The novelty of equation 4 consists of applying the *WTA* averaging scheme to the difference between $P^{i,j}_{\text{pre-window}}$ and $P^{i,j}_{\text{min}}$. All the pre-discussed terms express the same concepts.

$$GD = \frac{1}{N}\sum_{i=1}^{N}\frac{1}{W_i}\sum_{j=1}^{W_i}\left(P^{i,j}_{\text{pre-window}} - P^{i,j}_{\text{min}}\right) \tag{4}$$

Additional practices. Even with a high number of test samples, which generally improves accuracy estimates, we have to take into account anomalous data points. These could significantly distort the accurate detection of real minimum and recovery points. To ensure robustness, we consider using a smoothed version of the accuracy curves. Both authentic and noise-induced values are hereby replaced by windowed averages. Following the arguments of Lange et al. (2023), we compute the inspired metrics using the original data, as we are particularly interested in the worst-case behavior.

3 Experimental Setup

This section details the environment set up to facilitate experimentation with different optimization configurations, visualizing performance, and quantifying trends. We address the general model architecture, training regime, optimizer, and evaluation details. The experimental scripts use the PyTorch v2.7.0 framework (Paszke et al., 2019) with Python v3.10.4 and executed on a 2024 MacBook Air equipped with an Apple M3 chip, as our computational hardware. Moreover, a forked version of the original repository[2] from van de Ven et al. (2022) has been adapted to suit our experimental needs.

Model Architecture. A fully connected neural network with 3 hidden layers, each consisting of 400 ReLU-activated neurons, is utilized as our model across all experiments. The input layer processes flattened 28x28 (784-dimensional input) images sampled from the Rotated-MNIST dataset (LeCun et al., 1998), while the output layer is composed of 10 units corresponding to the classification labels (0-9). The initial weights are randomly assigned using the Kaiming uniform distribution scheme, as described by He et al. (2015), which is the default of PyTorch for layers using ReLU.

Training Regime. The model is trained according to the mini-batch learning approach, while an incremental joint training scheme is used for retraining on previous data. Consequently, introducing a new task implies training on a shuffled concatenation of the sets corresponding to all the contexts seen so far. To account for a growing and more diverse training set, we are scaling the size of the mini-batches at every moment a new task is introduced, starting with and increasing it by 128. A task-specific training phase runs for a fixed number of 500 iterations. The model is trained using the standard cross-entropy loss function.

Optimizers and Hyperparameters. A gridsearch procedure, detailed in **Appendix A.3**, is run over different hyperparameters for every optimizer choice (*SGD* and *NAG* with multiple *momentum* values, *AdaGrad*, *RMSprop*, *Adam*) to identify optimal experimental configurations.

Table 1 reports the optimal values found that will be used in our benchmarks. For completeness, we also include the momentum values under evaluation in the cases of *SGD* and *NAG*.

[2]**https://github.com/GMvandeVen/continual-learning**

Optimizer	Learning Rate	Additional Parameters
SGD	0.1	$\mu \in \{0, 0.3, 0.5, 0.7, 0.9\}$
NAG	0.1	$\mu \in \{0.3, 0.5, 0.7, 0.9\}$
AdaGrad	0.01	–
RMSprop	0.001	$\alpha = 0.9$
Adam	0.001	$\beta_1 = 0.9, \beta_2 = 0.99$

Table 1: Gridsearch hyperparameter results, according to **Appendix A.3**. Maintaining a fixed learning rate across the experiments of a momentum-based optimizer helps isolate the effect of momentum. The unspecified hyperparameters take PyTorch's default values.

Evaluation. As argued by Lange et al. (2023), we employ an evaluation periodicity ρ_{eval} of 1 to understand the full picture of performance. At the end of every iteration, the model is evaluated on a number of 2000 test samples per previously encountered evaluation task, recording the accuracy.

To enhance statistical reliability, we study each optimization configuration in 10 randomly seeded experimental runs. The **ACC**, **FORG**, **min-ACC**, **TBP**, **GD**, **DS**, and **RS** coefficients are computed for each run. Following this, a new aggregated entry is reported, which contains the sample mean, accompanied by the standard error calculated across the 10 simulations. We use a uniform filter with a window length of 5 to smoothen the accuracy curves used for **TBP**, **GD**, **SD** and **SR**. In building our qualitative analysis of the stability gap shape and its context, we plot the mean performance curves and shade the uncertainty region described by the standard error.

4 Experimental Results

The following section is divided into two parts, each dealing with different aspects of the experimental results. Both parts are based on the unified Table 6 of **Appendix A.2**, with optimizers grouped accordingly. The first part presents the measurement changes recorded as a result of different momentum values in the cases of *SGD* and *NAG*. The second part compares the results obtained using the more recently developed first-order optimization algorithms.

Momentum-based results. In terms of stability gap components, the results in Table 2 reveal consistent patterns linked to changes in momentum. Progressively increasing momentum in both *SGD* and *NAG* results in a monotonically decreasing **TBP** and **DS**, while **RS** increases. For example, **TBP** decreases from 208.8 to 166.2 for *SGD* and from 206.5 to 180.3 for *NAG*, indicating a narrow stability gap. At the same time, **RS** grows from 0.029 to 0.231 for *SGD* and from 0.044 to 0.168 for *NAG*, signaling an accelerated recovery. Similarly, a reduction in **DS** indicates a sharper performance drop. In both cases, **GD** remains stable for the lower momentum values, but presents a record high 5.02 for *SGD* and 3.56 for *NAG* when μ is set to 0.9, suggesting greater instability. Both optimizers present similar performances, though *NAG* manages to produce lower oscillations at higher momentum.

