HOCHSCHULE MITTWEIDA
University of Applied Sciences

MASTER THESIS

Mr.
Christoph Demus

Analysis of Continuous Learning Strategies at the Example of Replay-Based Text Classification

Mittweida, July 2023

Faculty of **Applied Computer Sciences and Biosciences**

MASTER THESIS

Analysis of Continuous Learning Strategies at the Example of Replay-Based Text Classification

Author:
Christoph Demus

Course of Study:
Cybercrime / Cybersecurity

Seminar Group:
CY20wS-M

First Examiner:
Prof. Dr. rer. nat. Dirk Labudde

Second Examiner:
Prof. Dr. rer. nat. Michael Spranger

Submission:
Mittweida, 30.07.2023

Defense/Evaluation:
Mittweida, 2023

Faculty of **Applied Computer Sciences and Biosciences**

MASTER THESIS

Analyse von Strategien für Kontinuierliches Lernen am Beispiel Replay-Basierter Textklassifikation

Author:
Christoph Demus

Course of Study:
Cybercrime / Cybersecurity

Seminar Group:
CY20wS-M

First Examiner:
Prof. Dr. rer. nat. Dirk Labudde

Second Examiner:
Prof. Dr. rer. nat. Michael Spranger

Submission:
Mittweida, 30.07.2023

Defense/Evaluation:
Mittweida, 2023

Bibliographic Description

Demus, Christoph:
Analysis of Continuous Learning Strategies at the Example of Replay-Based Text Classification. – 2023. – 75 S.
Mittweida, Hochschule Mittweida – University of Applied Sciences, Faculty of Applied Computer Sciences and Biosciences, Master Thesis, 2023.

Abstract

Continuous learning is a research field that has significantly boosted in recent years due to highly complex machine and deep learning models. Whereas static models need to be retrained entirely from scratch when new data get available, continuous models progressively adapt to new data saving computational resources. In this context, this work analyzes parameters impacting replay-based continuous learning approaches at the example of a data-incremental text classification task using an MLP and LSTM. Generally, it was found that replay improves the results compared to naive approaches but achieves not the performance of a static model. Mainly, the performances increased with more replayed examples, and the number of training iterations has a significant influence as it can partly control the stability-plasticity-trade-off. In contrast, the impact of balancing the buffer and the strategy to select examples to store in the replay buffer were found to have a minor impact on the results in the present case.

Referat

Kontinuierliches Lernen ist ein Forschungsgebiet, das in den letzten Jahren durch komplexe Machine- und Deep-Learning-Modelle einen enormen Aufschwung erfahren hat. Während statische Modelle komplett neu trainiert werden müssen, wenn neue Daten zur Verfügung stehen, passen sich kontinuierliche Modelle schrittweise an neue Daten an und sparen so Rechenressourcen. In diesem Zusammenhang untersucht diese Arbeit Parameter, die sich auf replay-basierte kontinuierliche Lernansätze auswirken, am Beispiel einer Daten-inkrementellen Textklassifizierungsaufgabe unter Verwendung eines MLPs und eines LSTMs. Es wurde festgestellt, dass Replay die Ergebnisse im Vergleich zu trivialen Ansätzen verbessert, aber nicht die Leistung statischer statischer Modelle erreicht. Hauptsächlich stieg die Leistung mit einer höheren Anzahl von wiedergegebenen Beispielen, und die Anzahl der Trainingsiterationen hat einen großen Einfluss gezeigt, da sie den Stability-Plasticity-Trade-Off teilweise steuern kann. Im Gegensatz dazu wurde festgestellt, dass die Auswirkungen eines ausbalancierten Replay-Puffers und der Strategie zur Auswahl der Beispiele für die Speicherung im Puffer im vorliegenden Fall nur geringe Auswirkungen auf die Ergebnisse hatten.

Contents

List of Figures

List of Tables

Acronyms

A-GEM	Averaged Gradient Episodic Memory
ACC	Average Accuracy
AIA	Average Incremental Accuracy
BWT	Backward Transfer
CF	Catastrophic Forgetting
CL	Continuous Learning
CLS	Complementary Learning System
EWC	Elastic Weight Consolidation
FWT	Forward Transfer
GEM	Gradient Episodic Memory
GR	Generative Replay
HAT	Hard Attention to the Task
iCaRL	Incremental Classifier and Representation Learning
IDBR	Information Disentanglemant based regularization
LAMOL	Language Modeling for Lifelong Language Learning
LCA	Learning Curve Area
LCSH	Library of Congress Subject Headings
LSC	Lifelong Sentiment Classification
LSTM	Long Short-Term Memory Network
LwF	Learning without Forgetting
MBPA	Memory-based Parameter Adaptation
MLP	Multi-Layer Perceptron
NLP	Natural Language Processing
PNN	Progressive Neural Networks
SI	Synaptic Intelligence

1 Introduction

The fields of big data analysis, machine learning, and artificial intelligence have received a lot of attention in recent years, and much progress has been made. Even though essential ideas like the perceptron model (Rosenblatt, 1957), which are still the basis of current neural networks, were already developed in the second half of the 20th century, the progress and new possibilities regarding computational resources during the last years brought a massive boost to these research fields. For example, the global total corporate artificial intelligence investment has increased sevenfold from 2015 to 2022 (Stanford University, 2023) and the worldwide market size of artificial intelligence is forecasted to increase eighteenfold until 2030 from 2021 (Next Move Strategy Consulting, 2023).

The development went from classical models like Support Vector Machines or decision trees to more complex deep learning models. A large field of research and application of machine learning and artificial intelligence is Natural Language Processing (NLP) with tasks such as text classification, translation, question answering, and text generation but also part-of-speech-tagging and named entity recognition to name only a selection of possible tasks. NLP, in particular text classification, is also the focus of this work.

1.1 Motivation and Classification

Most research about supervised learning tasks focuses on building models to best solve a specific task defined by a static training dataset. This means a labeled dataset is used to train a model, which is then evaluated on a test split of the dataset with a similar feature distribution. Recently, large language models like Transformers often achieved good results and became state-of-the-art in many NLP-tasks (Minaee et al., 2021). Those developments are significant efforts in the research field. However, in many cases, the experimental setup of a static training and test data set is not realistic in the long-term usage for real applications as the data underlays shifts regarding the feature and label distribution. For instance, in a hate speech classification system for text comments, the topics of the comments and the proportion of hate speech can change over time. This usually leads to a decrease in performance with the result that those static models become outdated after a specific amount of time. Then a model would need to be retrained with updated data. In practice, this is the problem because regularly used large language models have high requirements regarding computational resources, which end users do not have available in most cases. Additionally, manual effort is required to do the retraining. Both points are major drawbacks that may prevent the practical application of those models (W. Zhang et al., 2023).

This problem is addressed by Continuous Learning (CL). CL deals with the issue of continuously updating models to omit the retraining step from time to time in static models. Continuous updates of models in small steps instead of complete retraining have the advantage that the requirements on computational resources are smaller, as the computation is spread over a long time. When a model is often updated with just a few new training examples, the computational power needed to fulfill the minor update is much smaller than when complete retraining must be done. In other words, it could be thought that the retraining is spread into many small updates executed over a long time span. The second advantage is that the model is updated continuously with each new data point or in small steps instead of bigger updates in the case of complete retraining. Ideally, CL omits decreases in the performance over time.

Nevertheless, CL is still a big challenge. It was only a minor research area before it became more popular when models became much more complex - too complex to be retrained within short periods. This raised the ContinualAI[1] research organization and open community in 2018, intending to drive research in this area. CL is confronted with a trade-off between efficiency, scalability, and adaptability to achieve the goal of sustainable models that can be used in the long term (Cossu et al., 2021). Compared to static models, in CL many processes are different - beginning in the data acquisition and not ending with the evaluation process. There are still no established definitions of most related terminology, making work in this field difficult. At the time of writing, research in this field is growing. Much research focuses on image processing tasks, whereas only a minor part focuses on NLP-tasks. Furthermore, it must be distinguished in multiple application scenarios, making related work rarer again.

1.2 Objective and Contribution

This work deals with CL in NLP tasks, concretely with text classification. The first part aims to highlight the main difficulties of CL and current solutions for addressing those challenges. In this context, it will among others be referred to dataset shifts, Catastrophic Forgetting (CF), and the stability-plasticity trade-off.

In the second part, a data-incremental text classification scenario is used to examine a replay-based CL-approach in more detail regarding influential parameters. Replay-based approaches save selected examples from the training data in a memory. In the further training process, a small amount of preserved examples is added to the new training examples and replayed to prevent the network from forgetting previously learned knowledge. As of now, no research analyzing those parameters in the mentioned constellation. This work, therefore, contributes to the following two main research questions:

Research Question 1 In a replay-based approach, how do the number of replayed examples, the number of training iterations, and the buffering strategy affect the learning process and performance?

Research Question 2 To what extent does replay improve the results compared to a naive approach, and how does it compare to a model trained with all data at once (static model)?

For all the experiments carried out to answer the research questions, a Multi-Layer Perceptron (MLP) and a Long Short-Term Memory Network (LSTM) model are compared as representatives of a simple and a more complex neural network respectively. For question 1, each parameter is changed while keeping the others constant to be able to measure its impact. The buffer-strategy-parameter is again split into three parts. The impact of balancing the

[1] https://www.continualai.org/ (visited on June 14, 2023)

saved examples in the memory buffer and the strategy for selecting examples to be saved in the buffer is examined. Furthermore, both are combined. Research Question 2 aims to classify the performance of the model. As a lower bound commonly naive models apply no strategies to address the challenges of CL. Models trained with the complete dataset at once (classical models) can be seen as the upper bound of the performance. For research question 2 it should therefore be examined where in between models using the replay-based approach lie.

1.3 Structure of this Work

This work begins with the theoretical background of CL depicting differences compared to classical models and evaluation. Chapter 2 gives an overview including a formal definition of CL, descriptions of possible learning scenarios, and the main challenges. Subsequently, the main strategies to address CL are introduced in Chapter 3 including a description of methods for the learning process as well as evaluation protocols for the continuous case. Afterward, in Chapter 4 NLP comes more in the focus with related work regarding CL in NLP-tasks before the practical analyses are done in Chapter 5. In this part, first, the methodology is introduced (Section 5.2), followed by the description of the experiments (Section 5.3) and finally their results and discussion (Section 5.4). The findings of this work are summarized in Chapter 6 and an outlook for future work is given in Chapter 7.

In addition, supplementary material, including the created dataset, all models, and the code used, is available online.[2]

[2]https://drive.google.com/drive/folders/1K6XkQ7iWWNjd8DtS6skQ6MT8ZioeuA5s

2 Overview of Continual Learning

This chapter introduces CL in general, starting with a description of the workflow compared to classical, static models. Subsequently, the CL algorithm is formally defined. As CL can be applied to a wide range of different types of applications, it is essential to distinguish them because a system's requirements differ depending on the type of application. Those types are coarsely divided into three scenarios which are explained in Section 2.3. Building upon the basics of CL, lastly the main problems and challenges are explained in this chapter.

2.1 Workflow of Continuous Learning Systems

Classic, static machine, or deep learning models are usually trained on a static dataset. Once the dataset is created, it is used to train a model, which in turn is deployed and used for prediction. Google Cloud (2023) describes the machine learning workflow as a sequence of seven stages, including data preparation, model coding followed by training, deployment, and finally, generating predictions, monitoring them, and version management. The problem with this procedure is the assumption of a stationary data distribution and that the workflow does not provide for any update. This means it is assumed that the unseen data in the prediction phase has the same distribution as the training set (Hu et al., 2020; Hurtado et al., 2023). However, the model has limited usability if the distribution of the data to predict changes as it reflects only the data seen so far in the training phase (Hurtado et al., 2023).

The workflow must be adjusted for CL to allow model updates using new data. Mundt et al. (2022) suggests using the workflow of Google Cloud (2023) cyclically while applying changes and extensions in each step to fit the workflow for CL. However, the cycle does not represent continuous updates optimally. In particular, the stages of coding the model and deployment raise imprecision as coding the model should include the update process, and once the model is deployed, it should be updated in its work environment. Another workflow closer to the demands of CL and more specific is proposed by Hurtado et al. (2023). Building upon this, an adapted and extended version of a CL workflow is shown in Figure 2.1. It starts with the data collection. The data can arrive in batches or as a stream, depending on the scenario. The data needs to be preprocessed for further steps, including feature extraction and conversion to a machine-readable representation. The drift detection as the next step can be actively done if new classes or tasks are detected or passively in case of a constantly evolving data stream. According to the update strategy chosen, the model is then updated using the preprocessed data. Finally, continuous evaluation or monitoring of the predictions is important to ensure the stability and robustness of the model. At this point, the cycle starts again from the beginning when new data arrives. In this way, a model can be continuously updated to adapt to changing data distributions.

2.2 General Definition

This section introduces a general definition of CL and its formal notation. Generally speaking, a CL-System is a system that can learn new tasks by continuously adapting the current state of the system without losing the ability to solve prior tasks. Losing the ability to solve prior

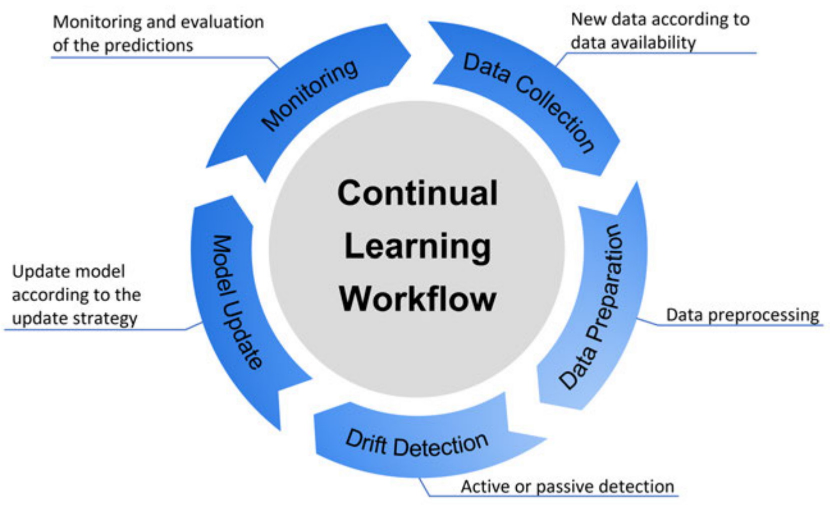

Continual Learning Workflow
Data Collection
New data according to data availability
Data Preparation
Data preprocessing
Drift Detection
Active or passive detection
Model Update
Update model according to the update strategy
Monitoring
Monitoring and evaluation of the predictions

Figure 2.1: Continual learning workflow (adapted from Hurtado et al., 2023).
Workflow of a CL-system. The cycle starts with newly arriving data (data collection) which is subsequently preprocessed. The system detects data drift actively (e.g., new task or class) or passively (continuously evolving data stream). Afterward, the model is updated and evaluated to ensure robustness.

tasks during the adaption process is also called catastrophic forgetting (Hassabis et al., 2017; Parisi et al., 2019; Thrun & Mitchell, 1995) and will be explained in detail in Section 2.4.2. This means the system is ideally able to incrementally learn new tasks without losing the ability to solve tasks learned before. Thereby the goal is to constantly improve the system's performance on all learned tasks by a progressive generalization of the system (Biesialska et al., 2020).

Formally the system has to learn sequentially arriving tasks $t_1 ... t_n$ with $t_i \in T$ where T is the set of task descriptors and each t_i is a specific task descriptor. A task descriptor could, for example, be a number in the easiest way. For each task, the data objects arrive in batches $b \in B_t$. Here, b is the batch index of the current tasks batch, and B_t is the respective set of batch indices for the current task (L. Wang et al., 2023). The specific data points of a batch are finally contained in the set $D_{t,b} = \{X_{t,b}, Y_{t,b}\}$ where $X_{t,b}$ contains the input samples of batch b in task t and $Y_{t,b}$ contains the respective labels.

Often in literature, the data objects are also summarized into a single training set for each task, and the subdivision in batches is omitted (Biesialska et al., 2020; Lesort et al., 2020). As the split in batches is more general, it is included in this definition. On the one hand, if the batch number per task is chosen to be one, the same result is achieved without batch notation. On the other hand, the use of batches of sequentially arriving single data points can be described by choosing the batch size one (one object per batch). This makes it very flexible and allows to describe a broad range of problems with a common notation.

The actual goal is then to learn a function (the actual prediction model) f and continuously update it with new arriving training samples to possibly changed circumstances, i.e., to new tasks, classes or domains (Biesialska et al., 2020):

$$f : X \times T \to Y \tag{2.1}$$

This means the model progressively uses the training samples X from all tasks T to learn to solve all existing tasks. Thereby the solution of a task is given by the prediction (label Y) for a given data point for a specific task.