As shown in Table 3, momentum-based trends can also be observed for the general quantification of stability and

Metric	Trend $\mu \uparrow$	*SGD* ($0.3\mu \rightarrow 0.9\mu$)	*NAG* ($0.3\mu \rightarrow 0.9\mu$)
TBP (it.)	$\downarrow$	$208.8 \rightarrow 166.2$	$206.5 \rightarrow 180.3$
GD (%)	Peaks at 0.9μ	$2.33 \rightarrow 5.02$	$2.49 \rightarrow 3.56$
DS	$\downarrow$	$-0.092 \rightarrow -0.384$	$-0.068 \rightarrow -0.309$
RS	$\uparrow$	$0.029 \rightarrow 0.231$	$0.044 \rightarrow 0.168$

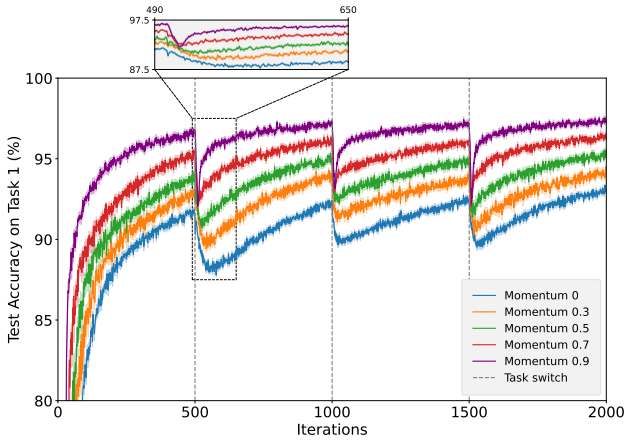
Table 2: Stability gap trends as momentum increases in *SGD* and *NAG*. We report the sample mean across 10 independent runs. **TBP** and **DS** decrease, while **RS** increases. **GD** spikes at higher momentum. These indicate that the duration of the stability gap becomes shorter, the decrease and recovery phases are accelerated, while the performance drop also increases. *NAG* presents higher stability compared to *SGD* with higher momentum. We consider the same momentum interval for a reasonable comparison.

Metric	Trend $\mu \uparrow$	*SGD* ($0\mu \rightarrow 0.9\mu$)	*NAG* ($0.3\mu \rightarrow 0.9\mu$)
ACC (%)	$\uparrow$	$93.49 \rightarrow 96.58$	$93.44 \rightarrow 96.82$
FORG (%)	$\uparrow$	$-1.47 \rightarrow -0.62$	$-1.4 \rightarrow -0.67$
min-ACC (%)	$\uparrow$	$89.25 \rightarrow 90.12^{*}$	$89.1 \rightarrow 91.92$

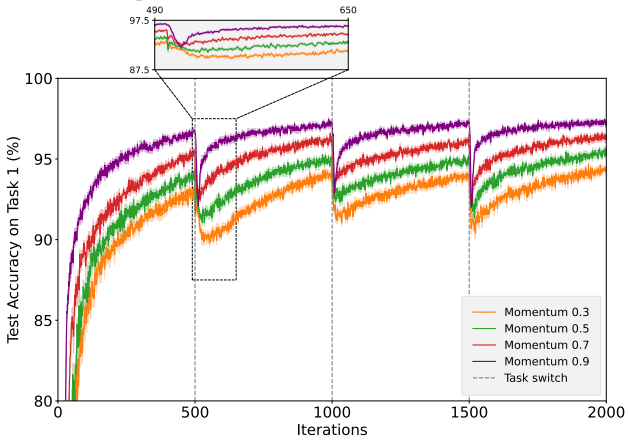
Table 3: Stability-plasticity trends as momentum increases in *SGD* and *NAG*. Each entry holds the sample mean across 10 independent runs. **ACC**, **FORG** and **min-ACC** all increase with momentum. This indicates higher overall performance, slightly more forgetting proneness and an improved worst-case behavior. An * symbol is placed to signal a higher value recorded with 0.7μ.

plasticity indicators. Following the same momentum rise in both scenarios, **ACC** increases by 3.09% for *SGD* and 3.38% for *NAG*, indicating higher learning potential. Furthermore, **FORG** also increases, which can also be interpreted as a sign of a limited available space for further improvement. The incremental joint training scheme allows for significant knowledge gain after each task-specific training is completed, justifying the negative values recorded. Following the same trend, the worst-case **min-ACC** metric reports an improvement from 89.25 to 90.12 for *SGD* and from 89.1 to 91.92 for *NAG*. In the case of *SGD*, there is a dip from 91.14 to 90.12 when the momentum shifts from 0.7 to 0.9, which can be correlated to the large **GD** recorded in Table 2 for momentum 0.9. For *NAG*, this continues to improve.

Qualitatively, Figure 5 displays the performance comparisons of the *SGD*-optimized and *NAG*-optimized models, respectively, on task 1 with multiple momentum values. We observe that increasing the momentum in both cases tends to sharpen the performance drop and accelerate the recovery phase of the stability gap, narrowing it. The depth of the stability gap also increases with higher momentum values.



(a) *SGD* performance with different momentum values.



(b) *NAG* performance with different momentum values.

Figure 5: Performances of (a) *SGD* and (b) *NAG* on Task 1 under different momentum values. Each curve represents the sample mean across 10 randomly seeded runs with a shaded standard error region. Higher momentum values determine a steeper, deeper and shorter stability gap. Increasing the momentum also leads to greater final accuracy and faster convergence.

AdaGrad, RMSprop and Adam results. The quantification of the stability gap components for *AdaGrad*, *RMSprop* and *Adam* is presented in Table 4. This shows that *RMSprop* and *Adam* perform well in terms of **TBP**, with values of 134.1 and 150.1, respectively, while *AdaGrad* experiences a longer recovery, scoring 208.4. *Adam* presents a significantly greater **GD** compared to both counterparts. Alongside its lowest -0.296 **DS** and highest 0.191 **RS** among the three, these observations indicate the largest instability when transitioning. *AdaGrad* reports the smallest absolute slope values, describing a gradual and prolonged stability gap.

General metrics are reported in Table 5, with *Adam* achieving the highest **ACC** of 96.82, while *RMSprop*'s 96.69 coming just short of it. All three score comparably in **FORG**, with Adam's -0.83 slightly coming on top in terms of knowledge retention and gain. RMSprop scores the highest **min-ACC** values, of 93.61, supporting its best overall **GD**.

Metric	*AdaGrad*	*RMSprop*	*Adam*
TBP (it.) ↓	208.4 ± 7.9	**134.1** ± 8.0	150.1 ± 5.4
GD (%) ↓	1.68 ± 0.06	**1.4** ± 0.08	4.31 ± 0.13
DS	-0.084 ± 0.005	-0.141 ± 0.035	-0.296 ± 0.010
RS	0.031 ± 0.002	0.070 ± 0.017	0.191 ± 0.011

Table 4: Stability gap components of *AdaGrad*, *RMSprop* and *Adam*. The arrows indicate the order of superiority in performance. **DS** and **RS** are not succeeded by these because they do not directly reflect improved behavior. *RMSprop* reduces both **GD** and **TBP** the most, indicating a reduced drop and a quicker general recovery than its counterparts. *Adam* presents the largest absolute **DS** and **RS** values, implying increased abruptness of the gap.