The sequential update rule for the prediction function f can now be formulated as in the following CL-algorithm A^{CL} (Lesort et al., 2020):

$$A_s^{CL} : < f_{s-1} \, , \, X_{t,b} \, , \, M_{s-1} \, , \, Y_{t,b} > \rightarrow < f_s \, , \, M_s > \quad \forall t \in T, b \in B_t, s \in \mathbb{N} \tag{2.2}$$

The goal of the algorithm is to update the model f_{s-1}. s indicates the time step here, and therefore f_{s-1} is the model from the last time step, whereas the model after the update is denoted as f_s. The model learns from the training examples $X_{t,b}$ and their respective labels $Y_{t,b}$ in each time step. This states that a complete batch b of a specific task t is processed in each iteration. Therefore a batch denotes the smallest unit to process during training. M_{s-1} is a memory for storing data. It can store previous examples, partially trained models, or other relevant data (Awasthi & Sarawagi, 2019). This is necessary as it is often not automatically allowed or even possible to access previously used training examples again, except those are explicitly stored in the given memory. Usually, the memory is limited for each task, batch, or the complete system but not endless. Using the mentioned inputs, the algorithm updates (learns) the model resulting in the updated model f_s and updated memory M_s.

2.3 Continual Learning Scenarios

CL is needed when the circumstances of the problem that should be solved change over time. The model needs to continuously adapt to keep or improve the performance, whereas a static model without continuous training would experience decreasing performance and may be useless after some time.

Depending on which circumstances related to the problem are changing, different CL-Scenarios are distinguished. The three main scenarios are *task-incremental learning*, *domain-incremental learning*, and *class-incremental learning* (De Lange et al., 2021). Beyond that, other constellations or deviations of these scenarios are possible. For simplification, as it is the main focus of this work and also done in most literature, it is assumed here to have a classification problem (De Lange & Tuytelaars, 2021; De Lange et al., 2021; Hsu et al., 2018; Sodhani et al., 2022; van de Ven et al., 2022; Z. Wang et al., 2022), since classification problems are the main focus of this work. For other problem types, e.g., regression, the following definitions must be extended.

Three main objectives distinguish the scenarios between the sequentially learned tasks. These are the output distribution $P(Y)$, the observed classes $\{Y\}$, and if a task-identifier t is provided in the prediction phase or if it is not provided and possibly needs to be inferred by the system itself (De Lange et al., 2021; Hsu et al., 2018; Sodhani et al., 2022). In some cases, the input distribution $P(X)$ is also taken into account (Hsu et al., 2018; Sodhani et al., 2022). The output distribution $P(Y)$ describes the probability of observing a given class label, and therefore it also represents the ratio of the classes to each other. Conversely, the input distribution $P(X)$ is the probability of observing a given example x. In simple terms, this describes how the data is spread in the input space.

Scenario	Output Distribution	Classes	Task-ID
domain-incr.	$P(Y_t) = P(Y_{t+1})$	$\{Y_t\} = \{Y_{t+1}\}$	not required
class-incr.	$P(Y_t) \neq P(Y_{t+1})$	$\{Y_t\} \neq \{Y_{t+1}\}$	only given in training
task-incr.		$\{Y_t\} \cap \{Y_{t+1}\} = \emptyset$	given in training and prediction (evaluation)

Table 2.1: Distinction of the main CL-Scenarios.

CL-Scenarios are distinguished by comparison of their output distribution (class ratio), the present classes in a given task, and if, respectively, when a task-identifier is given. Y_t denotes the available class labels of task t.

It is essential here to understand the concept *task* as "an isolated training phase with a new batch of data, belonging to a new group of classes, new domain or a different output space" (De Lange et al., 2021). This does not necessarily coincide with the common understanding of the word *task*, i.e., a problem that has to be solved. Here this is related to the learning phase of the system, and it is two different tasks if the system first learns to distinguish zebras from horses, for example, and afterward, it learns to distinguish rabbits and cats. As new classes emerge in the second training phase, it is a new task by the given definition (van de Ven et al., 2022). If a system learns to solve one single problem but with successively arriving data, the learning procedure can also be divided into different tasks by the batches of arriving data.

Table 2.1 gives an overview over the mentioned CL-scenarios and their properties. In the following, each scenario is described in more detail.

The **task-incremental** scenario is, in general, the easiest of the three scenarios. This is because the task identifier is given for every task in the training and prediction (evaluation) phase. Consequently, multiple task-specific components can be separately used, and for a given task, the respective component can be activated for the prediction. In this case, the challenge is finding approaches to share knowledge between the separate components to reduce computational complexity (van de Ven et al., 2022). Otherwise, one could also use completely different models for each task. Nevertheless, separation is often necessary, as tasks could be of utterly different problem types with a different number of classes (Hsu et al., 2018). As the tasks are usually unrelated to each other with respect to their output, the class labels of the tasks are disjoint, and the output distributions are different, too.

The second scenario is the **domain-incremental** scenario. It is mainly characterized by the absence of task-identifiers as the system solves the same problem with the same classes but in different domains in every task (van de Ven et al., 2022). Therefore task identifiers are not necessary. Caused by the change of the domain, the input distribution changes with every task, such that $P(X_t) \neq P(X_{t+1})$. Most literature also assumes to have the same output distribution in all tasks ($P(Y_t) = P(Y_{t+1})$) (De Lange et al., 2021; Hsu et al., 2018; Sodhani et al., 2022). This simplifies the problem but must not always be true. If, for example, a hate speech classifier is trained to detect hate speech in different social networks (where the training on each network corresponds to one task), the assumption would mean the hate speech percentage is the same in all social networks. This is not necessarily true. In this scenario, the goal is to generalize the model such that it can solve the given problem in different domains with little domain-specific training.

It should be pointed out, that the domain-incremental scenario is not the same as domain adaptation. The latter aims only at a good performance on an unseen domain while it was trained on another domain before (Pan & Yang, 2010). Just the performance on the unseen domain is relevant. In contrast, domain-incremental learning aims at keeping the performance high on all domains (Hsu et al., 2018).

Finally, the **class-incremental** scenario is considered the most difficult of these three scenarios, as in the prediction phase, the system first needs to infer the task, and building upon this, it predicts one of the respective classes (van de Ven & Tolias, 2019). Though, the task identifier is only given in the training phase. In this scenario, each task contains new classes the system should learn to distinguish. Furthermore, the system should not only be able to distinguish the classes of each task but also to distinguish all of the growing number of classes (van de Ven et al., 2022). Due to the changed set of classes, also the output distribution changes with every task. An example would be a system that learns to classify images from the MNIST-dataset. In the first task, it may learn to distinguish the numbers 0 and 1. In the second task, it learns the numbers 2 and 3 and so on. In the end, one would ask the system which of the numbers from 0 to 9 a given image shows (Hsu et al., 2018).

Some literature furthermore considers data-incremental learning as a more general concept of learning from a data stream, i.e., the data arrives incrementally (De Lange & Tuytelaars, 2021; De Lange et al., 2021). This paradigm is very similar to the domain-incremental scenario, as the data does also not come with a task identifier. However, the input distribution $P(X)$ must also not change. Often both scenarios cannot clearly be separated, as changes in the input distribution are often constantly (Hurtado et al., 2023), making it difficult to define boundaries between domains.

2.4 Main Problems and Challenges

Compared to the classic machine learning workflow, CL comes with more complex difficulties, which is also one reason why it is rarely used yet, even though it would theoretically be more efficient to use continuously adapting models for many tasks. One can find three main and mutually dependent factors that make CL as difficult as it is. These are *distributional shifts in the dataset*, *CF*, and the *stability-plasticity-dilemma*. The starting point that makes continuous learning necessary in the first place are dataset shifts. In the adaption to those dataset shifts it comes to the problem of CF which is again related to the stability-plasticity-dilemma - a trade-off between learning new information and forgetting older knowledge. All these factors are interdependent and influence each other. In the following subsections, each of the factors is described in detail.

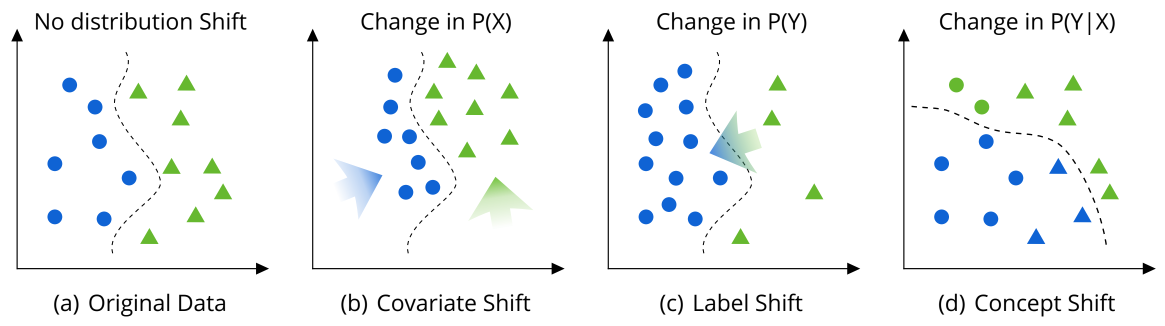


Figure 2.2: Types of dataset-shifts (adapted from Ataei et al. (2021).
Figure **(a)** shows exemplary output data in the feature space. In **(b)**, a covariate shift is depicted, where a shift in the feature distribution P(X) is observed. In contrast, in a label shift **(c)**, the distribution of labels P(Y) between the classes changes. Last, in a concept shift **(d)**, the separation line changes, with the result that the labels of data points change compared to the original data in (a). The shape of the data points highlights changed labels.

2.4.1 Dataset Shifts

Shifts in the data distribution are the first challenging factor, and one can understand dataset shifts as the reason why CL is necessary in the first place. Sometimes, dataset shift is imprecisely also called concept drift. However, concept drift is just one kind of possible dataset shifts (Gepperth & Hammer, 2016). Dataset shifts generally describe changes of various kinds in the underlying data over time. As a result, the environment of the machine- or deep learning system changes. This is where static systems will fail if the changes exceed a certain range and where CL-systems are necessary, as they should be able to adapt to those changes. However, to adapt to possible changes, it is first necessary to detect them (Quiñonero-Candela et al., 2008). Therefore, the following gives an overview of the three main dataset shifts observed in practice (Ataei et al., 2021). A more fine-grained categorization can be found in Quiñonero-Candela et al. (2008). It is essential to be aware of the main observable dataset shifts when designing CL-systems and their requirements for what changes they should be able to adapt to.

For an easier understanding, in the following, a system for hate speech detection in social media comments is used as an example. Thereby, the feature set of a comment, which can be constructed from its textual content and possibly other features, is denoted by X. In this example, the respective labels, which would be hate speech or not hate speech, are denoted by Y.

First, one could observe topic changes and a changing language over time. As a result, new words or new meanings of words might be observed dependent on the topic. Especially in social media, language is changing rapidly because of the use of slang and colloquial language. Consequently, a static model becomes outdated as it cannot cover such changes. Formally spoken, those observations concern changes in the feature space $P(X)$ (Ataei et al., 2021; Quiñonero-Candela et al., 2008). This kind of shift is also called *Covariate Shift* (Ataei et al., 2021; Quiñonero-Candela et al., 2008). In Figure 2.2, one can see that the data points are differently distributed over the input space compared to the original data. However, the label distribution, representing the number of data points per class, stays constant.

Another observation could be a changing amount of posted hate speech in a social network during a given time span, i.e., more hate speech is written as one year before. This is, therefore, a change in the label distribution $P(Y)$ (*Label Shift*) (Ataei et al., 2021; Quiñonero-Candela et al., 2008). Such changes are often not only a problem in CL but also in static models when training and target data are compared, as the training data is often not an adequate representation of the real-world target data the model should be applied to in production (Quiñonero-Candela et al., 2008). In Figure 2.2, the label shift is visible by fewer data points in the green (triangle) class than in the original data. That means the label distribution has changed.

Third, the understanding of what is defined as hate speech could change over time, which means the underlying concept of the task changes. Therefore, this shift is also called *Concept Shift* (Ataei et al., 2021). A concept shift means the same data point in the feature space might be classified into a different class than before. Formally spoken, the class probability given the data points X changes: $P(Y|X)$ (Ataei et al., 2021; Quiñonero-Candela et al., 2008). In this case, it is necessary that the model, which determines $P(Y|X)$, is able to adapt to the concept shift. In the diagram on the right in Figure 2.2 the concept shift can be seen by the changed separation border in the feature space. Depending on the extent of the concept shift and the model, one decides between global and local adaptation (Ataei et al., 2021; Gepperth & Hammer, 2016). In local adaptation, the model is only refitted for selected decision regions. For example, specific branches of a decision tree could be changed. Conversely, a global adaptation influences the whole model (Ataei et al., 2021).

The mentioned dataset shifts can also be distinguished in *real* and *virtual* shifts (Gepperth & Hammer, 2016; Widmer & Kubat, 1993). The concept shift $P(Y|X)$ is a real shift, as it is caused by changes in the underlying concept in the real world. In a virtual shift, on the other hand, there is no distributional change in the real world, but there is a changed input distribution $P(X)$ as a result of a bias in the selection of the data samples (Gepperth & Hammer, 2016; Widmer & Kubat, 1993). For example, the feature distribution of the training data could be different from the feature distribution of the test data. As in this case, the change of $P(X)$ did not happen in the real world but is caused by the selection of the training data, it is considered a virtual shift.

2.4.2 Catastrophic Forgetting

Natural cognitive systems, i.e., humans, are able to lifelong learn different task one after another without abruptly and totally forgetting tasks learned before (French, 1999). Humans do forget, but a slowly forgetting of selective information is not necessarily a disadvantage. De Lange et al. (2021) describes this as *graceful forgetting*. However, neither a less complex MLP nor Deep Neural Networks are able to satisfactory implement this capability in its standard implementations (Chen & Liu, 2018, pp. 55–57). In the classical concurrent training, the network is trained with all data at once (Ratcliff, 1990). Accordingly, if the data is well shuffled the data distribution appears stationary for the model (Goodrich & Arel, 2014). However, in CL-scenarios not all data is available at the same time and the order of the data is given by the application task and cannot be changed. Depending on the concrete application it may only be possible to change the order of new data examples within smaller subsets. This leads to the problem, that the data is not presented stationary as in the concurrent training but in a non-stationary manner (Goodrich & Arel, 2014). CF can occur in all CL-scenarios including data-incremental learning which can be understood as a single task (Toneva et al., 2018).

As a result it comes to the problem referred to as CF or catastrophic interference (Weaver et al., 1998) and was first observed by McCloskey and Cohen (1989) and Ratcliff (1990). It describes the phenomenon of a neural network to abruptly lose information about previous tasks when training on new data of a new task (Kirkpatrick et al., 2017; McClelland et al., 1995; Ratcliff, 1990; Robins, 1995). This leads to disruption of the performance of the previous task(s) .

The reason for CF is, that in CL the parameters of a model cannot be jointly optimized (Kirkpatrick et al., 2017). In contrast, at the beginning they can only be optimized for the first task which means to find one of multiple minima regarding the optimization (Ratcliff, 1990). When training is continued with new data, the parameters need to be changed to fit the new task. As the starting point of the further optimization of the network is given by the set of weights learned before, the updated solution will be near the previous solution (Ratcliff, 1990). This does not necessarily mean, that this new solution is an optimal one due to the restriction of sequential training. In simple words, a concurrent training jointly optimizing the parameters using all data at once would likely find a different solution.

In depth, the problem is mainly cause by two factors. First, shifts in the input space (new data) where each subset covers just a small local region of the entire input space (Toneva et al., 2018; Weaver et al., 1998). This makes it representative for a specific task but unrepresentative for the whole domain. Goodrich and Arel (2014) describe this as a temporal bias towards the respective training data of the task. Consequently, the optimization processes for each of the tasks converge to strongly different solutions (Toneva et al., 2018).

Second, the network architecture itself causes difficulties, as the same weights are relevant to multiple tasks. Accordingly, the change of a weight does affect several learned regions that refer to different tasks (Goodrich & Arel, 2014; Chen & Liu, 2018, pp. 55–57). The problem gets worse, when the network uses activation functions that are non-zero over large parts because the larger the non-zero part of the activation function is, the higher is the probability that the outcome of the neuron changes and effects previous knowledge (Goodrich & Arel, 2014). If there is redundancy within the networks weights, this can cushion the forgetting. However, latest after a certain number of learned tasks redundancy will decrease and therefore favor forgetting (Goodrich & Arel, 2014).