Metric	*AdaGrad*	*RMSprop*	*Adam*
ACC (%) ↑	95.49 ± 0.03	96.69 ± 0.10	**96.82** ± 0.07
FORG (%) ↑	-0.36 ± 0.10	-0.74 ± 0.13	**-0.83** ± 0.14
min-ACC (%) ↑	92.61 ± 0.06	**93.61** ± 0.05	90.76 ± 0.17

Table 5: Stability-plasticity indicators for *AdaGrad*, *RMSprop* and *Adam*. Arrows define the performance superiority. *Adam* ranks superior in **ACC** and **FORG**, showcasing high learning potential, however the differences to *RMSprop* are not major. *RMSprop*'s highest **min-ACC** represents the best worst-case behavior amongst the three.

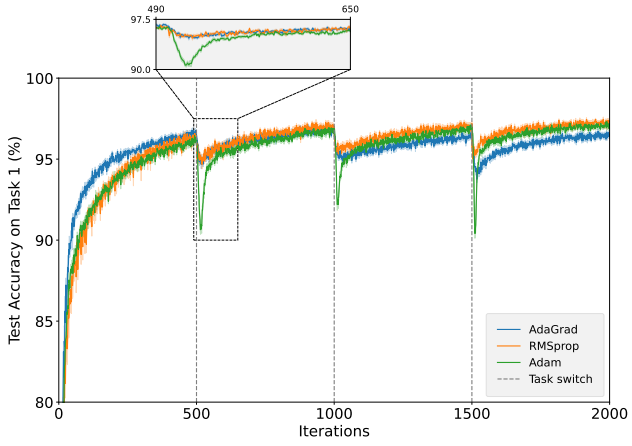


Figure 6: Performances of *AdaGrad*, *RMSprop* and *Adam* on Task 1. Each curve represents the sample mean across 10 randomly seeded runs with a shaded standard error region. *Adam* exhibits the largest stability gap. As new tasks are introduced, *AdaGrad* struggles the most to recover lost performance. *RMSprop* presents the shortest-lasting and lowest performance decrease.

Figure 6 provides a visual representation of the accuracy curves corresponding to the performance on task 1 of *AdaGrad*, *RMSprop* and *Adam*.

Appendix A.2 includes the individually plotted performances of the optimizers on all tasks. These include both momentum-based and adaptive learning rate strategies.

5 Discussion

The performance trends presented allow us to draw general conclusions and provide an interpretation of the key findings. We formulate our discussion by first looking at the stability gap implications of the strategies employed. Following this, we place these insights within the ultimate goal of continual learning: *reaching a high overall accuracy while minimizing inefficient resource consumption*.

Based on our previous quantitative and qualitative stability gap analyses, we can state that our momentum-based results confirm the following. Increasing the momentum coefficient in *SGD* and *NAG* leads to a more sudden loss of accuracy, followed by a faster recovery phase. In between these two stages, the stability gap also becomes progressively narrower with a rising momentum. According to our gathered data, after a certain momentum threshold, around 0.5μ, the magnitude of the drop also increases. It can also be stated that *NAG* manages to better attenuate the drop and reduce the gap's depth, in comparison with *SGD*. Our findings complement hypotheses **H1** and **H2**. In a broader context, as momentum increases, the learning curves in Figure 5 shift upward and leftward, indicating higher accuracy convergence.

The momentum component of both *SGD* and *NAG* is widely considered essential to efficiently train a high-performance model. We discover that, in our domain-incremental learning scenario, maximizing the benefits of this accelerated learning process comes at the expense of a steeper, deeper, although narrower stability gap.

We can reason that increasing momentum causes the parameters to pick up a joint update trajectory more quickly after a new task is introduced. This rapid performance change accelerates both the decrease and recovery stages. The model thus undergoes a temporary destabilization followed by a faster adaptation. The increased depth of the stability gap may result from overshooting optimal parameter configurations due to aggressive updates. At the same time, the ability to escape shallow local minima encountered is also improved, reflected in a higher and accelerated final convergence.

In terms of the behavior of adaptive learning rate optimizers, both *AdaGrad* and *RMSprop* report minor stability gap depths, confirming hypothesis **H3**. However, we can attribute *AdaGrad*'s longer-lasting performance recovery to its irreversibly decreasing learning rates. The same limitation of *AdaGrad* inhibts adaptation to new tasks, reflected in lower joint and worst-case accuracy scores. The flexibility of *RMSprop*'s exponentially decaying learning rates provides both long-term plasticity and efficient adaptation, scoring better than its counterpart in all aforementioned categories. On the other hand, *Adam*'s momentum-driven updates follow the behavior of the previous high-momentum optimizers, increasing the abruptness and amplitude of the gap. Similarly, this fast-paced update dynamic enables quick recovery and results in a high average accuracy across all tasks.

Considering all previous observations, we can declare *RMSprop* to be the superior choice in reducing the magnitude and duration of the stability gap in this simplified experimental setup, thereby disproving hypothesis **H4**. However, controlling the gap comes at the cost of a slightly lower final

accuracy than *Adam* and *NAG*.

Interestingly, Wilson et al. (2017) demonstrate that despite converging faster, adaptive methods (*AdaGrad, RMSprop* and *Adam*) generalize worse than SGD and its variants as training progresses. An explanation could be that they favor rare features, relevant in sparse contexts, but steer away from flatter minima, which are associated with better generalization. We therefore hypothesize that extending our continual learning context with new tasks can result in *SGD* and *NAG* with high momentum outperforming the other optimizers in terms of joint accuracy, worst-case, and stability gap behavior.

Limitations. Our ablation study aims to draw a suggestive picture of a forgetting-prone scenario, within reasonable experimental time and space complexity constraints. However, previous research on the topic of catastrophic forgetting confirms that increasing the capacity of the model tends to discourage general forgetting (Goodfellow et al., 2014). Although the insights gained are promising, they come as a result of a simplified experimental setup based on a single dataset that might not completely capture the complexity of real-world tasks.

6 Conclusions

The purpose of this research was to provide an analysis of the influence that various optimizers have on shaping a temporary forgetting trend that Lange et al. (2023) first identified and termed the *stability gap*. We developed and adopted a series of metrics to capture the slopes, size, and duration of the gap, as well as the general performance of the model. We highlighted the inevitability of the stability gap across all optimization strategies and closely observed the performance trends sparked by task switching.

Answering the questions. Following the experiments conducted, we can conclude that a higher momentum implies a steeper and more drastic drop, succeeded by a faster recovery phase. This pattern holds for both *SGD* and *NAG*, with *NAG*'s predictive updates making it more resilient in minimizing the loss. Our results confirm *AdaGrad*'s controlled drop and limited plasticity potential, while *Adam* presents the same volatile behavior as high-momentum *SGD* and *NAG*. *RMSprop* ranks first overall in mitigating the magnitude and duration of the drop while achieving a high joint accuracy.

Future exploration. While our comparative study provides key insights into how optimizers model the stability gap, the scalability and generalizability of the results require further exploration. In this regard, applying the strategies to more complex, real-world datasets, increasing the number of tasks, and switching the learning paradigm could strengthen our understanding of optimizers' behavior. Future work can also expand the comparison to upcoming optimization schemes. Additionally, whether the same optimizer-induced stability gap dynamics hold when the capacity and architecture of the model change represents a future research avenue.

In conclusion, our comparative research on the effect of optimizers on the stability gap contributes to the continual learning goal of acquiring information without forgetting. By understanding and optimizing the stability gap, we can significantly improve efficiency and robustness while minimizing training costs. These achievements could greatly benefit adaptive machine learning systems, such as autonomous robotics and personalized recommendation systems, which are becoming increasingly more prevalent today.

7 Responsible Research

This section aims to justify the adherence of our study to ethical standards and academic principles. This is done by exclusively using and crediting publicly available accredited materials, while transparently and integrally reporting the experimental methodology and the results obtained. We argue that these are in compliance with the Netherlands Code of Conduct for Research Integrity from 2018, and that our data handling respects the FAIR principles (Wilkinson et al., 2016).

Reproducibility. We ensure that the materials, data, and codebase underlying our research are **findable** and **accessible**. We base our training and testing processes on rotational variants of the MNIST dataset (LeCun et al., 1998), which is freely available for academic use. We upload the code to a public GitHub repository[3], an open-source environment, to ensure the reproducibility and verifiability of the entire processing pipeline. Our discussions also detail the provenance of the stability gap, the sources of optimization algorithms, and measurement procedures.

We argue that the presented dataset, tools, and numerical findings can be integrated with different analysis and processing systems, as well as replicated and combined in future work. This categorizes them as **interoperable** and **reusable**. Our codebase is accompanied by relevant documentation and guidelines for experimental replication. We wish to facilitate verification and future exploration into the stability gap phenomenon with upcoming optimization choices. In the context of replicability, it is worth pointing out that machine learning experiments inherently involve a randomness factor. This determines that replicating our calculations will naturally lead to results that are close to the values reported in this study, rather than identical ones.

Ethical considerations. Our choice of dataset, consisting of rotated grayscale digits, ensures that our study is free of sensitive or private information. To mitigate potential biases in the datasets, such as underrepresented digits, and arrive at reliable results, we randomly seed the shuffling of our training and testing samples. We also rerun each configuration an equal number of times before reporting an aggregated figure. This grants a robust evaluation of the model that accounts for possible fluctuations. We acknowledge that our research is niche, specifically focusing on task-based, domain-incremental learning scenarios, with a narrow set of classification targets. This requires broader exploration to establish universal applicability in the realm of continual learning.

[3] **https://github.com/chrisobis28/stability-gap-optimization**

References

Choi, D., Shallue, C., Nado, Z., Lee, J., Maddison, C., and Dahl, G. (2019). On empirical comparisons of optimizers for deep learning. *arXiv preprint arXiv:1910.05446*.

Duchi, J. C., Hazan, E., and Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12:2121–2159.

Flesch, T., Balaguer, J., Dekker, R., Nili, H., and Summerfield, C. (2018). Comparing continual task learning in minds and machines. *Proceedings of the National Academy of Sciences*, 115(44):E10313–E10322.

Goodfellow, I. J., Mirza, M., Da, X., Courville, A. C., and Bengio, Y. (2014). An empirical investigation of catastrophic forgeting in gradient-based neural networks. In *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*.

Graves, A. (2014). Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*.

Grossberg, S. (1982). Processing of expected and unexpected events during conditioning and attention: A psychophysiological theory. *Psychological review*, 89:529–72.

He, K., Zhang, X., Ren, S., and Sun, J. (2015). Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *2015 IEEE International Conference on Computer Vision, ICCV 2015, Santiago, Chile, December 7-13, 2015*, pages 1026–1034.

Hess, T., Tuytelaars, T., and van de Ven, G. M. (2024). Two complementary perspectives to continual learning: Ask not only what to optimize, but also how. In *Proceedings of the 1st ContinualAI Unconference, 2023*, volume 249 of *Proceedings of Machine Learning Research*, pages 37–61.

Kingma, D. P. and Ba, J. (2015). Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*.

Kirkpatrick, J., Pascanu, R., Rabinowitz, N., Veness, J., Desjardins, G., Rusu, A. A., Milan, K., Quan, J., Ramalho, T., Grabska-Barwinska, A., Hassabis, D., Clopath, C., Kumaran, D., and Hadsell, R. (2017). Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences*, 114(13):3521–3526.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems 2012. Proceedings of a meeting held December 3-6, 2012, Lake Tahoe, Nevada, United States*, pages 1106–1114.

Kudithipudi, D., Aguilar-Simon, M., Babb, J., Bazhenov, M., Blackiston, D., Bongard, J., Brna, A., Chakravarthi Raja, S., Cheney, N., Clune, J., Daram, A., Fusi, S., Helfer, P., Kay, L., Ketz, N., Kira, Z., Kolouri, S., Krichmar, J., Kriegman, S., and Siegelmann, H. (2022). Biological underpinnings for lifelong learning machines. *Nature Machine Intelligence*, 4:196–210.