Even though the performance sharply drops when observing CF it was found that not all samples have the same probability to be forgotten. Toneva et al. (2018) distinguished the examples in forgettable and unforgettable examples. The latter were examples which were never forgotten by the system during the entire training procedure. Their analysis showed, that both groups contained many examples that were first learned during three to four training iterations but the set of forgettable examples contained more examples which needed more training iterations to be first learned. Furthermore, it was found, that learning items in groups makes them more resistant to forgetting (Ratcliff, 1990; Robins, 1995). The bigger the groups of examples are, the more elements can be jointly learned which is advantageous for the solution.

As CF is a main problem in CL, most CL-approaches deal with the question of how to overcome CF. An overview of possible strategies is given in Chapter 3.

2.4.3 Stability-Plasticity-Dilemma

The stability-plasticity dilemma represents a fundamental trade-off between two essential aspects of a learning system. Stability refers to the ability of a model to retain previously learned information and patterns without being significantly impacted by new data (De Lange et al., 2021; Parisi et al., 2019). In other words, a stable model resists changes to its existing representations when training is continued on new data, ensuring that past knowledge remains intact. In contrast, plasticity refers to the ability of a model to adapt and incorporate new information effectively (De Lange et al., 2021; Parisi et al., 2019). A highly plastic model can learn rapidly from new data, allowing it to update its representations or weights and adapt to changing environments.

Whereas humans tend to steadily forget throughout the lifetime it is unusual to rapidly forget when learning a new task (French, 1999; Parisi et al., 2019). This balance between stability and plasticity is in the human brain solved by the interaction of different brain areas (Hassabis et al., 2017; Kumaran et al., 2016). The theory behind is referred to as Complementary Learning System (CLS) (Kumaran et al., 2016; Parisi et al., 2019). Among others, the combination of the hippocampus and neocortex plays an important role. The hippocampus is characterized by a fast learning rate (high plasticity) but cannot manage long term memories (Hassabis et al., 2017). On the other side the neocortex has a slow learning rate but being able to build strong, overlapping connections to make long term memories possible (Hassabis et al., 2017; Kumaran et al., 2016).

The challenge arises from the inherent tension between these two competing objectives. A model that is too stable may struggle to learn from new data, as it tends to resist any modifications to its existing representations. Consequently, it helps to prevent CF (Hurtado et al., 2023), however new knowledge can rarely be learned, leading to a degradation of the overall performance when new tasks are added.

On the other hand, a model excessively plasticity can be too sensitive to new data. It could rapidly accommodate new patterns but losing track of previously learned knowledge in the process which is called CF. This lack of stability results in difficulties in maintaining information on previously seen tasks, as the model keeps adapting to new information at the expense of retaining old knowledge.

Finding the right balance between stability and plasticity to prevent CF is crucial for creating robust CL systems. Various approaches have been proposed to address the stability-plasticity dilemma, aiming to mitigate catastrophic forgetting while allowing the model to learn from new data effectively (Hurtado et al., 2023; Parisi et al., 2019).

3 Methods Addressing Continual Learning

The problems and challenges regarding CL mentioned before hamper the creation of robust and reliable CL-systems. Therefore, most approaches suggested in the area of CL address CF and the stability-plasticity-dilemma. Most approaches are based on neural networks as those are inspired by the mammalian brain, which is perhaps the best role model and neural networks naturally provide the possibility to step-wise continue training. There exist also non-neural-network methods (e.g. Benavides Prado, 2020; Gepperth & Hammer, 2016), but they are less common as it needs more effort to adapt the systems to enable continuous training. Due to the higher relevance, this work concentrates only on neural-network-based approaches.

The following introduces three general concepts to address the mentioned challenges. Furthermore, specific evaluation protocols and metrics are explained as evaluation is much more complex and case-specific in the continuous setting than for static models.

3.1 Categorization of the Main Learning Strategies

Most CL approaches use neural networks. This has mainly two reasons: First, neural networks became very popular for machine and deep learning, and second, they are highly customizable, adaptable and can be naturally trained continuously, which means a neural network can be trained with initial data and training can be continued with additional data later on. Nevertheless, this naive approach usually works poorly and often results in CF. Therefore, more sophisticated approaches (also called strategies) are necessary for incremental and continuous training of neural networks.

A wide range of strategies are applied to various use cases (i.e., for different CL-scenarios). These strategies can be divided into three main categories: *replay-based methods*, *regularization-based methods*, and *architectural methods*. The categorization is based on where the strategies apply in the training cycle of a neural network (Figure 3.1) (L. Wang et al., 2023). The data features are fed into the neural network model. The model's output is a prediction. Finally, using the prediction and the data labels, the loss function is used to compute the loss and update the model's parameters. The green colored boxes in the figure depict where each of the strategies mainly applies in the learning cycle. All the methods aim to overcome the primary problem of CF while reducing necessary computational and memory resources (Lesort et al., 2020). Replay-based methods try to avoid CF by storing and replaying previously seen data later. Regularization-based methods try to prevent or slow down the change of selected parameters that were important for previous tasks. This is mainly done by additional regularization terms in the loss function. Last, architectural methods address the problems of CL by specific or changing network architectures. Therefore, this approach stays in the context of the model in Figure 3.1. Not all CL-systems can generally be categorized into one of these categories as different strategies are often combined. As each strategy applies to another part of the neural network learning cycle, such hybrid approaches are possible.

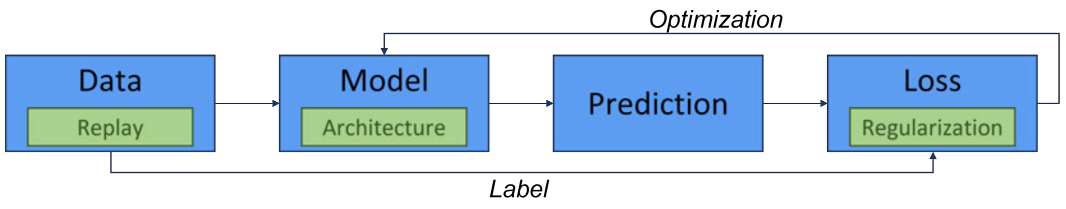


Figure 3.1: Overview of strategies for CL with neural networks and their entry points in the machine learning cycle (adapted from L. Wang et al., 2023).
The blue rectangles depict the main elements of the training process of a CL-system. The green rectangles depict, to which elements the main CL-strategies apply.

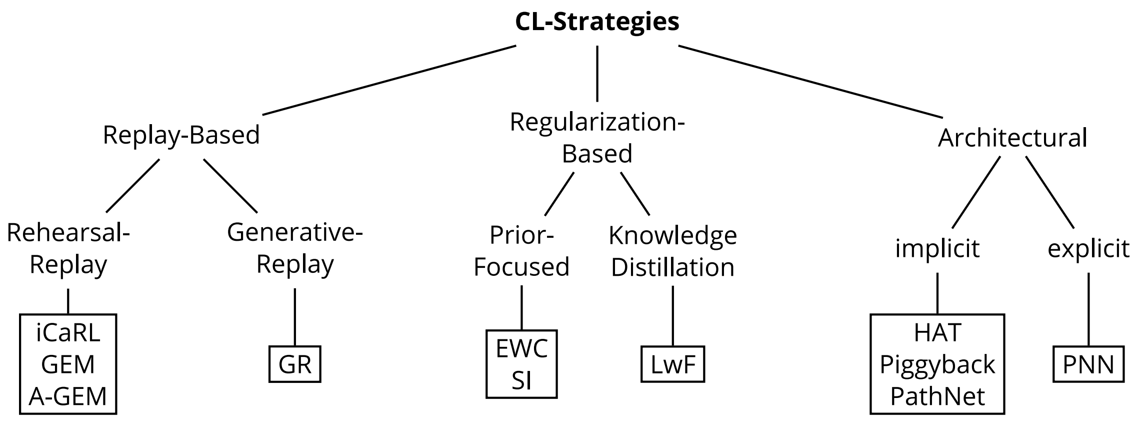


Figure 3.2: Overview of general strategies to address CL and example algorithms.
The boxes denote specific examples of the general strategies addressing CL. All mentioned examples are further described in Section 3.1: iCaRL (Rebuffi et al., 2017), GEM (Lopez-Paz & Ranzato, 2017), A-GEM (Chaudhry et al., 2019), GR (Shin et al., 2017), EWC (Kirkpatrick et al., 2017), SI (Zenke et al., 2017), LwF (Li & Hoiem, 2016), HAT (Serra et al., 2018), Piggyback (Mallya et al., 2018), PathNet (Fernando et al., 2017), PNN (Rusu et al., 2016).

It should be mentioned here that it exists no common definition of the methods established in literature by now. Other categorizations also exist (e.g., Lesort et al. (2020), Parisi et al. (2019), and L. Wang et al. (2023)). However, as most of them can be traced back to the mentioned categorization, this one was chosen in this work. The strategies are overviewed in Figure 3.2. The boxes highlight selected examples of approaches for the respective strategy. More comprehensive overviews containing further concrete approaches can be found in De Lange et al. (2021), Lesort et al. (2020), and Maltoni and Lomonaco (2019). In the following subsections, these strategies and the mentioned example approaches are described in more detail.

3.1.1 Replay-Based Methods

In replay-based methods, the basic idea is to somehow store selected examples from previous tasks to retrain the network on them together with the new tasks data and therefore prevent CF. This retraining is also called *replay* and describes the repeated training of examples during multiple tasks. If the network processes examples from previous tasks, again and again, this reduces CF.

Reply-based methods are inspired by the CLS-theory (Kumaran et al., 2016; McClelland et al., 1995), which describes the process of effective learning of many mammalian species. According to CLS-theory, the learning process consists of two complementary systems represented by the quick learning hippocampus and the slow learning neocortex (Kumaran et al., 2016). Transferred to the replay strategy, the hippocampus is represented by a memory that stores specific examples or the data distribution for all tasks. These examples are replayed later on. The neocortex corresponds to the actual classification system for prediction, which gradually acquires generalized and structured knowledge about all tasks (Kumaran et al., 2016). In this view, replay-based strategies can be seen as a bio-inspired approach (Parisi et al., 2019).

One distinguishes two replay methods: rehearsal-replay and pseudo-rehearsal- or generative replay.

In **rehearsal-replay**, selected examples of previous tasks are explicitly stored in a designated memory to be replayed in later training phases (Biesialska et al., 2020). The goal is to select examples to be stored such that they best represent the data distribution of previous tasks data (Lesort et al., 2020; L. Wang et al., 2023). Therefore, different sampling strategies exist. A simple strategy is to use random selection (L. Wang et al., 2023). More advanced strategies often use optimizable approaches. Aljundi, Lin, et al. (2019) suggest an approach that aims to find a feasible region with reduced constraints (the stored examples) most similar to the feasible region of the original problem containing all data examples. They showed that this can be achieved by a maximal diversity of selected examples in the reply memory. As the optimization approach is computationally expensive, they propose a heuristic greedy method as an alternative. The question is which examples to remove from the buffer to make room for new examples. In their method, examples in the buffer with high cosine similarity to other randomly selected examples from the buffer have a higher probability of being removed, as the method aims to obtain diversity in the buffer Aljundi, Lin, et al. (2019). Therefore, keeping similar elements in the buffer is not constructive.

Next to this storage on data-level, rehearsal-replay can also be done on feature-level. In this case, it is not the complete data example stored, but its extracted features (L. Wang et al., 2023). This reduces needed storage in many cases. This approach's difficulty is managing representation shifts, i.e., the change of the feature extractor due to continuous adaption. As a result, the previously stored features do not align with the current changed feature space (L. Wang et al., 2023).

A general disadvantage of rehearsal-replay is privacy concerns, as raw data is stored for a potentially long time (Lesort et al., 2020). This could already be an exclusion criterion in some use cases.

One of the first algorithms in this category focusing on class-incremental learning is Incremental Classifier and Representation Learning (iCaRL) (Rebuffi et al., 2017). For every learned class, iCaRL dynamically selects a set of exemplar images or data points using a herding-based prioritized exemplar selection. This iteratively selects exemplars such that the feature vector of the selected examples is most similar to the average of all feature vectors of the data points in the respective class. Concerning resource usage, the total number of data points is bounded by a fixed parameter. Besides the sample selection, the algorithm updates the neural network-based feature selection routine during training. For the latter, it also uses regularization-based methods, which are described in the following section (Section 3.1.2) in more detail. For the classification of unlabelled data points, the *nearest-mean-of-exemplars* strategy is used, which determines a prototype vector for each class using the respective exemplar images and compares it to the feature vector of the data point to classify (Rebuffi et al., 2017).

Another representative for replay-based methods is Gradient Episodic Memory (GEM) (Lopez-Paz & Ranzato, 2017). It has a memory for each task to store selected examples. While learning, the underlying network is optimized so that the loss of samples in each task's memory stays constant or decreases. The advantage to other methods (e.g., iCaRL) is that it allows positive backward transfer, which means an improvement on previous tasks while learning the current task (Lopez-Paz & Ranzato, 2017). Although GEM showed good results when only trained a single epoch per batch, it still has a high resource consumption, making it inefficient and unsuitable for common devices. Chaudhry et al. (2019) propose with Averaged Gradient Episodic Memory (A-GEM) a more efficient version. The key idea is to optimize the loss over all task's memories together by averaging the single losses. Therefore optimization problem is reduced from $t-1$ constraints to a single constraint, where t is the number of the current task (Chaudhry et al., 2019).

The **pseudo-rehearsal**, also called **generative replay**, approach aims to train a second generative model next to the actual classification model, which can generate artificial examples of the previous tasks data distribution (Biesialska et al., 2020; De Lange et al., 2021; Lesort et al., 2020; L. Wang et al., 2023). These artificially generated examples can be used for replay in the next step as before. The advantage is that this approach does not need additional memory. Furthermore, it is easily possible to balance generated data using conditional generative models. In this case, examples of a given class $P(X|Y)$ can be generated (Lesort et al., 2020). Nevertheless, it is challenging to continuously train a generative model to represent the data distribution of previous tasks well. Due to a considerable overhead during training, this approach does not scale up well to large datasets (Parisi et al., 2019; L. Wang et al., 2023).

An example of this method is the Deep Generative Replay (GR) approach described in Shin et al. (2017). The approach trains a scholar model for each task which is then used to train a global generator model that can generate samples. The last part is the solver, which learns from real-input data of the current task and artificially generated data.

3.1.2 Regularization-Based Methods

Regularization-based approaches avoid storing data examples of previous tasks to cope with limited memory (De Lange et al., 2021). Instead, they apply regularization on network parameters. A central problem in learning new tasks is the unwanted change of parameters, so old tasks cannot be solved adequately anymore (CF). To face the problem, regularization-based methods are orientated to the mammalian brain, for which it is assumed that it may avoid forgetting by protecting previously learned knowledge in neocortical circuits (Kirkpatrick et al., 2017). Therefore, regularization-based approaches try to prevent parameter changes with negative consequences on the result by introducing regularization-terms to the loss function (Figure 3.1) to limit changes of essential parameters of previous tasks (Lesort et al., 2020; Parisi et al., 2019). This method is also called prior-focused regularization (De Lange et al., 2021) or penalty-computing (Lesort et al., 2020). A second common approach in this group is knowledge distillation, which aims to optimize the complete network (instead of a few selected parameters) for the new task while preventing changes in the prediction for old tasks (Li & Hoiem, 2016; Parisi et al., 2019). As constraints are incrementally added with each task in both cases, training time usually increases over time (Parisi et al., 2019).

The main characteristic of **prior-focused regularization** (also called **penalty-computing**) is to prevent significant changes to parameters that are important to solve previous tasks (L. Wang et al., 2023). Accordingly, if important parameters are not changed, the performance of old tasks should also not change significantly. One of the most essential approaches in this category is Elastic Weight Consolidation (EWC) (Kirkpatrick et al., 2017). It measures the importance of weights (parameters) for specific tasks and adds it as a quadratic penalty loss to the update rule of the network (Kirkpatrick et al., 2017). To cut it short, it aims to prevent changes to vital parameters. The importance of parameters is hereby calculated using the Fisher information matrix (Kirkpatrick et al., 2017). A primary disadvantage of EWC is the disability to learn new classes incrementally. Furthermore, multiple variations and extensions of EWC exist, for instance, using dropout or early stopping, but the main idea to prevent or slow down changes to specific parameters is the same (Lesort et al., 2020). However, in online-learning scenarios, the problem with EWC is that it computes the importance of parameters only task-wise and at the end of each task (offline). Therefore, splitting the stream artificially into tasks would be necessary in an online scenario. This computation of the parameter importance was changed in the Synaptic Intelligence (SI) approach, which continuously (in an online manner) computes the parameter importance, which makes it more suitable for online-learning scenarios (Zenke et al., 2017). Due to the restriction to parameter changes, it comes in particular in prior-focused regularization to a challenging trade-off between learning new tasks and preventing forgetting (Lesort et al., 2020).