Lange, M. D., van de Ven, G. M., and Tuytelaars, T. (2023). Continual evaluation for lifelong learning: Identifying the stability gap. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*.

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

Li, F.-F., Adeli, E., Johnson, J., and Durante, Z. (2025). Cs231n: Deep learning for computer vision notes.

Li, Z. and Hoiem, D. (2017). Learning without forgetting. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 40(12):2935–2947.

McCloskey, M. and Cohen, N. J. (1989). Catastrophic interference in connectionist networks: The sequential learning problem. volume 24 of *Psychology of Learning and Motivation*, pages 109–165. Academic Press.

Nesterov, Y. E. (1983). A method for solving the convex programming problem with convergence rate O(1/kˆ2). *Soviet Mathematics Doklady*, 27:372–376.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Köpf, A., Yang, E. Z., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. (2019). Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 8024–8035.

Polyak, B. (1964). Some methods of speeding up the convergence of iteration methods. *USSR Computational Mathematics and Mathematical Physics*, 4(5):1–17.

Ratcliff, R. (1990). Connectionist models of recognition memory: Constraints imposed by learning and forgetting functions. *Psychological Review*, 97:285–308.

Robins, A. V. (1995). Catastrophic forgetting, rehearsal and pseudorehearsal. *Connection Science*, 7(2):123–146.

Rolnick, D., Ahuja, A., Schwarz, J., Lillicrap, T. P., and Wayne, G. (2019). Experience replay for continual learning. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 348–358.

Sutskever, I., Martens, J., Dahl, G., and Hinton, G. (2013). On the importance of initialization and momentum in deep learning. In *Proceedings of the 30th International Conference on Machine Learning*, volume 28 of *Proceedings of Machine Learning Research*, pages 1139–1147, Atlanta, Georgia, USA. PMLR.

Tieleman, T. and Hinton, G. (2012). Lecture 6.5 - rmsprop: Divide the gradient by a running average of its recent magnitude. *Coursera : Neural Networks for Machine Learning*, 4:26-31.

van de Ven, G. M., Soures, N., and Kudithipudi, D. (2014). Continual learning and catastrophic forgetting. *arXiv preprint arXiv:2403.05175*.

van de Ven, G. M. and Tolias, A. S. (2019). Generative replay with feedback connections as a general strategy for continual learning. *arXiv preprint arXiv:1809.10635*.

van de Ven, G. M., Tuytelaars, T., and Tolias, A. S. (2022). Three types of incremental learning. *Nature Machine Intelligence*, 4(12):1185–1197.

Wilkinson, M., Dumontier, M., Aalbersberg, I. J., Appleton, G., Axton, M., Baak, A., Blomberg, N., Boiten, J.-W., Bonino da Silva Santos, L. O., Bourne, P., Bouwman, J., Brookes, A., Clark, T., Crosas, M., Dillo, I., Dumon, O., Edmunds, S., Evelo, C., Finkers, R., and Mons, B. (2016). The fair guiding principles for scientific data management and stewardship. *Scientific Data*, 3.

Wilson, A. C., Roelofs, R., Stern, M., Srebro, N., and Recht, B. (2017). The marginal value of adaptive gradient methods in machine learning. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 4148–4158.

A Appendix

A.1 Mathematical Formulations of Optimizers

The following formulas define the first-order optimization strategies under analysis and are in accordance with PyTorch's default methods. We decide to include and detail these for completeness.

Stochastic Gradient Descent with Momentum.
The following equations are derived from the work of Sutskever et al. (2013).

$$g_t \leftarrow \nabla_\theta f_t(\theta_{t-1}; X_{batch}, y_{batch}) \tag{5}$$

$$v_t \leftarrow \mu v_{t-1} + g_t \tag{6}$$

$$\theta_t \leftarrow \theta_{t-1} - \gamma v_t \tag{7}$$

Equation 5 defines the gradient g_t of the loss function $\nabla_\theta f$ at timestamp t, with respect to the model parameters θ at iteration $t-1$. This is computed on a mini-batch X_{batch} of the training data, with the corresponding target labels y_{batch}. Although the formula also covers the case of a time-dependent loss function, this is time invariant in our case. Equation 6 refers to the velocity buffer v at iteration t, shaped by the momentum coefficient μ, the accumulation of gradients v_{t-1} and the previously calculated one g_t. The initial velocity buffer is given the value of the gradient in the first step, compared to other instances of the algorithm where this is initialized to zero. Equation 7 gives the update of parameters θ at iteration t, by subtracting the velocity term v_t, scaled by the learning rate γ, from their previous values of θ_{t-1}.

Nesterov Accelerated Gradient (NAG).
Similarly to the previous optimizer, the formulas are inspired by Sutskever et al. (2013).

$$g_t \leftarrow \nabla_\theta f_t(\theta_t + \mu v_{t-1}; X_{batch}, y_{batch}) \tag{8}$$

$$v_t \leftarrow \mu v_{t-1} + g_t \tag{9}$$

$$\theta_t \leftarrow \theta_{t-1} - \gamma v_t \tag{10}$$

Equation 8 reflects the integration of the momentum-scaled velocity component μv_{t-1} in computing the current gradient g_t. Intuitively, equations 9 and 10 mirror 6 and 7, respectively. As in mini-batch SGD, the same principle of initializing the momentum buffer is applied.

Adaptive Gradient Algorithm (AdaGrad).
The following implementation of *AdaGrad* corresponds to the algorithm proposed by Duchi et al. (2011).

$$g_t \leftarrow \nabla_\theta f_t(\theta_{t-1}; X_{batch}, y_{batch}) \tag{11}$$

$$G_t \leftarrow G_{t-1} + g_t^2 \tag{12}$$

$$\theta_t \leftarrow \theta_{t-1} - \frac{\gamma}{\sqrt{G_t} + \epsilon} g_t \tag{13}$$

Equation 11 defines the standard loss function gradient formula, as previously described. Equation 12 calculates the buffer G of summed square gradients g^2, at iteration t, scaling down the learning rate γ for all future parameter θ updates (equation 13). A small constant ϵ is added to the denominator to exclude the possibility of dividing by zero. The meaning of the other terms does not change.

Root Mean Square Propagation (RMSprop).
The works of Graves (2014) and Tieleman and Hinton (2012) gave rise to the following formulas that correspond to our application of *RMSprop*.

$$g_t \leftarrow \nabla_\theta f_t(\theta_{t-1}; X_{batch}, y_{batch}) \tag{14}$$

$$v_t \leftarrow \alpha v_{t-1} + (1-\alpha) g_t^2 \tag{15}$$

$$\theta_t \leftarrow \theta_{t-1} - \frac{\gamma}{\sqrt{v_t} + \epsilon} g_t \tag{16}$$

The update of the gradient memory buffer v in equation 15 uses a smoothing constant α to dynamically adjust the weights of old and recent gradient components. Equations 14 and 16 naturally follow from their correspondents in *AdaGrad*, preserving the meaning of the parameters.

Adaptive Moment Estimation (Adam).
Finally, *Adam* is applied according to its introduction by Kingma and Ba (2015).

$$g_t \leftarrow \nabla_\theta f_t(\theta_{t-1}; X_{batch}, y_{batch}) \tag{17}$$

$$m_t \leftarrow \beta_1 m_{t-1} + (1-\beta_1) g_t \tag{18}$$

$$v_t \leftarrow \beta_2 v_{t-1} + (1-\beta_2) g_t^2 \tag{19}$$

$$\hat{m}_t \leftarrow \frac{m_t}{1-\beta_1^t} \tag{20}$$

$$\hat{v}_t \leftarrow \frac{v_t}{1-\beta_2^t} \tag{21}$$

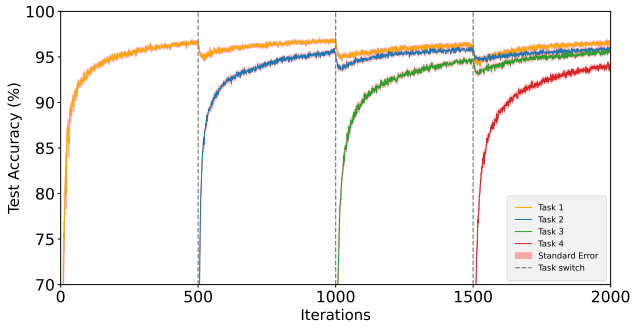
$$\theta_t \leftarrow \theta_{t-1} - \frac{\gamma \hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon} \tag{22}$$

Equation 17 represents the standard computation of the loss function gradient, given a mini-batch training approach. Both equations 18 and 19 produce exponentially averaged results, with the former used as momentum-like behavior (called the first moment estimation), and the latter allowing for an informed adaptation of the learning rates, as in the case of *RMSprop*. Equations 20 and 21 define bias correction for both moment estimates to ensure stable updates in early iterations, as $\lim_{t\to\infty} \hat{m}_t = m_t$ and $\lim_{t\to\infty} \hat{v}_t = v_t$ with $0 < \beta_1, \beta_2 < 1$. The update of parameters in equation 22 follows naturally from all the components mentioned above.

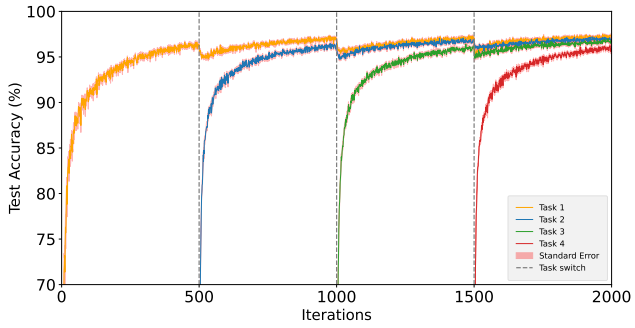
A.2 Optimizers performance

Optimizer	ACC (%) ↑	FORG (%) ↑	min-ACC (%) ↑	TBP (it.) ↓	GD (%) ↓	DS	RS
SGD 0μ	91.78 ± 0.12	**-1.61** ± 0.15	87.35 ± 0.08	220.6 ± 4.3	2.30 ± 0.11	-0.054 ± 0.003	0.022 ± 0.001
SGD 0.3μ	93.49 ± 0.09	-1.47 ± 0.16	89.25 ± 0.09	208.8 ± 10.1	2.33 ± 0.12	-0.092 ± 0.005	0.029 ± 0.002
SGD 0.5μ	94.47 ± 0.06	-1.36 ± 0.13	90.54 ± 0.05	196.2 ± 8	**2.24** ± 0.09	-0.145 ± 0.012	0.041 ± 0.003
SGD 0.7μ	95.63 ± 0.11	-0.95 ± 0.16	**91.14** ± 0.08	194.8 ± 9.1	3.06 ± 0.11	-0.283 ± 0.015	0.118 ± 0.008
SGD 0.9μ	**96.58** ± 0.04	-0.62 ± 0.10	90.12 ± 0.14	**166.2** ± 15.1	5.02 ± 0.15	-0.384 ± 0.011	0.231 ± 0.008
NAG 0.3μ	93.44 ± 0.12	**-1.40** ± 0.18	89.1 ± 0.09	206.5 ± 6.1	2.49 ± 0.14	-0.068 ± 0.004	0.044 ± 0.007
NAG 0.5μ	94.5 ± 0.07	-1.21 ± 0.11	90.16 ± 0.12	195.6 ± 8.2	**2.38** ± 0.25	-0.194 ± 0.051	0.067 ± 0.018
NAG 0.7μ	95.61 ± 0.09	-0.88 ± 0.10	91.62 ± 0.08	191.5 ± 10.8	2.47 ± 0.09	-0.252 ± 0.014	0.087 ± 0.008
NAG 0.9μ	**96.82** ± 0.04	-0.67 ± 0.09	**91.92** ± 0.10	**180.3** ± 10.1	3.56 ± 0.09	-0.309 ± 0.01	0.168 ± 0.011
AdaGrad	95.49 ± 0.03	-0.36 ± 0.10	92.61 ± 0.06	208.4 ± 7.9	1.68 ± 0.06	-0.084 ± 0.005	0.031 ± 0.002
RMSprop	96.69 ± 0.10	-0.74 ± 0.13	**93.61** ± 0.05	**134.1** ± 7.2	**1.4** ± 0.08	-0.141 ± 0.035	0.070 ± 0.017
Adam	**96.82** ± 0.07	**-0.83** ± 0.14	90.76 ± 0.17	150.1 ± 5.4	4.31 ± 0.13	-0.296 ± 0.010	0.191 ± 0.011