In the **knowledge distillation** method, the underlying idea is to transfer knowledge from a big, complex neural network to a smaller network while preserving the performance (Hinton et al., 2015). This means the smaller network predicts the same outputs for given inputs as the original network. The transfer of knowledge is thereby not based on the transfer of parameters. Instead, the resulting simpler network aims to learn a more straightforward mapping from input vectors to output vectors (Hinton et al., 2015). Sometimes, knowledge distillation is also illustrated as a student model that learns to generate predictions of a complex teacher model (Biesialska et al., 2020).

In CL, knowledge distillation can be used to transfer knowledge from the model of the previous tasks and a model trained on the new task together into one new model. The model of previous tasks, together with the model of the newly learned task, can be seen as a complex ensemble neural network simplified into a new (smaller) model (De Lange et al., 2021). Learning without Forgetting (LwF), which was introduced by Li and Hoiem (2016), uses this technique: Given a set of shared parameters θ_s, task-specific parameters of previously learned tasks θ_o and task-specific parameters of the new task θ_n, they optimize the all the parameters of the network to achieve a good performance on the new task while omitting changes to the outputs of previous tasks. As without memory, only the new task's data can be used for training. The authors aimed to prevent changes in the predictions of previous tasks on the new (possibly domain external) data between the old and the new model as this goes hand in hand with the prevention of (significant) parameter changes for the previous tasks (Li & Hoiem, 2016). This mirrors knowledge distillation in LwF. Another usage of the student-teacher constellation is the work of Shan et al. (2020).

3.1.3 Architectural Methods

Architectural methods address CL by implicit or explicit modification of the network's architecture itself (De Lange et al., 2021; Lesort et al., 2020). Both methods aim to separate learned tasks such that new tasks can be learned without interfering with previous tasks (Biesialska et al., 2020; Lesort et al., 2020). In the explicit modification, the network is dynamically expanded when learning new tasks, i.e., new layers are added. Therefore it is also called dynamic network expansion. In contrast, implicit modification has a static, fixed network where other methods like masking separate parameters. As this is not done by regularization terms in the loss function, this method distinguishes from regularization-based methods (Figure 3.1).

In detail, according to their name, **implicit** (fixed network) architectural methods have a static network architecture. Parameters of learned task are separated by activation and deactivation of learning units when learning new tasks (De Lange et al., 2021; Lesort et al., 2020). This prevents CF as parameters of the previous task cannot be changed. On the other hand, when learning new tasks, those units can be reused in the forward pass. This can be understood as the reuse of earlier learned knowledge. Learning units can be weights, neurons, network layers, or others, depending on the specific system. Realized is the activation/deactivation primarily by somehow masking out learning units (De Lange et al., 2021; Fernando et al., 2017).

Algorithms using masking are, for example, Piggyback or Hard Attention to the Task (HAT). In the Piggyback algorithm, the model learns binary masks for the task to mask out neurons to prevent parameters from changing when learning new tasks (Mallya et al., 2018). The network architecture itself stays fixed. As the masks are attached to the underlying network, the authors called it Piggyback. The HAT approach works similarly but uses a different strategy to learn the attention masks (Serra et al., 2018). A different approach is used in PathNet, which tries to find appropriate pathways through the network for new tasks by choosing the pathway to best possible reuse parts of the network (Fernando et al., 2017). PathNet is agent-based, meaning pathways are modeled by agents optimized by a genetic algorithm. The underlying network consists of multiple connected subnetworks called modules. Each module can only be trained once. If it was trained, it could not be changed anymore in the future. This is why PathNet strictly tries to reuse modules for new tasks.

In **explicit** (dynamic network expansion) architectural methods, the network's structure changes over time. As new tasks are learned, the network is expanded by adding new parameters, neurons, layers, or other structures (Lesort et al., 2020; Parisi et al., 2019). Typically, a new part or branch is added to the network for each new task (De Lange et al., 2021). The disadvantage of this approach is a growing number of parameters which can lead to huge networks with high complexity over time (Biesialska et al., 2020; Parisi et al., 2019). Nevertheless, this method also makes it possible (and efficient) to have shared parameters between tasks. For example, one could use a standard feature extraction and add smaller task-specific components on top (L. Wang et al., 2023).

Dynamic network expansion is done in the Progressive Neural Networks (PNN) approach, for example (Rusu et al., 2016). For every task, each layer of the network is extended with additional, task-specific neurons, which, however, learn lateral connections to neurons of other tasks. When training the new task's parameters, all previous tasks' parameters are frozen to prevent performance loss.

3.2 Evaluation Protocols and Metrics

The evaluation of a CL-system is much more complex compared to static classification systems as it is not only the overall performance which is relevant to evaluate the system, but many more features. The fact that there is a wide range and huge heterogeneity of possible CL-constellations and scenarios makes evaluation even more complex as for different systems different evaluation strategies are necessary (Parisi et al., 2019). This is also the reason, why there does not exist a general accepted evaluation strategy for CL-systems by now. Consequently, a meaningful comparison of the different approaches described in the literature is difficult and sometimes impossible.

In general the evaluation of a CL-system should take all aspects of the learning and application problem into account to get an all-encompassing view of the system (Lesort et al., 2020). Thereby, the overall average accuracy is only one of many metrics to measure (Lopez-Paz & Ranzato, 2017). Furthermore, it should be looked at the accuracy on the current, on previous and possibly also on future tasks because the goal of improved performance on all task dissociates CL from transfer learning, were only the performance on the target task is relevant (Lesort et al., 2020). Based on these measures, one can compute metrics to evaluate the forgetting, stability and plasticity of the system and draw conclusions about the ability to transfer and reuse previous learned knowledge (Lopez-Paz & Ranzato, 2017; L. Wang et al., 2023). Another aspect is the consumption of computational and memory resources, as one goal of CL-systems is, to reduce those resources. Next to specific evaluation metrics, it is important to think about when and how often to evaluate a system during the learning process and even how continuously monitor the performance in real world applications to avoid a significant loss of performance. Regarding this, different evaluation protocols exist including approaches to estimate worst case performance which is important for systems in critical applications (De Lange et al., 2023).

In the following subsections, the evaluation protocols and subsequently an ensemble of possible evaluation metrics is described in more detail. An overview of relevant metrics is given in Table 3.2.

		Evaluation Sets		
		$\mathbf{E_1}$	$\mathbf{E_2}$	$\mathbf{E_3}$
	$\mathbf{T_1}$	$R_{1,1}$	$R_{1,2}$	$R_{1,3}$
Train Sets	$\mathbf{T_2}$	$R_{2,1}$	$R_{2,2}$	$R_{2,3}$
	$\mathbf{T_3}$	$R_{3,1}$	$R_{3,2}$	$R_{3,3}$

Table 3.1: Evaluation matrix of a CL-system (based on Lesort et al., 2020; Lopez-Paz & Ranzato, 2017).

After training the system on each training set T_i, the system is evaluated on all evaluation sets E_j with performance metric R. Each row represents the evaluation results after the training up to the respective training set on the evaluation sets E_j for every task.

3.2.1 Evaluation Protocols

An evaluation protocol describes the basic strategy for evaluating a CL-system. This includes specifying the training-, validation- and test data, when and how often to evaluate the system during the ongoing learning process (evaluation periodicity), and which metrics to use with the aim of assessing the system with respect to the whole application.

The most common setting is using separated train and test data for each task. That means two datasets per task exist: a training set and a test set which are usually disjoint (De Lange et al., 2023). The procedure, then, is to train the system on the first tasks training data until the specified first evaluation point is reached, where the system is evaluated on all evaluation sets. Afterward, training is continued on the next training set again, followed by an evaluation on all evaluation sets, and so on. The reason for evaluating on all evaluation sets (even from future tasks) is to be able to draw conclusions about forgetting, stability, and plasticity afterward. As a result, one receives a performance matrix as shown in Table 3.1. In the table, the rows represent the training sets, and the columns the evaluation sets. The values $R_{i,j}$ denote the system's performance on task j after training until the i-th task. For example, $R2, 1$ denotes how the model that was trained on tasks one and two performs on the test set of task one. The performance measure itself can be chosen depending on the problem setting. The accuracy is regularly used, for example.

Until now, it was assumed to evaluate the system after training on the whole training set of each task. This is the most common approach, called *task-based* evaluation (De Lange et al., 2023). Nevertheless, if the training sets are big, this is a relatively coarse evaluation, and it is not known what happens during training and if the performance possibly fluctuates. Therefore, a more fine-grained evaluation can be applied. For example, evaluation can be done after each batch, mini-batch (Lesort et al., 2020) or up to evaluating the system after each m training sample, which is also called *continual evaluation* (De Lange et al., 2023). Consequently, the most detailed result is achieved when choosing m equal to one. As shown in De Lange et al. (2023), continual evaluation is particularly important to determine the worst-case performance.

A third method, which is, in particular, suitable for online learning scenarios when the available amount of data is small, is prequential evaluation (Bittencourt et al., 2019; Gama et al., 2013). In this procedure, a new data point is first used for evaluation by predicting it with the current system. Afterward, the same data point is used for further training (Bittencourt et al., 2019). Consequently, each data point is used for both evaluation and training. Compared to the protocols using a train-test-split discussed before, the disadvantage is that it is not possible to measure the performance on previous data as no data is held out.

3.2.2 General Performance Metrics

General performance metrics express the system's overall performance at a given time. Typically, these metrics are based on accuracy, but other measures like the F1-Score could also replace it.

An intuitive measure is the *Average Accuracy (ACC)*, which is the average accuracy from the first up to the current task t_k (Chaudhry et al., 2018; Lopez-Paz & Ranzato, 2017). $R_{k,i}$ is according to the notation in Table 3.1 the performance (in this case accuracy) of the model trained up to task t_k evaluated on task i:

$$ACC_k = \frac{1}{k}\sum_{i=1}^{k} R_{k,i} \tag{3.1}$$

When the ACC is computed after each task, it gives insights into the development of the performance.

If one single value is preferred instead of many values, Rebuffi et al. (2017) suggest averaging the ACC values from previous time steps. The authors of the paper call it *Average Incremental Accuracy (AIA)*:

$$AIA = \frac{1}{T}\sum_{i=1}^{T} ACC_i \tag{3.2}$$

With the AIA, it is important to remember that it reflects the entire historical evolution of the model, including any outliers in performance, whether positive or negative (L. Wang et al., 2023).

The AIA includes the evolution of the performance, but due to averaging, it gives no information about how the performance developed over time. To get insights about the evolution and how quickly a model learns, the *Learning Curve Area (LCA)* can be used (Chaudhry et al., 2019). In contrast to the metrics before, it evaluates after training with each mini-batch and is more fine-grained. First, the b-shot performance needs to be defined, which is the average performance of the model for each of the T tasks after it has been trained on b mini-batches each (Chaudhry et al., 2019):

$$Z_b = \frac{1}{T}\sum_{i=1}^{T} R_{i,b,i} \tag{3.3}$$

The performance value $R_{i,b,j}$ denotes the model's performance after being trained up to the b-th mini-batch on task i, and it is evaluated on task j. As i equals j in this case, this means the model is always just evaluated on the current task.

Group	Performance Metric	Reference
General Performance	Average Accuracy (ACC)	Lopez-Paz and Ranzato, 2017
	Average Incremental Accuracy (AIA)	Rebuffi et al., 2017
	Learning Curve Area (LCA)	Chaudhry et al., 2019
Forgetting Metrics	Forgetting Measure (f_i^j)	Chaudhry et al., 2018
	Average Forgetting (F_j)	Chaudhry et al., 2018
	Windowed Forgetting	De Lange et al., 2023
Stability and Plasticity Metrics	Backward Transfer (BWT)	Lopez-Paz and Ranzato, 2017
	Forward Transfer (FWT)	Lopez-Paz and Ranzato, 2017
	Relative Overall Performance Ω	Hayes et al., 2018
	Intransigence Measure (I)	Chaudhry et al., 2018
Computational- and Memory Resources	Model Size (MT)	Rodríguez et al., 2018
	Sample Storage Size (SSS)	Rodríguez et al., 2018
	Computational Efficiency (EF)	Rodríguez et al., 2018

Table 3.2: Performance metrics for evaluation of CL-systems.

Metrics to evaluate CL-systems can be divided into four groups. General performance measures are used to assess the performance of the predictions. Forgetting metrics give insights about the amount of knowledge the system forgets from previously learned tasks. Stability and plasticity are helpful to assess the robustness and needed computational and memory resources give information about hardware requirements.

The b-shot performance is then used to compute the learning curve of the model. For this, the b-shot performance Z_b is computed for all $b \in [0, \beta]$. As a result, the learning curve shows the development regarding the performance for the number of seen mini-batches. This reflects how much data the model needs for training to achieve a given performance. In other words, the curve displays how fast the model learns.

Finally, to make learning curves easily comparable, Chaudhry et al. (2019) suggested computing the area under the learning curve as a representative value for the learning speed of a model:

$$LCA_\beta = \frac{1}{\beta+1}\int_0^\beta Z_b \, db = \frac{1}{\beta+1}\sum_{b=0}^{\beta} Z_b \tag{3.4}$$

The higher the LCA_β of a model is, the better it is, as the area under the Z_b curve is greater if the model achieves a high performance with few training samples (small b). Therefore, a high LCA_β score shows that the model learns faster than a low LCA_β score (Chaudhry et al., 2019).

3.2.3 Forgetting Metrics

In the previous sections, CF was discussed as a main problem in CL. Following, it must be adequately monitored to notice the occurrence of forgetting with the goal of minimizing it. To clarify, forgetting of a CL-system is understood as the difference between the maximum knowledge the model had about a task during the learning process in the past and the current knowledge of the model about the same task (Chaudhry et al., 2018). Hereby, the accuracy commonly measures the knowledge, but other metrics could also be used. Accordingly, the forgetting on task t_i after the model was trained up to task k is defined as (Chaudhry et al., 2018):

$$f_i^k = \max_{j \in \{1,\ldots,k-1\}} R_{j,i} - R_{k,i} \qquad \forall i < k. \tag{3.5}$$

Based on the forgetting for each task, the average forgetting F_k of the model trained up to task t_k is computed by

$$F_k = \frac{1}{k-1} \sum_{i=1}^{k-1} f_i^k. \tag{3.6}$$

This means F_k is the average forgetting over all previous tasks learned by the current model. The forgetting is a value in the range of -1 to +1. A great value means that there is much forgetting (catastrophic forgetting), whereas a small value indicates less forgetting. In contrast, negative forgetting means knowledge is transferred from later tasks to previous ones (De Lange et al., 2023). In other words, it indicates that the model can improve the performance on previous tasks by later learned knowledge from other tasks.

De Lange et al. (2023) adapted the forgetting measure in the *Windowed Forgetting* also for data incremental learning, where the data is not task-based. In this case, the forgetting is analogously measured over the last m evaluations, where m is the window size.

3.2.4 Stability and Plasticity Metrics

The third important property of a CL-system is the plasticity of a model, i.e., the ability to transfer knowledge from one task to another to improve the performance on previous or even future tasks when learning the current task. To evaluate such knowledge transfer, the most important metrics are the *BWT* and *Forward Transfer (FWT)* for task t_k (Lopez-Paz & Ranzato, 2017). The BWT reflects the ability to transfer knowledge of the current task to previous tasks by measuring the difference in the performance of the current and previous model states on learned tasks:

$$BWT_k = \frac{1}{k-1} \sum_{i=1}^{k-1} R_{k,i} - R_{i,i}. \tag{3.7}$$

The sum computes for each learned task t_i the difference of the performance of the current model $R_{k,i}$ (trained until current task t_k and evaluated on test set e_i) and the model state directly after learning task t_i. Regarding the evaluation matrix after task t_i (Table 3.1), the BWT averages the element-wise differences of the last row and the diagonal entries while the last column is ignored.

Analogously, the FWT_k reflects the ability of the model to use knowledge of the current task t_k to solve future tasks. If the model has a good FWT, this is also referred to as zero-shot-learning, as the model is able to learn to solve a future task without task-specific training data. Formally, FWT is defined as

$$FWT_k = \frac{1}{k-1} \sum_{i=2}^{k} R_{i-1,i} - \tilde{R}_i, \tag{3.8}$$

where $\tilde{R}_i$ is the performance of a model with randomly initialized parameters on evaluation set e_i (Lopez-Paz & Ranzato, 2017).

Generally, a high BWT and FWT stand for a good ability of the model to transfer knowledge which represents a good trade-off between the stability and plasticity of the model. Stability, on the one hand side, as an improvement on previous or future tasks unlikely occurs together with high forgetting and plasticity on the other hand, as the model is variable enough to improve.