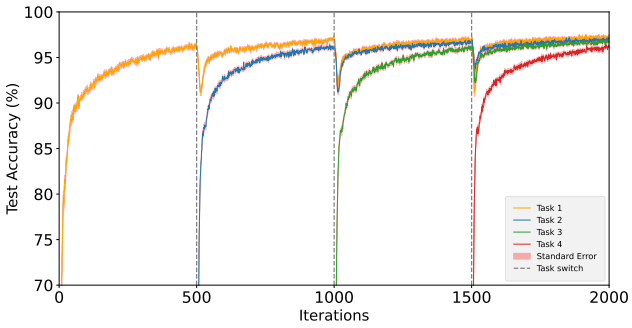
Table 6: Stability gap and general stability-plasticity evaluation of optimizers. Metrics include average accuracy (**ACC**), average forgetting (**FORG**), minimum accuracy (**min-ACC**), time below performance (**TBP**), gap depth (**GD**), decrease slope (**DS**), and recovery slope (**RS**). We report each metric figure as the sample mean and standard error across 10 randomly seeded experiments. The arrows indicate performance superiority, while highlighted scores represent the best performances recorded within specific optimization contexts. **DS** and **RS** are not included as comparison means because they do not directly reflect performance. These are given by horizontally dividing the table in 3 parts: *SGD* with momentum, *NAG* with momentum, and adaptive methods.



(a) AdaGrad

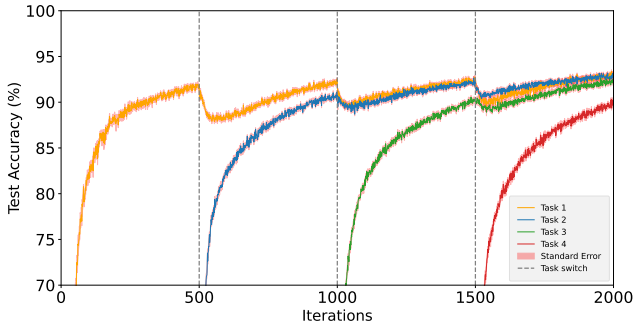


(b) RMSprop

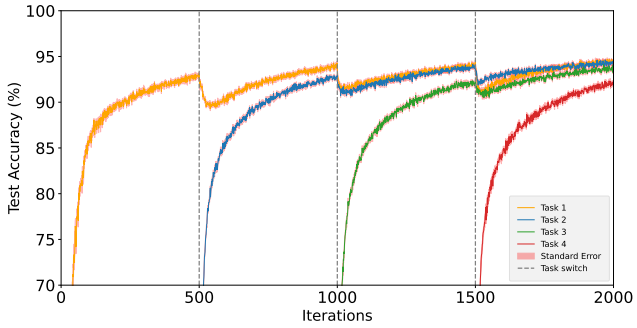


(c) Adam

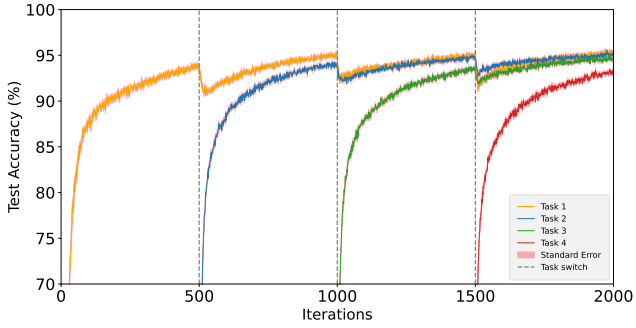
Figure 7: Performances of the (a) *AdaGrad*, (b) *RMSprop*, and (c) *Adam* optimizers on four sequentially introduced tasks. We plot the sample mean of 10 randomly seeded experimental runs and shade the surrounding area given by the standard error across them. *AdaGrad* and *RMSprop* both present a minor stability gap depth and decrease slope, while the former struggles more to integrate new tasks and recover from accuracy loss. *Adam* experiences a sudden, deep and short-lasting stability gap.