Furthermore, there is the *relative overall performance* Ω_k which measures the performance of the CL-system relative to an offline system that is trained with all data up to task t_k at once (joint training) (Hayes et al., 2018). Joint training is considered as the upper bound of the performance that can be reached (Lesort et al., 2020). The relative overall performance is given by:

$$\Omega_k = \frac{1}{k}\sum_{i=1}^{k}\frac{R_{i,i}}{R_{i,i}^{joint}} \tag{3.9}$$

where $R_{i,i}^{joint}$ is the performance of a system with joint training using all training data up to task t_k at once instead of training continuously. Therefore, the relative overall performance would be one if the continuously trained system achieves the same performance as the joint training system, and the lower (with minimum zero), the bigger the performance gap between both models is. Theoretically, Ω could also get greater than one if the continuously trained system performs better than the joint training system. However, this is usually not the case in practice as continuous training is much more difficult than joint training (Lesort et al., 2020).

The same idea is used in the *Intransigence Measure* introduced by Chaudhry et al. (2018). It is defined as the inability of a model to learn new tasks, which directly represents plasticity. The measure compares the performance of a model trained with joint training and the continuously trained model, but it takes the difference instead of the ratio in the relative overall performance:

$$I_k = R_{k,k}^{joint} - R_{k,k}, \tag{3.10}$$

where I_k is the intransigence measure after the k-th task. The result is also in the range of -1 to +1, but here a small value indicates a good model.

Both, the relative overall performance and the intransigence measure give insights about the plasticity. If the measures indicate just a slight difference between the joint performance and the continuously trained model, it shows that the CL-system has a high ability to adapt to the tasks, which is the definition of plasticity (Chaudhry et al., 2018). Without high plasticity, it would be unlikely for the model to be able to adapt task by task so well that it reaches almost the joint performance in the end. However, it also depends on the tasks themselves, especially their similarities (Lesort et al., 2020). If the tasks are very different, it is more complex, and the model needs a higher plasticity than similar tasks.

3.2.5 Computing and Storage Resources

The metrics explained before are used to evaluate a model concerning its classification performance and the ability to adapt to new tasks or data. Beyond that, it is often of interest to get insights about computational and memory resources the model uses as this reflects the efficiency and makes it possible to compare CL-systems with static models that are retrained in certain intervals.

Those metrics depend on the specific architecture used in the CL-system. Here, three metrics are introduced, giving a basic idea of measuring used resources. The metrics can also be customized depending on the actual system and the environment.

For every model, it is desirable that it does not grow too fast in size. This applies especially to architectural methods where new parameters are added over time. The memory size at task i is quantified in the number of parameters θ denoted as $Mem(\theta_i)$ (Rodríguez et al., 2018). The following formula then measures the growth of the model size (MS) (Rodríguez et al., 2018):

$$MS = min(1, \frac{\sum_{i=1}^{N} \frac{Mem(\theta_1)}{Mem(\theta_i)}}{N}). \tag{3.11}$$

The higher the value (up to one), the less the model grows in size. If the model does not grow (or even shrink) over time, the maximum value of one is reached.

For replay-based approaches, measuring the efficiency of memory usage is also valuable. Rodríguez et al. (2018) proposes to measure the efficiency of the memory usage with the Samples Storage Size (SSS) efficiency defined as follows:

$$SSS = 1 - min(1, \frac{\sum_{i=1}^{N} \frac{Mem(M_i)}{Mem(D)}}{N}) \tag{3.12}$$

The numerator takes the sum of the percentages of each task's memory occupation $Mem(M_i)$ from the memory occupation $Mem(D)$ the whole dataset would need in bits. This sum is then divided by the number of tasks making it interpretable as efficiency.

Similarly, the computational efficiency can be computed using the number of required operations for forward and backward passes as a ground measure. For details, it is referred to Rodríguez et al. (2018).

4 Related Work of Continuous Learning for NLP-Tasks

The topic of CL in the context of machine- and deep learning was a minor research area for a long time. However, with the popularity of large language models in recent years, the topic got more attention due to their high computational requirements and the lack of flexibility in fine-tuning and re-training (W. Zhang et al., 2023). Starting from about 2018, CL became more active, which can be seen in the number of research papers published. An exemplary request on the Elsevier bibliographic database Sciencedirect[3] for the search query *"continual learning" OR "continuous learning"* returned 1,342 results of type review articles, research articles, and book chapters in the computer science subject. 55% of all articles were published from 2020 to 2023, whereas the result list already contains for the first half of 2023 a number of 224 results. However, by now, most research is done on computer vision tasks, including image classification for example, and the results heavily thin out when adding the key phrase "natural language processing"[4] to the query. In this case, the results number reduced to 167, having alone in the first half of 2023 a number of 53 publications and 85% being published in the period from 2020 to 2023.

Similar results shows the search on IEEE Xplore[5] with 668 results for the search phrase *"continual learning" OR "continuous learning"* in the publication topic[6] *learning (artificial intelligence)*, *neural nets* and/or *deep learning (artificial intelligence)*. When the publication topic was changed to "natural language processing" the number of results reduced to only 16, from which 13 publications were published in the period from 2020 to 2023.

The number of results must not include all relevant results, but on the other side, it might also include some irrelevant publications. But the exemplary search queries show that research on this topic got more attention just in the last three to five years. Research about related topics was also done before, but this was mainly more basic research to CF, stability and plasticity of networks also with the view to neuroscience (e.g., French, 1999; Grossberg, 1982; Mermillod et al., 2013; Ratcliff, 1990; Robins, 1995). In contrast, the research in recent times proposes, tests, and compares in large parts approaches to address the challenges in CL. In 2018 the non-profit research organization ContinualAI[7] was founded aiming to connect researchers and to promote research in the field of CL but also to educate.

Comprehensive overviews of CL, its backgrounds and approaches are from De Lange et al. (2021), Lesort et al. (2020), Parisi et al. (2019), and L. Wang et al. (2023). Biesialska et al. (2020) surveyed the topic with the focus on NLP.

[3] https://www.sciencedirect.com/ (visited on July 13, 2023)
[4] The complete search query was in this case *("continual learning" OR "continuous learning") AND "natural language processing"*
[5] https://ieeexplore.ieee.org/ (visited on July 13, 2023)
[6] *Publication Topic* is an available filter in the IEEE Xplore search with a given selection of topics.
[7] https://www.continualai.org/ (visited on July 13, 2023)

The first works addressing the problem of CF were already done in the 1990s by Ratcliff (1990) and Robins (1995). Both had found that learning new information in a CL-setting leads to rapid forgetting of previously learned knowledge and did experiments on so-called base populations, which are just input-output-pairs of binary vectors. Ratcliff (1990) analyzed several different network architectures next to an approach with a replay buffer. They found the replay approach was the only tested approach that could reduce CL. Robins (1995) continued the experiments using replay with the addition that they had the idea of pseudo-rehearsal approaches as this overcomes the problem that previously learned examples must be later available. For the buffer, they created pseudo-items by using random (with constraints) initialized vectors which were passed through the network and its output was assigned as the label of the pseudo-item. Next to differences between specific implementations, they generally found that a higher number of replay improved the results.

One of the first research papers combining NLP tasks with CL is from Blitzer et al. (2006, 2007). They suggested semi-supervised Structural Correspondence Learning for domain adaption to overcome performance losses when a model is trained on one domain but new domains are added later. Therefore, they identified correspondences between the features of different domains and modeled their correspondence by pivots (Blitzer et al., 2006). The approach was tested for POS-Tagging (Blitzer et al., 2006) on parts of the Penn Treebank (Marcus et al., 1993) and for sentiment classification of Amazon reviews of different product types (Blitzer et al., 2007). Even though domain adaption was only from one domain to one and not multiple others, these were first steps regarding CL.

A different approach not built upon neural networks but on Naive Bayes classification algorithm is the Lifelong Sentiment Classification (LSC) suggested in Chen et al. (2015). Their system does not store previous examples but the probabilities for each word to occur in each class and the absolute numbers of words occurring in positive or negative documents. Furthermore, they introduce virtual counts optimized by stochastic gradient descent to improve the classification in newly learned domains. To address the problem of domain-dependent sentiment words, they also added penalty terms to the optimization step. Similar to Blitzer et al. (2007) before, they tested the approach for only learning one new of in total 20 domains treating the other 19 domains as previously seen. The corpus also consists of Amazon reviews of different product types. The results showed that the LSC approach separately optimized for each target domain outperforms an SVM and standard Naive Bayes trained on all training samples, including the target domain.

As the results of the described works are not comparable due to different datasets, later works for task-incremental learning in NLP (D'Autume et al., 2019; Ho et al., 2023; Huang et al., 2021; F.-K. Sun et al., 2020; J. Sun et al., 2020) commonly used the collection of text datasets of X. Zhang et al. (2015). Created initially to do analyses on character-level convolutional networks, it is well suited for task-incremental learning as the collection contains six datasets with classification tasks with two to 14 classes. Namely, these are AG's News and Sogou News (news classification), DBPedia (Wikipedia article classification), Yelp reviews (sentiment and five-star-rating), Yahoo (topic classification), and Amazon reviews (sentiment and five-star-rating) (X. Zhang et al., 2015).

A wide range of approaches has been used for task-incremental learning on this dataset collection. Comprehensive comparisons are from D'Autume et al. (2019) and Huang et al. (2021). Both used the dataset collection to train their models on the different datasets coming with different tasks progressively. They focus on the comparison of multiple replay-based methods. D'Autume et al. (2019) used standard replay with a replay rate of 1%, A-GEM (Chaudhry et al., 2019) and three versions of Memory-based Parameter Adaptation (MBPA) originally introduced by Sprechmann et al. (2018). MBPA consists of a slowly learning but generalizing standard neural network (parametric component) and a fast adapting non-parametric component (episodic memory) which can rapidly temporarily change weights in the neural network (Sprechmann et al., 2018). The memory is, therefore, not used for replay in MBPA. Generally, the approach of using a slow and a fast learning component is based on the CLS orientating on human learning with a long-term and short-term memory (McClelland et al., 1995).

The results for five successively learned text classification tasks in D'Autume et al. (2019) reached an average accuracy from 0.578 using simple replay to 0.706 with the MBPA++ version. The A-GEM achieved an accuracy of 0.669 on average. Clearly, the tested approaches significantly improved the results compared to a naive baseline (accuracy 0.184). The upper bound multitask model reached an accuracy of 0.736 which is 0.030 higher than the MBPA++ model. On the same task, F.-K. Sun et al. (2020) could improve the MBPA++ model with further hyper-parameter-tuning about 0.036 to 0.742. The Language Modeling for Lifelong Language Learning (LAMOL) approach, also suggested by F.-K. Sun et al. (2020), further outperforms this result with an accuracy of 0.765. LAMOL is a generative-replay model replaying pseudo-samples from previously learned tasks. It, therefore, learns to generate pseudo-samples as well as to solve the actual classification tasks (F.-K. Sun et al., 2020). Huang et al. (2021) further compare the described methods to their Information Disentanglemant based regularization (IDBR) approach. The regularization-based method represents texts by combining task-generic and task-specific components using separate regularization to fit the model better. The accuracy of, on average, 0.732 on the same datasets as before showed the model's effectiveness. Additionally, they analyzed the impact of the number of tasks learned and found that the average performance slightly increased with the number of tasks, which in turn shows that the model generalizes well and transfers knowledge from one task to the others (Huang et al., 2021). The underlying models vary from using BERT-base as encoder and key network (D'Autume et al., 2019), only using a BERT-model as feature extractor but an MLP as trainable network (F.-K. Sun et al., 2020) and in Huang et al. (2021) a pretrained GPT-2 model was used as underlying language model.

Just recently, Ho et al. (2023) worked on the dataset collection of X. Zhang et al. (2015), focusing on analyzing replay-related parameters, including the replay sample selection, memory size, and replay rate. Their prototype-guided memory replay (PMR) method uses prototypes for knowledge representation and to guide the selection of examples to store in memory for replay and is combined with a meta-learning framework. Mirzadeh et al. (2020) analyzed the impact of similar replay parameters, however, using image data instead of text data. Further research of replay-based methods, but on image data, was among others done in Aljundi, Lin, et al. (2019), Ramalho and Garnelo (2018), Tiwari et al. (2022), and Xiao et al. (2023).

Most previously described research focuses on task-incremental learning, which differs, depending on the similarity of the tasks, and often suffers more from CF than data-incremental learning (Blitzer et al., 2007). Capuano et al. (2021) analyzed data-incremental sentiment classification using customer communication data of companies for English and Italian language. Their outcomes showed that their results of a continuously trained system using replay generally improved over time. However, it did not reach the performance of joint training. De Lange and Tuytelaars (2021) use Continual Prototype Evolution for data-incremental learning on images. They use evolving prototypes with a nearest-neighbor classification schema in combination with a replay-based approach and a balanced memory. Their algorithm aims to minimize the variance within the classes and encourages the variance between classes.

The current research shows that the best results are achieved when using replay-based approaches, at least when replay is used as one of the possible other components. However, the impacts of replay-based parameters are rarely analyzed, and by now, no study is known that conducts replay-based parameters in the context of data-incremental text classification.

5 Practical Application of Replay-Based CL to Text Classification

In this chapter, parameters affecting replay-based approaches are practically examined in detail at the example of a data-incremental text classification task. It aims, in particular, to get deeper insights about impacting factors of replay-based strategies and to answer the research questions formulated at the beginning of this work.

After a detailed description of the concrete task at the beginning, the methodology is described, including the creation of the used dataset and details about evaluation and programmatic implementation. Subsequently, the experiments will be motivated and introduced in Section 5.3 before the results are shown and discussed in Section 5.4. The chapter closes with a summary of the results.

5.1 Task Description and Objective

The problem that will be explored in this section arose from the research project DeTox[8], which was about the detection of hate speech and toxic language on the internet. The project was done in collaboration with the Hessian Cyber Competence Center[9], a reporting office for offensive language in Germany, to build intelligent models to support the classification process of reported comments.

A major challenge for classification systems for comments on the internet is to continuously keep the systems up to date. Especially on the internet, current topics and language are subject to constant change. This leads to outdated classification models after a short time if they are only static and do not continuously adapt to current topics and language. The present case is an ideal environment for data- or domain-incremental-learning, depending on the view of the problem, as the reporting of comments is an ongoing process that continuously produces new annotated training data. After an initialization period, where data is thoroughly reviewed and annotated manually by human officers, models can be initialized using this data. Afterward, the models support human officers during the review process with suggestions, whereas human officers do a final classification of the comment. Subsequently, for every reported comment, a reliable annotation is naturally produced, which could, in turn, be used to continue the model's training and improve and adapt it. Over a long time, the models would steadily be fed with new training data, hopefully learning the vocabulary of new topics and changes in language, especially in colloquial language often used on the internet.

Due to a lack of adequate long-term hate speech data, the task of data incremental learning for text classification is transferred to another domain. Namely, the problem is examined in the domain of books with the task to classify excerpts and blurbs of books written in different time periods into the three categories *Adventure Stories*, *Love Stories* and *Conduct of Life* by

[8] https://fz.h-da.de/detox (visited on April 11, 2023)
[9] https://hessengegenhetze.de/ (visited on April 11, 2023)

its content. Models were trained starting with the oldest books and continuing with the training using newer books step by step. For this task, an appropriate dataset was created using existing resources. There has been a natural change in the language and topics of written books over several centuries, even in the same categories. The analyses focus on the replay-based method, as related work showed, this is state-of-the-art. In detail, the analyzed parameters are the number of replayed examples, the number of training iterations, and the buffer strategy. It can be assumed that general findings are transferable to the hate speech domain in future work and be helpful in general for CL in text classification.

This section first describes the methodologies (Section 5.2) in detail. In Section 5.3, the experiments and their specific goals are explained before the results are shown and discussed in Section 5.4.

5.2 Methodologies

In this section, the used methodologies are described in detail, including a dataset description and its creation process, preprocessing steps, the evaluation procedure, and details regarding the implementation.

5.2.1 Dataset

For the task mentioned above of text classification in the domain of books, it was necessary to create a dataset containing books over a long time (several hundreds of years) to ensure significant changes in the language and writing style as well as in topics. Additionally, annotations regarding content categories were required for the present supervised learning task.

To fulfill the requirements, two different public sources were used: First, the Project-Gutenberg[10] which is a collection of old books up to about 1975 and its metadata including classifications. Second, to receive data about current literature, the Goodreads web page[11] was used. Goodreads claims to be the world's largest site for readers and book recommendations (Goodreads LLC, 2023). As current books are not free of charge, only blurbs of those books were used. For the book's language, it was decided to use English as the Gutenberg-Project has with a number of 55,753 books by far the most books in English language compared to 2,064 German books, for example. The collection also contains books that are only partly written in English mixed with other languages. However, those books were not used to minimize noise in the data.