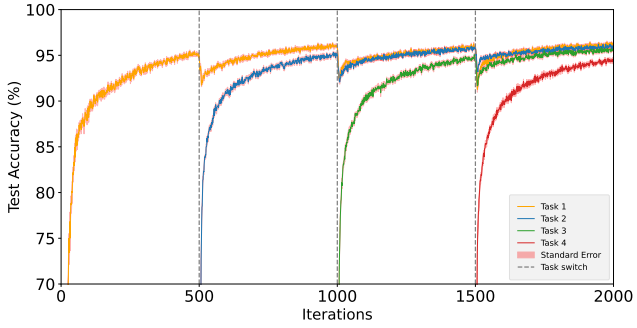
(a) SGD with Momentum 0



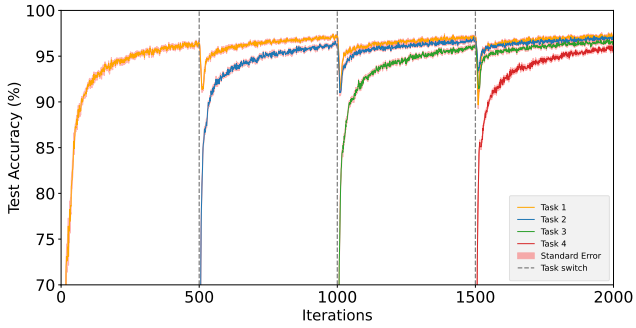
(b) SGD with Momentum 0.3



(c) SGD with Momentum 0.5

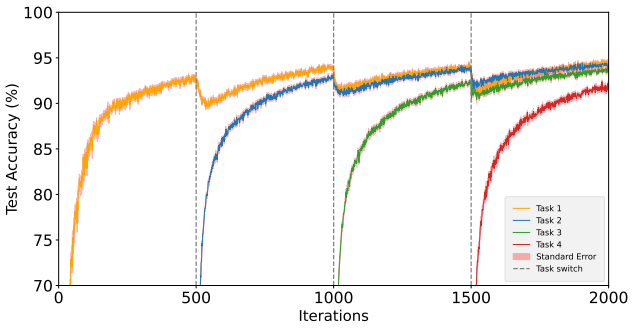


(d) SGD with Momentum 0.7

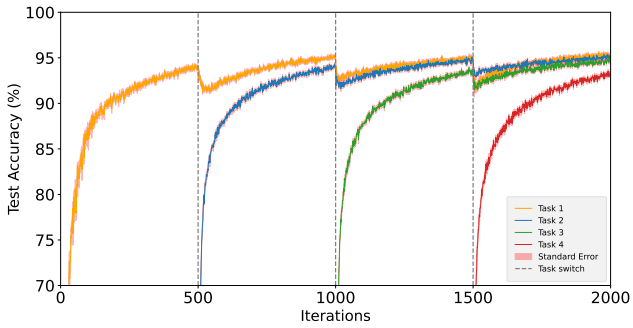


(e) SGD with Momentum 0.9

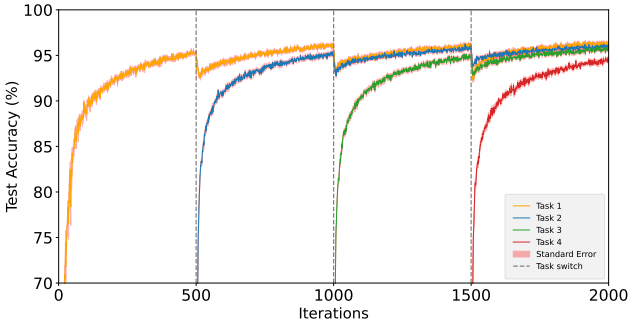
Figure 8: Performance of *SGD* on four progressively introduced tasks, using momentum (μ) values (a) 0, (b) 0.3, (c) 0.5, (d) 0.7, and (e) 0.9. Each plot shows the sample mean across 10 randomly seeded experiments, with an uncertainty region defined by the standard error, highlighted in red. Increasing the momentum coefficient narrows the stability gap and increases its magnitude, while improving convergence accuracy.



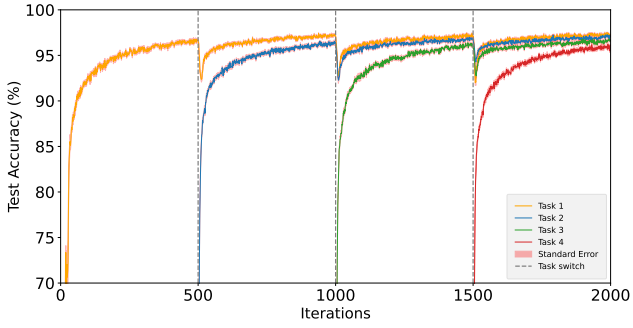
(a) NAG with Momentum 0.3



(b) NAG with Momentum 0.5



(c) NAG with Momentum 0.7



(d) NAG with Momentum 0.9

Figure 9: Performance of *NAG* on the four tasks under analysis, using different momentum (μ) values (a) 0.3, (b) 0.5, (c) 0.7, and (d) 0.9. Each plot contains the sample mean of 10 randomly seeded experiments and highlight the region of the standard error computed across these runs in red. The accuracies reported reflect the behavior of *SGD*, where a higher momentum implies a higher amplitude of the gap and a shorter duration. Similarly, higher momentum values accelerate convergence. However, *NAG* shows improved performance in minimizing the stability gap, given high momentum values.

A.3 Gridsearch Procedure

Optimizer	Additional Parameters	$\gamma = 0.0001$	$\gamma = 0.001$	$\gamma = 0.01$	$\gamma = 0.1$
SGD	$\mu = 0$	–	15.26 ± 0.06	67.18 ± 0.27	**92.19** ± 0.20
	$\mu = 0.3$	–	26.58 ± 0.24	70.96 ± 0.15	**93.48** ± 0.18
	$\mu = 0.5$	–	39.23 ± 0.13	75.64 ± 0.21	**94.52** ± 0.10
	$\mu = 0.7$	–	51.68 ± 0.19	82.87 ± 0.14	**95.72** ± 0.08
	$\mu = 0.9$	–	66.97 ± 0.17	91.74 ± 0.12	**96.66** ± 0.08
NAG	$\mu = 0.3$	–	25.51 ± 0.17	71.17 ± 0.18	**93.38** ± 0.18
	$\mu = 0.5$	–	34.15 ± 0.05	75.78 ± 0.16	**94.63** ± 0.12
	$\mu = 0.7$	–	50.63 ± 0.22	82.98 ± 0.17	**95.70** ± 0.10
	$\mu = 0.9$	–	67.34 ± 0.11	91.97 ± 0.08	**96.85** ± 0.08
AdaGrad	–	54.40 ± 0.28	79.54 ± 0.10	**95.36** ± 0.05	93.8 ± 0.14
RMSprop	$\alpha = 0.99$	90.05 ± 0.23	96.83 ± 0.09	94.22 ± 0.23	–
	$\alpha = 0.9$	89.50 ± 0.14	**96.89** ± 0.09	94.64 ± 0.29	–
Adam	$\beta_1 = 0.99, \beta_2 = 0.999$	86.64 ± 0.11	95.41 ± 0.11	94.41 ± 0.12	–
	$\beta_1 = 0.99, \beta_2 = 0.99$	86.55 ± 0.17	95.99 ± 0.05	94.08 ± 0.15	–
	$\beta_1 = 0.9, \beta_2 = 0.999$	89.66 ± 0.13	96.63 ± 0.09	95.95 ± 0.08	–
	$\beta_1 = 0.9, \beta_2 = 0.99$	89.76 ± 0.14	**96.79** ± 0.05	95.93 ± 0.13	–

Table 7: Unified table of the hyperparameter gridsearch conducted over all optimizers. The table entries contain the average accuracy **ACC** as the sample mean of 5 independent runs, accompanied by the standard error across these. For *SGD* and *NAG*, μ denotes momentum, *AdaGrad*'s decay rate is represented by α, while *Adam* uses β_1 and β_2 to calculate moment estimates. Tuning the hyperparameters is essential to ensure a fair comparison of the methods. We make our selection based on the stability-plasticity **ACC** metric scored to prioritize performance. We bold the highest scoring configuration for each optimization setting. This corresponds to the hyperparameters used to benchmark the optimizer in our final experiments.