In the following subsections, the creation process of the dataset will be described in detail before a statistical overview of the resulting dataset is presented.

[10] https://www.gutenberg.org/ (visited on May 9, 2023)
[11] https://www.goodreads.com/ (visited on May 9, 2023)

5.2.1.1 Creation Process

As mentioned before, dataset sources were the Gutenberg-Project and the Goodreads-Webpage. Because the Gutenberg-Project is the more limited resource, book categories or subjects which should be classified in further work were selected based on the number of books for existing subjects, the distribution of those books over time, and finally on the overlap of subjects in selected books. In detail, the books can be classified into multiple subjects (multiclass). However, a multiclass task is different from the goal of this work. Therefore, classes with low overlap were preferred, and books classified into more than one subject of the finally selected subjects were dropped to allow a precise classification.

5.2.1.2 Data Selection

Gutenberg-Project provides subject classifications of the books by the *Library of Congress Subject Headings (LCSH)*. LCSH is a comprehensive list of subject headings maintained by the Library of Congress[12], to organize books by its contents. The current 44th version of LCSH contains 355,980 topic subject headings organized in a flat hierarchical structure (Library of Congress, 2022). Table 5.1 lists the most frequent topics of English books. Different kinds of fiction are most present, followed by short stories. Both are coarse categories that do not precisely narrow topics. Instead, they cover a wide range of topics and often co-occur with other subjects. Therefore, both are unsuitable for subject classification. Following subjects in the frequency ranking are Adventure Stories, Juvenile Fiction as a subcategory of Conduct of Life, and Love Stories, which are subjects that are well separated from each other regarding the content. Further analyses showed that the coarse category Conduct of Life also occurs with other subtopics resulting in 1,401 books which is favorable for the present case. The same, but not at this scale, is valid for the categories Adventure Stories and Love Stories, such that the number of books for these subjects also slightly increases. In detail, 1,277 books belong to Adventure Stories and 742 to Love Stories. Further, the Venn Diagram (Figure 5.1) shows that there is overlap between those three subjects, but to a low extent. Conduct of Life and Love Stories have only an overlap of 7 books. The overlap between Adventure Stories and Conduct of Life is with 125 books much greater. Nevertheless, both subjects also contain more books than Love Stories. In total, 206 books in these three subjects had overlapped and were therefore removed in the further process.

The third criterion for the selection was the distribution of books per topic over time. As the metadata for the books does not contain the publishing years, the time of publishing was approximated from the author's lifetime. Concretely, if birthdate and death date were included in the Gutenberg Metadata, the average of both rounded to the next full year was used as the approximate publishing year. If just one of these dates was given, this respective date was used. If no year was given or the author was not specified, the book was dropped as it is unsuitable without a temporal classification. This concerned 494 books in total. The reason for this high number is that books written by multiple authors often have an empty authors field, so authors cannot be assigned automatically. Parsing the authors from the book's content could theoretically be done. However, as the books have very different structures,

[12]https://www.loc.gov/ (visited on May 9, 2023)

Rank	Subject (LCSH)	# Books
1	Science Fiction	2693
2	Short Stories	2474
3	Fiction	1416
4	Adventure Stories	1231
5	Conduct of Life –– Juvenile Fiction	856
6	Love Stories	719
7	Detective and Mystery Stories	717
8	Man-Woman Relationships –– Fiction	707
9	Historical Fiction	568
10	English Wit and Humor –– Periodicals	556

Table 5.1: Top 10 LCSH-Topics in the English part of the Gutenberg-Project-Corpus.

English language books of the Gutenberg-Project-Corpus were ranked by the number of occurrences of LCSH-subjects. The table lists the top ten subjects in 55,753 books. Double hyphens (––) indicate subtopics.

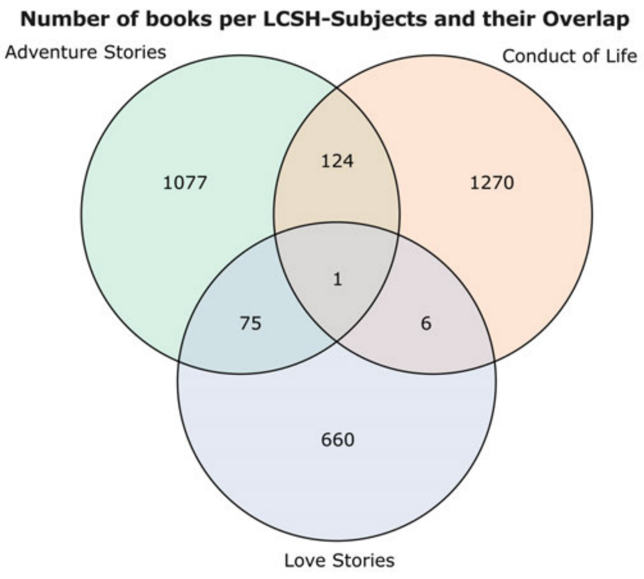


Figure 5.1: Overlap of classes.
The Venn-Diagramm shows the number of books contained in each of the categories only and the number of books assigned to multiple categories (overlap of circles).

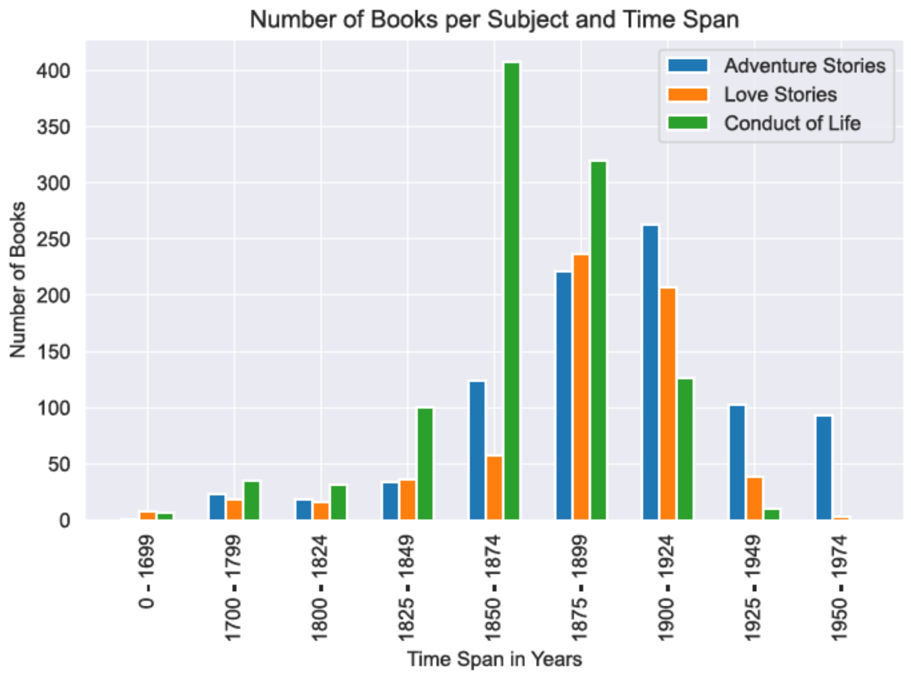


Figure 5.2: Distribution of Gutenberg-books over time.
The diagram depicts the number of suitable English language books for the three selected LCSH-subjects. Books with overlapping subjects and where the year could not be inferred were removed. In total, there are 2537 books left. There are no books after 1975.

it would take much work to parse authors. For this reason, books without given authors in the metadata table were dropped. To conclude, the number of books over the time span from year zero to 1974 is depicted in Figure 5.2. The chosen time intervals have a width of 25 years, except the first two intervals are wider, as only a few books from before 1800 are contained. For simplicity, the considered time intervals are numbered from one to eleven in the following (Table 5.3) and will be referred to as tasks. The term *task* in this context is explained in more detail in Section 5.3.

5.2.1.3 Crawling of Blurbs

In the second step, current literature for the same subject was crawled from the Goodreads web page. As current books are not open source, only blurbs were used. Goodreads does not use LCSH-subjects but organizes books on several shelves. The shelf *Romance* was taken as an equivalent to love stories, *Self-Help* as an equivalent for conduct of life, and a shelf *Adventure Books* did exist. Via these shelves, blurbs of respective books and their metadata, including publishing year, were crawled. Blurbs assigned to more than one of these subjects were removed. This resulted in 1,097 blurbs from the subject romance (love stories), 991 blurbs of self-help (conduct of life), and 823 blurbs of adventure stories (Table 5.2). In the following, the LCSH-subject names will also be used in relation to the blurbs instead of the Goodreads shelf names.

5.2.1.4 Text Extraction from Books

he last step was to extract smaller excerpts from the books. Due to the usually long length of the books, multiple excerpts were extracted from each book. Aiming to create a balanced dataset over the subjects and ideally also over the time, the number of excerpts to be extracted from each book was determined depending on the subject and time interval of a book, i.e., in time intervals with fewer books, more excerpts were extracted from each book than in time intervals containing a higher number of books to balance the number of excerpts in each time interval. It was targeted to have 1,000 excerpts or blurbs per time interval and subject. On this basis, the number of excerpts per book to extract was calculated. However, the maximum number of excerpts extracted per book was set at 20 to avoid biases in the final dataset. This approach allowed the number of excerpts to be balanced across subjects. Nevertheless, a good balance over time was not possible due to significant differences in the number of books per time span (Figure 5.2).Throughout the process, the number of blurbs per subject and time interval were also considered. In the following use of the dataset, it will not be further divided between extracted book excerpts and blurbs if it is irrelevant. Both will be referred to as *texts*.

For the extraction of excerpts from the book, the length of the excerpts was chosen based on the length of the blurbs to receive a homogeneous dataset. The average length of the crawled blurbs is 1,029 chars with a standard deviation of 467 chars. Under the assumption that the distribution of the blurb length is a normal distribution, about 68% of the data lies in the range of plus minus one standard deviation from the mean. For this reason, the excerpt length was chosen to be 1,500 chars which is approximately the sum of the mean and standard deviation. Using the higher length ensures to have long enough excerpts. For each book, the extracted excerpts were evenly spread over the whole book, beginning to extract each excerpt after a full stop, exclamation mark, or question mark, which usually indicate the end of a sentence.

5.2.1.5 Final Dataset and Train-Test-Split

With the described process, a dataset of 16,205 texts was created, from which 13,294 are excerpts from books and 2,911 texts are blurbs (Table 5.2). The dataset was split into training and test data, using 80% for training and 20% for testing (Table 5.3). The split was done such that the train-test-ratio is approximately kept for every time interval, and the subjects are again balanced in the train and test data for each time interval. The split could not be done with the exact ratio and perfectly balanced subjects because of the introduced constraint that all excerpts of books from one author are either in train or test data, but not both. Accordingly, excerpts from one book are also either in train or test data. In the following the time intervals will be referred to as task to use typical wording from CL. The task numbers from one to eleven are assigned as shown in the first two columns of Table 5.3.

The distribution of texts distinguished by the subject and source (Gutenberg-books or blurbs) is depicted in Figure 5.3. The diagram shows that the excerpts from books mainly cover the time until 1949, whereas from 1950 the dataset contains mainly blurbs.

	Adventure Stories	Love Stories	Conduct of Life	Total
Book-excerpts	4447	4556	4291	13294
Blurbs	823	1097	991	2911
Total	5270	5653	5282	16205

Table 5.2: Numerical overview over the complete dataset.

The table shows the number of texts in each category in the complete dataset. The number is broken down into the number of excerpts from Gutenberg-books and crawled blurbs.

Task-Nb.	Timespan	# Total	# Train	# Test
1	**0 - 1699**	76	67	9
2	**1700 - 1799**	957	756	201
3	**1800 - 1824**	844	667	177
4	**1825 - 1849**	1786	1428	358
5	**1850 - 1874**	3086	2444	642
6	**1875 - 1899**	3354	2689	665
7	**1900 - 1924**	2771	2218	553
8	**1925 - 1949**	579	462	117
9	**1950 - 1974**	100	82	18
10	**1975 - 1999**	365	291	74
11	**2000 - 2024**	2287	1832	455
	Sum	**16205**	**12936**	**3269**

Table 5.3: Train-test-split of the complete dataset.

The train-test-split was done aiming to achieve a ratio of 80% train data and 20% test data. The ratio is approximately kept in all time intervals, and subjects are evenly distributed.

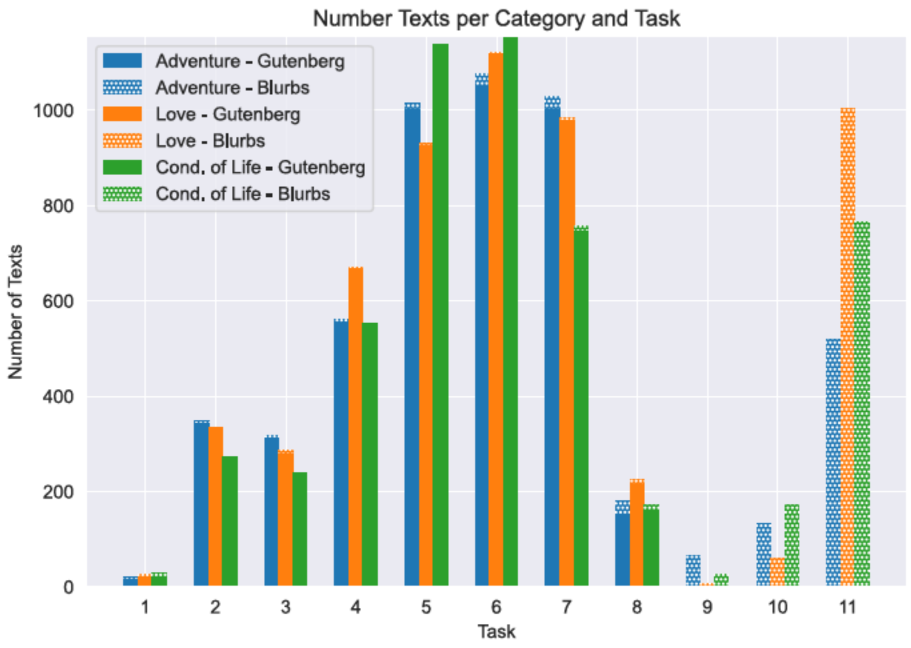


Figure 5.3: Distribution of the complete dataset.
The height of the bars depicts the number of texts per category and task. The filled parts of the bars represent the proportion of texts from books, whereas the dotted filling indicates the amount of blurbs.

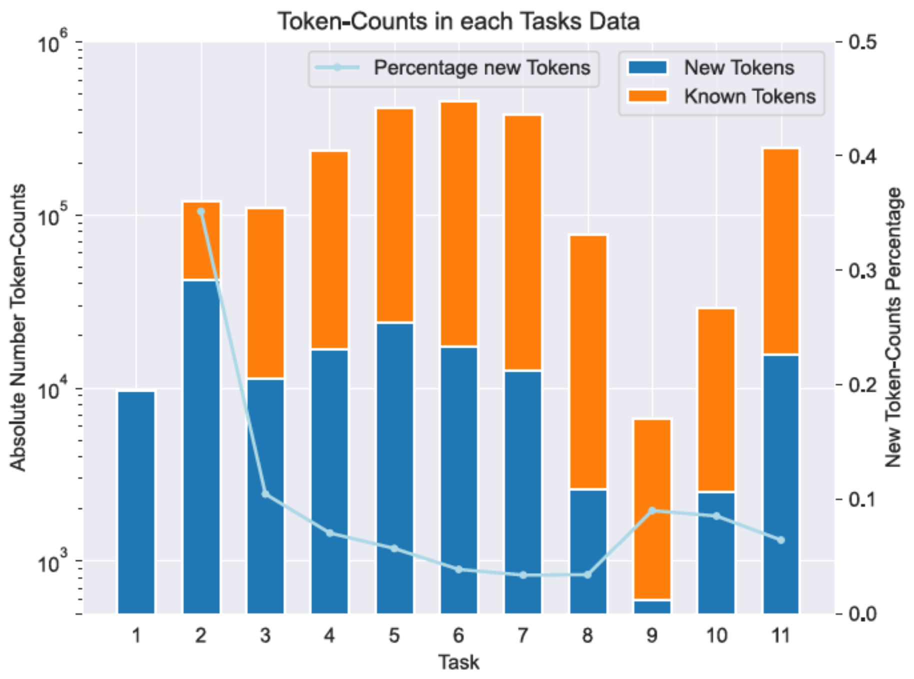


Figure 5.4: Token counts in the complete dataset.
The blue parts of the bars depict the absolute number of occurrences of new tokens in each task's data compared to the data of previous tasks. The total bar height shows the total number of tokens in each task's data. Furthermore, the line plot denotes the percentage of new tokens in each task's data based on the total token count of the task's data.

For the present case, how the data distribution develops over time is also of interest. This is analyzed regarding new tokens appearing over time (Figure 5.4). The bars in the diagram show the total number of tokens for each task (left y-axis, log-scaled). For each bar, the blue colored part depicts the proportion of new tokens first appearing in this task. The percentage of new tokens from the total number of tokens in the task's data is depicted by the line plot on the right y-axis. For better representation, the percentage for the first task is not plotted, as all tokens are new, which results in 100% of new tokens. Except for the first two tasks, the percentage of new tokens per task lies between 3.3% for task seven and eight and 10.4% for task three. At the beginning, the percentage of new tokens decreases before it flattens around tasks seven and eight. After task eight, the curve jumps up to almost 10% before it slightly decreases until task eleven again. The reason for the increase from task eight to nine is the change in the data from book excerpts to blurbs as seen in Figure 5.3. As blurbs are written differently from the actual content of books, this naturally goes with a higher percentage of new tokens.

5.2.1.6 Additional Data for Word Embeddings

The following experiments use the previously described dataset for training, validation, and test purposes. However, as for all experiments word embeddings are used, an additional dataset to train the word embeddings is needed. The previous dataset cannot be used to train word embeddings because in an CL setting at the beginning only a small amount of books is available for an initial training. Step by step, the model sees further data examples used to continue the model training. The initial training data would not be enough to train word embeddings, and later data examples cannot be used to train word embeddings before they are visible for the model too. These circumstances make it necessary to build another corpus of texts that can be used to train word embeddings. The restriction hereby is that the corpus can only contain texts from before the first time interval in the training dataset. Otherwise, the word embedding corpus might contain words that arose later in time and would not have been known at the beginning of the training phase. Simply put, the word-embedding corpus would be an oracle able to know words that arise in the future.

For this reason, a separate word-embedding dataset fulfilling the mentioned constraints was created. As the dataset before, it was built from books from the Gutenberg-Project. This assures that it contains the same type of text, in this case books. The book categories are not of interest for a word-embedding training corpus. Therefore, the books were not filtered by their categories. The only restriction was that the year of the book had to be 1600 or before, where the year of the book was determined as previously described in Section 5.2.1.2. The year 1600 is not before the first time span, which starts in year zero, but as the amount of 76 books in the first time span is small and second to have a corpus large enough to train word embeddings, the year 1600 was chosen as the boundary. The resulting word embedding corpus contained 845 books in the English language.

5.2.2 Data Preprocessing

A basic preprocessing was done on both datasets (training/test dataset and word-embedding dataset). For all further tasks and experiments the preprocessed data was used. For English texts, as in the present case, the Python package NLTK (Bird et al., 2009) offers many functions for text preprocessing, which were widely used in the preprocessing step.

At first, the texts were tokenized, followed by the removal of the last token in case the text was a book excerpt, as the excerpts were extracted by a fixed number of characters which led to the cut-off of words at the end. Then the texts were converted to all lowercase before the words were lemmatized using NLTKs Word-Net-Lemmatizer. In the last step, stop words, punctuation, and special characters were removed. For stop words NLTKs stop word list for English was used. The result formed the basis for all further tasks.

5.2.3 Evaluation

The evaluation was done using predictions of the models on the test data. Generally, k-fold cross-validation should be preferred to a static train-test-evaluation to overcome possible biases in the test dataset. The cross-validation was not performed for two reasons. First, it is challenging to find different train-test-splits for each fold in the cross-validation, as it needs to be ensured to fulfill the constraints to have each author either in the training dataset or the test dataset, and the class balance must be kept as good as possible. Second, cross-validation efforts higher computational resources due to repeated training which were not available. To still receive robust results, each run was repeated with the same parametrization five times, and the results were averaged over the five runs.

Generally, the F1-Score was used as the primary performance measure. It was preferred over accuracy as it provides more meaningful results if datasets are unbalanced. Even though the present dataset is class-balanced over large parts, it shows imbalances in tasks five, seven, ten, and eleven. Depending on the experiment, the results were analyzed regarding average F1-Score, task-wise evaluation, the range between the performance of the best and worst performing class, and forgetting and BWT. All measures are based on the F1-Scores. The average F1-Score is always only measured on the tasks already trained, i.e. if the model is trained until task five, only the first five tasks are included in the average F1-Score. This is close to real-world scenarios. An exception are task-wise analyses, where the performance on each task is measured separately. The range between the best and worst performing class at each task is essential to assess if all classes have an equal performance (small range) or if one class has a very high performance whereas another class performs much worse.

5.2.4 Implementation

For all experiments, Python was chosen as the programming language as it offers many possibilities for machine and deep learning scenarios. A primary reason for Python was furthermore the existence of the CL-framework Avalanche (Lomonaco et al., 2021), which is an open-source end-to-end library for CL based on PyTorch mainly developed by members of the ContinualAI non-profit research organization. Due to the complexity and the effort to implement all learning strategies from scratch, Avalanche vastly simplifies the whole development cycle. It consists of five modules: The Benchmarks-, Training-, Model-, Evaluation-, and Logging-Module. Starting with a Benchmarks-Modul, it provides benchmark datasets ready to use for continuous learning. Due to limited resources for text classification, the selection of available datasets is mainly limited to computer vision benchmarks by now. However, it also provides base classes for datasets and data loaders, such that custom datasets can be integrated and prepared for CL-tasks with affordable effort. The Training-Module provides by the time of writing (Avalanche version 0.3.1) this thesis 22 learning strategies, which can be combined with pre-defined models from the Models-Module. All strategies can be customized, and they can be combined as far as the strategies are not in conflict with each other. Additionally, all PyTorch models can be connected to the framework. Finally, the Avalanche provides valuable tools for evaluation and logging during training with the Evaluation- and Logging-Modules. More details about Avalanche can, among others, be found in Lomonaco et al. (2021).

5.3 Experiments

In the following subsections, the performed experiments are described in detail before their results are shown in Section 5.4. In the beginning, two comparative experiments were done to assess all further experiments better. The first is cumulative training which can be seen as the upper bound of the achievable performance with CL-approaches, and the second, a simple naive strategy is used as a baseline experiment which is used to see how replay can improve the results compared to the naive baseline.

In the following subsection, general settings applying to all experiments are explained before the experiments are introduced separately, including backgrounds and hypotheses about expected results.

5.3.1 General Settings

For all experiments, which will be described in detail in Section 5.3, an MLP and an LSTM are used as underlying models. The MLP was chosen as most CL-approaches are based on neural networks, but it was designed to be still a simple and less complex model (15,303 trainable parameters). Unless otherwise specified, the MLP was used with a single hidden layer containing 75 neurons and a dropout of 0.35. As an activation function, ReLU was used, and it was trained with an Adam-optimizer and a constant learning rate of 0.001. Second, LSTMs are well suited for sequential tasks, as is the case in text classification, as they are able to capture long-term dependencies (i.e., dependencies between words that are not close to each other in the word order in the sentence, Otter et al. (2021). It is expected to achieve better results compared to the MLP, but the LSTM also has higher computational requirements. The LSTM was parameterized with a maximum sequence length of 250, one LSTM-layer with a hidden size of 64 neurons and 0.1 dropout in the input layer resulting in 68,291 trainable parameters. The Adam-optimizer with a learning rate of 0.001 and the cross-entropy-loss-function was used in the LSTM-training process.

For all experiments, the documents are represented by fastText (Bojanowski et al., 2017) based embeddings. FastText word embeddings were trained on the word embedding corpus (Section 5.2.1.6). In detail, a skipgram model with a minimum number of three occurrences of each word, a context window size of five words, and a vector size of 200 dimensions was trained. The advantage of fastText word embeddings is that fastText can handle out-of-vocabulary words by using character n-grams. This means even words not contained in the word embedding corpus can be mapped to word vectors (Bojanowski et al., 2017). This works well for compound words that consist of character n-grams of words with coherent meanings. Neologisms not including related character n-grams might therefore be not necessarily represented as meaningful word vectors by the model. To be able to represent out-of-vocabulary words is, in particular, for the present problem of high importance as new words occur over time. Having a static word embedding model that can vectorize out-of-vocabulary words therefore simplifies the process, as the embeddings do not need to be updated over time. This would be a non-trivial problem in CL and would need to be analyzed separately.

For the detailed embedding, it must be distinguished between the MLP and LSTM. For the MLP, the vectors of each word in the preprocessed text were determined and averaged over each text. As the word embeddings are 200-dimensional, this results in a 200-dimensional document vector. The document vectors were then directly used as the input for the model. The LSTM was provided with sequences of word embeddings of the words/tokens in the documents. The maximum sequence length was here set to 250 tokens, whereas most documents had fewer tokens. To conclude, the input for the LSTM was a matrix of a maximum of 250 rows (maximum sequence length) and 200 columns (fastText word embedding dimension)

Experiment	# Replay	# Iterations	Buffer	Details
Cumulative Training	—	—	—	Section 5.4.1
Naive Training	0	2	—	Section 5.4.2
Number Replay	[0, 1, 3, 5, 7, 10, 15]	2	task-bal., rand. selection	Section 5.4.3
Number Iterations	7	[1, 2, 5, 7, 9, 15]	task-bal., rand. selection	Section 5.4.4
Buffer Constellation	7	2	refer to exp. description	Section 5.3.6
Optimized Parameters	15	5	task-bal., close2ctr	Section 5.4.8

Table 5.4: Overview of performed experiments.

Overview of parameter values used in each experiment. For all experiments, except cumulative and naive training with no buffers, the memory size was chosen to be 647 texts (5% of the training data), and the data batch size was ten. Each experiment is described in detail in the referenced section. The experiments number replay and number iterations used task-balanced buffers with random selection of elements to store in the buffer. The run with optimized parameters used a task-balanced buffer with close-to-center selection strategy. Details about the buffer strategies are given in Section 5.3.6.

for each document. Additional features to the word embeddings were not used but could be added to improve the results. As this work focuses on CL, no additional feature engineering was done.

For the training of all models, the described split of the data into eleven tasks was used. As the time span of each task is small, and the evaluation is task-wise, the examples were pseudo-shuffled within each task. Pseudo-shuffling with always the same seed was used to have the same conditions in each experiment. Each task was then split into batches of ten texts each. Then the model was successively trained on the batches of each task.

A replay buffer was used in all experiments except for cumulative and naive training. The maximum buffer size was chosen to be 647 texts in all experiments, which corresponds to 5% of the total training data. The buffer was furthermore adaptive. This means it is completely filled during the first iterations and elements are removed to make space for elements of new training data during the further learning process.

An overview of the parameterization of the performed experiments is given in Table 5.4. The number of replayed elements (# replay) denotes the number of examples from the buffer that were replayed for each batch of new training data, i.e., the number of replays per ten new examples. Elements from the training data stored in the buffer are selected by a particular strategy or randomly (rand. selection). Furthermore, the buffer can be balanced regarding tasks (task-bal.) or classes. Details about buffer constellations are explained in the respective experiment descriptions and in Section 5.3.6. For replay, examples from the buffer are sampled randomly as this is common sense in replay-based methods (Aljundi, Belilovsky, et al., 2019). The number of training iterations (# iterations), also called training passes, is the number of passes the neural network is trained on the subset of data before the next data batch is shown to the network. It is equivalent to training epochs in the training of static neural networks. It was set to two in most experiences. In other words, if the number of training iterations is, for example, two, every batch containing ten new data points is shown twice to the model for training purposes before the model is trained on the next batch.

5.3.2 Cumulative Training

In order to better assess all the following results, in the first experiment cumulative training was performed. Cumulative training trains the model for each task on the whole training data of the previous and current task at once. Therefore the models do not build on each other, instead they are retrained. Therefore, cumulative training produces static models (as in classical machine learning), but it can be seen as an upper bound for the performance of CL-models (Lesort et al., 2020). That means, due to higher constraints in CL-models, it is commonly expected that CL-models perform worse than static models, which saw all the training data at once.

For this experiment, the MLP and LSTM were used as described before. Additionally, an SVM was used for comparison purposes. SVMs are classical machine learning models but still they often achieve comparable results in text classification tasks (e.g. Struß et al., 2019). The difference to neural networks is that the result of an SVM is not dependent on the initialization, but with the same training data it mathematically computes the optimal solution leading to the same result if trained multiple times.

5.3.3 Naive Training Baseline

Naive training was chosen as a baseline for CL on the described task. Naive training means that no particular strategy is used to address the problems of CL. Instead, the training on the network is just continued with new data samples and no buffer is used. It is the most straightforward approach to update the model continuously. Naive training was tested with two different models: an MLP and an LSTM. The SVM could not be used in this case, as it is not based on a neural network, making it impossible to transfer this strategy to an SVM and requiring other approaches. The parameters regarding the MLP and LSTM, as well as the embeddings, were used as described before in Section 5.3.1.

The outcome of the experiments was evaluated after the model was completely trained on each task's training data. First, the average performance using the F1-Score over the classes was measured over the tasks. After each task's training, the models were evaluated on all tasks up to the current one. In Analogue, the progress in the performance for each of the three classes is evaluated. Furthermore, the performance was analyzed separately for each task. This gives insights about the performance of each of the eleven tasks over time. This makes it also possible to draw conclusions about the mutual influence of the task. Last, the average forgetting and BWT were visualized.

As each data example is only seen twice by the network (two train passes) and no replay or regularization is applied in naive training, it is expected that the trade-off between stability and plasticity will not be balanced, resulting in an unstable model and partly forgetting of learned knowledge. This could be visible in decreasing performance of some tasks and a fluctuating curve of the average F1-Score over time. The average performance is therefore also expected to be significantly lower compared to the cumulative trained models. As before, the experiment was repeated five times to get reliable results and to be able to measure the standard deviation over the runs. The latter is particularly relevant as fluctuation is expected. Therefore it is likely, that the results clearly differ in some sections, which would be visible by a high standard deviation.

5.3.4 Number of Replayed Examples

In replay-based approaches, the question arises of how many of the examples stored in the replay buffer should be replayed in each training step which also depends on the number of new examples per training step. As the number of new examples per training step is set to ten for all experiments in this work, in the further description and analyses only the number of replayed elements will be mentioned.

In the naive approach before, no examples were replayed. The opposite would be to replay all elements stored in the replay buffer in each training step. However, if the number of new instances in each training step is small, as in the present case, this leads to substantial computational effort depending on the buffer size.

Therefore, this experiment aims to get insights into the correlation between the ratio of replayed examples and the resulting performance. Using replay has the goal of overcoming performance decreases in previous tasks when learning new tasks. On the other hand, there might be a threshold where a higher number of replayed examples per training step does not further improve performance.

For the experiment, a task-balanced buffer of size 647, which is 5% of the total number of training examples, was chosen. This means the buffer can store a maximum of 647 elements. As it is task-balanced, these 647 elements are proportionally divided to the number of examples in the already seen tasks, i.e., from tasks with a higher number of examples, more elements are buffered than from tasks with a low number of examples. Furthermore, the buffer was designed as an adaptive buffer which means, in the beginning, all elements are stored until the buffer is completely full. From this point on, examples are removed step by step to maintain the task ratio while new data arrives. According to the ratio, new data points can thus be added. The buffer could also be split into fixed numbers of examples per task as the total number of tasks and examples is known. However, the adaptive behavior is more realistic for real-world scenarios as the number of new tasks is commonly unknown initially. Finally, regarding the buffer, the selection strategy of new elements plays a role. In this case, the elements were selected randomly. That covers training examples that are added to the buffer as well as the elements that are removed from the buffer are selected randomly within the elements of each task. It is noticeable that this strategy may result in an imbalance of classes in the buffer. However, due to the law of large numbers, a random selection will, over the long term, result in the same class balance in the buffer as in the training data.

For the experiment, the parameter of replayed examples was varied between the values 1, 3, 5, 7, 10, and 15. Additionally, 0 replayed examples, equivalent to no replay (naive training), was added to the diagrams for better comparison.

The results were analyzed under consideration of the F1-Scores after each task. Furthermore, it was looked at the range between the F1-Scores of the best and worst performing class at each task. The background is that all three classes should ideally have equal performance. Therefore the range between the best and worst performing classes should be low. Third, as before, the forgetting and BWT are considered in the analysis.

For the result, the performance is expected to increase with the number of examples replayed, as the forgetting should decrease. As a result of lower forgetting, the performance graph should stabilize, which means that it shows less fluctuation, as sudden performance drops should be dampened. The goal is to get insights about the number of examples to replay necessary to achieve a visible improvement in the performance and if there is an upper bound where replaying more examples does not lead to further improvements.

5.3.5 Number of Training Iterations

In the training of neural networks, the number of training epochs is a parameter that defines for how many iterations the network should be trained to fit the network's weights. Typically, in the first training epochs, the loss decreases faster before the loss curve flattens and converges.

Analogue, in CL, the number of training iterations per batch can be varied. This is the number of passes the neural network is trained on the subset of data before the next data batch is shown to the network. The number of iterations is relevant, as it decides how often the network can see each data sample besides replay. If the number of iterations is too low, the network might not be able to fit its weights adequately. If the number of iterations is too high, the model might more likely forget previous knowledge and overfit on the last batch due to too much change in the weights. As also the replayed examples are trained together with the new examples for the chosen number of training iterations per batch, it needs to be analyzed together with replay and the results could vary when no replay is applied. In the latter case, the number of training iterations might be higher because, due to missing replay, this is the only time the network sees the examples. For this reason, it is essential to analyze the number of training iterations separately in the case of replay.

Six different numbers of training iterations, ranging from one to 15, were examined in the experiments. The number of examples to replay was set to seven, as this is in the middle of the tested numbers of replay in the previous experiment. Therefore, it is known from the previous experiment that it performs better than no replay, but it still has some shortcomings. If the number of examples for replay had been chosen according to the best performance achieved in the previous experiment, it would have been harder to see the effects of varying numbers of training iterations, as there would have been little room for improvement.

As before, the experiments were analyzed regarding the average performance, the range between the best and worst performing classes, forgetting, and BWT.

5.3.6 Impact of the Buffer-Strategy

A potentially significant impact in replay-based methods has the replay buffer and the selection of the elements from the training data to be stored in the buffer. Therefore, it is of relevance which elements of the current training examples are newly added to the buffer and which elements from the buffer are replaced by the new elements. The replacement strategy is only relevant after the buffer is completely filled. In this work, only adaptive buffers are examined. This means the buffer of its complete size is used at any point to store selected examples. If new tasks or classes are added during the learning process, the examples from the buffer are removed to make space for new elements. In contrast, non-adaptive buffers specify beforehand the number of examples to store for each task or class. Respectively, there is no need to remove elements from the buffer at any time. However, in practical applications, the latter has the disadvantage that either the buffer size grows (potentially infinitely) over time due to new added tasks respective classes or the number of tasks or classes must be specified before the start of the learning process, which is in turn rarely practical in natural CL-applications. In adaptive buffers, all training data is usually added to the buffer at the beginning until it reaches the specified maximum size. This is the point where it becomes of interest which elements to add and which to be replaced in the buffer.

Generally, the buffer is influenced by two factors. First, it is interesting whether the buffer is balanced in some way or not. Possibilities to balance a buffer are primarily class-balanced and task balanced. This means the amount of samples in the buffer is proportional to the number of classes or tasks. Class and task balancing can furthermore be combined. Second, the strategy of selecting the examples stored in the buffer plays a role. Many different approaches can be considered to achieve this. Among others, possible ideas can include various feature-based selection methods (e.g., similarity-based) or based on predictions (e.g., based on correct or false classification). Besides this, random-based approaches are possible.

In the experiments carried out, both factors, balancing and the selection strategy, are analyzed separately and in combination. Table 5.5 gives an overview of tested strategies grouped by the mentioned factors. The columns *class-balanced (Class-Bal.)* and *task-balanced (Task-Bal.)* refer to the balancing aspect. The markers in columns *Exp. 1 - 3* show which buffer constellations are compared in which of the three sub-experiments. The random strategy is listed separately from the groups in the table. The selection strategies used are the following:

- **random:** The random selection assigns a random value from zero to one to each new example in the training data. To decide which element to add respectively to replace from the buffer, the elements from the buffer concatenated with the new elements are ordered by their assigned random values. Subsequently, the first m elements of the ordered list are kept in the buffer, where m is the maximal buffer size.
- **close-to-center:** The strategy selects the examples based on Euclidean distance to the feature mean of the elements in the buffer (including new examples to be added). It can be understood such that all new elements of a batch are added to the buffer, then the feature mean of all elements in the buffer is computed, and finally, the elements are ordered by their distance to the feature mean from which the m closest elements are kept in the buffer. As features, the fastText document vectors (average over all word vectors of each document) are taken.

- **similarity-based (diversity):** The strategy is adapted from the approach suggested by Aljundi, Lin, et al. (2019). It selects elements based on cosine similarity aiming to maintain a wide variety of examples in the buffer. As in the strategies before, the new training data is added to the buffer, and the pairwise cosine similarity between all elements is computed. To maintain diversity, one of each two most equal elements is removed. This strategy fills the buffer such that the elements cover a wide range of the feature space.
- **false classified:** The strategy selects elements based on their classification by the current model. The elements are ordered from false classified to correctly classified, whereas the ordering within false or correct is random. As a result, the buffer is intended to keep the false classified examples and may be filled with correct classified if false classified examples do not fill the whole buffer. The intention is to improve the model by replaying false classified examples from the previous tasks.

Regarding the balancing, class-balanced means the buffer is split into three parts of equal size (in the present case, one-third of the complete buffer size each), whereas each part is reserved for one class. That means balancing is not necessarily ensured in the beginning phase when the buffer just starts filling. From the point where the buffer is completely filled, class balancing is definitely ensured. However, in the present case, the classes are balanced in the dataset. Therefore this is not an issue. Task-balanced means the buffer is balanced over the tasks, such that the sizes of the buffer subsets are proportional to the number of elements in each task. If the buffer is not balanced, this means the elements are selected randomly, such that imbalance can occur.

For comparison to all buffer constellations, an unbalanced buffer with random selection is used.

To get insights about the impact of different buffer constellations, the analysis was split into three sub-experiments as shown in Table 5.5. The first sub-experiment analyses the influence of balancing while using a random selection strategy. The second sub-experiment analyses different selection strategies using no balancing, and the third sub-experiment tests balancing and different strategies in combination.

5.4 Experimental Results and Discussion

This section shows and discusses the results of the described experiments. The sections are chosen analog to the experiment description sections, with the difference that the results of the buffer analyses are discussed in three separate subsections. Additionally, subsection 5.4.8 uses all parameters leading to the best results when analyzed separately. It combines the separately optimized parameters in one run before the section closes with a summary.

5.4.1 Analysis of Cumulative Training Baselines

The cumulative training was performed to get an impression of the upper bound of the performance that can be achieved with the described feature set and models and to be able to better assess the performance of the CL-approaches later.

Name/Constellation	**Selection Strategy**	**Class-Bal.**	**Task-Bal.**	**Exp. 1**	**Exp. 2**	**Exp. 3**
random	random	✗	✗	●	●	●
Balancing						
task-bal-rand	random	✗	✓	●		
class-bal-rand	random	✓	✗	●		
class-task-bal-rand	random	✓	✓	●		
Selection Strategy						
classification-based	false classified	✗	✗		●	
sim-based	similarity-based (diversity)	✗	✗		●	●
close2ctr-based	close-to-center	✗	✗		●	●
Balancing & Selection						
task-bal-close2ctr	close-to-center	✗	✓			●
task-bal-sim	similarity-based (diversity)	✗	✓			●
class-bal-close2center	close-to-center	✓	✗			●
class-bal-sim	similarity-based (diversity)	✓	✗			●

Table 5.5: Overview of analyzed buffer constellations.

The table overviews the buffer strategies used in the respective experiments. Besides the random approach, all others are grouped in upper classes regarding balancing of the buffer, selection of examples to store in the buffer and both combined. The markers in columns Exp. 1-3 show in which experiment(s) each constellation is used.

Regarding the average performance (Figure 5.5(a)), the diagram shows the F1-Score measured after each task on the test data of the current and all previous tasks. Additionally, shaded areas show the standard deviation of the F1-Scores over five repeated runs. After training on task one, the performance is very high (approx. 0.6 to 0.75), with a big standard deviation for the MLP and LSTM. The performance massively drops at task two to values between 0.3 and 0.4. The high performance with the big standard deviation at the beginning is caused by the fact that the test data of task one contains only nine examples. This is too less to get reliable results. After task two, the performance of all models steadily increases up to task 5, where the F1-Score flattens around 0.6. Until task ten, it rises slightly above 0.6 before it goes up in the last task again. The LSTM reaches with an F1-Score of 0.6645 the highest performance, followed by the SVM (0.6549) and MLP (0.6486, Table A.1). However, the LSTM has mostly a slightly higher standard deviation than the MLP, which means the differences in the performance between the five repeated runs are larger. The SVM has a standard deviation of zero. This model is in contrast to neural networks deterministic and does not depend on the initialization.

Similar results between all three models also show the class-wise analysis after each trained task (Figure 5.5(b)). The class *Conduct of Life* reaches already after task three an F1-Score of 0.5 to 0.6, whereas the *Adventure Stories* and *Love Stories* reach this performance only after task five. However, in the further process, the performance of all three classes slightly rises except for the SVM for the class *Love Stories*. The latter starts with higher performance and slightly decreases in some sections of the training process. Besides minor deviations, models achieve comparable results after the last task with no significant differences between the classes. In conclusion, the differences between the three models tested are marginal at this stage.

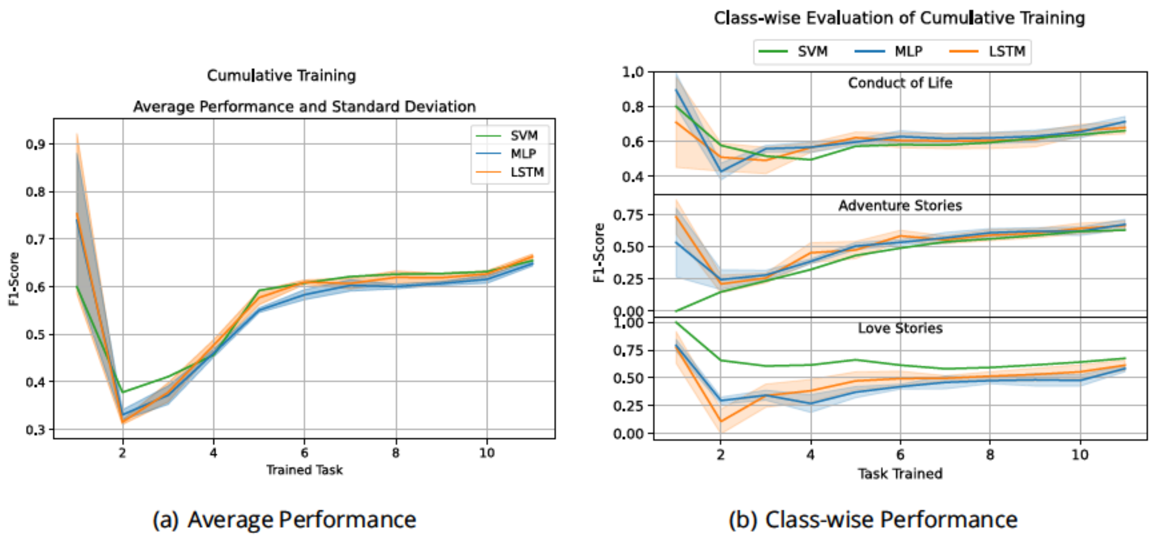


Figure 5.5: Performance of cumulative trained models.
An SVM, MLP, and LSTM were cumulatively trained, and after each learned task, they were evaluated on all previous and the current task. The colored areas depict the standard deviation from the mean over five repeated runs. Sub-figure (a) shows the macro-averaged F1-Score over the classes after each task. In (b), the performance is shown separately for each class.

In the task-wise analysis of the cumulative training (Figure 5.6), the progress of the performance is analyzed separately for each task. For example, the diagram of task three shows the performance of task three over the whole time from task one to eleven. After task one, the F1-Score of the SVM, MLP, and LSTM was around 0.3, even though the models were only trained on data from task one. Subsequently, the models were further trained on task two data, on task three data (marked by the dashed vertical line), and so on. At task three, the models were first trained on the actual task it was evaluated on in this example. Accordingly, low performance on a task before the model has actually been trained on that task (before the vertical dashed lines in the diagrams) should not be viewed negatively.

From the task-wise analysis (Figure 5.6), it can be seen that the models perform equally well. In tasks one and nine, the reason for differences in the performance of the models is with high probability due to the small number of examples in both tasks, which results in higher fluctuation. In general, the performance in all tasks, except task one, increases over time until task five to seven before most tasks' performance stagnates. After the last task, the performance rises visibly again in tasks nine to eleven. This observation can be traced back to the fact that tasks nine to eleven contain blurbs instead of excerpts of books which come with a higher amount of new tokens again (compare Figure 5.4). Because tasks nine and ten both have a relatively small amount of training data, the performance visibly increases just after task eleven with a much higher amount of training data. The reached F1-Scores at the end increase with the task number. Tasks one to three stay below 0.6 for the F1-Score, tasks four to eight reach F1-Scores in the range of 0.6 to 0.8, and tasks nine to eleven have F1-Scores of over 0.8 at the end.

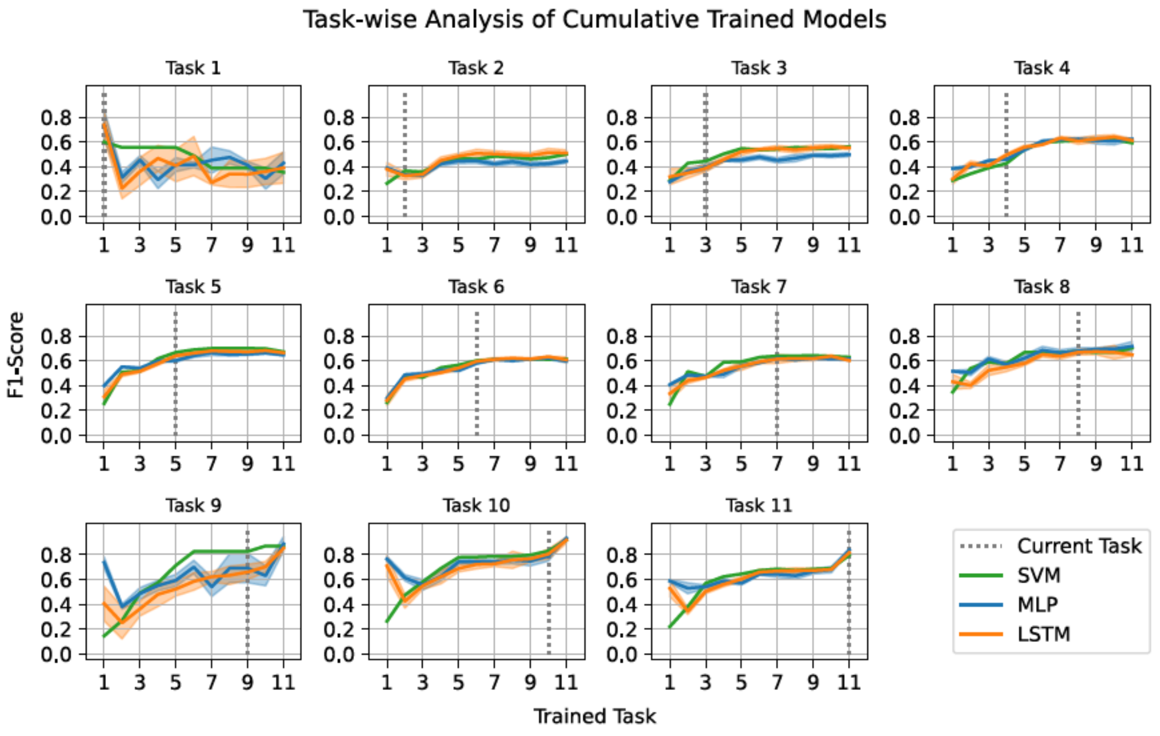
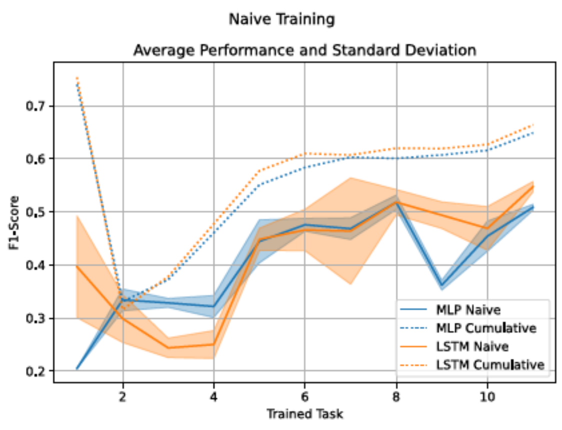


Figure 5.6: Task-wise performance (F1-Score) of cumulative trained static baseline models. The x-axes in the diagrams mark the tasks and on the y-axes the F1-Score is plotted. The semi-transparent areas indicate the standard deviation from the mean over five repeated runs. Each of the sub-diagrams shows the performance of the respective task. The vertical dashed line indicates the task where the model first saw the training data of the respective task.

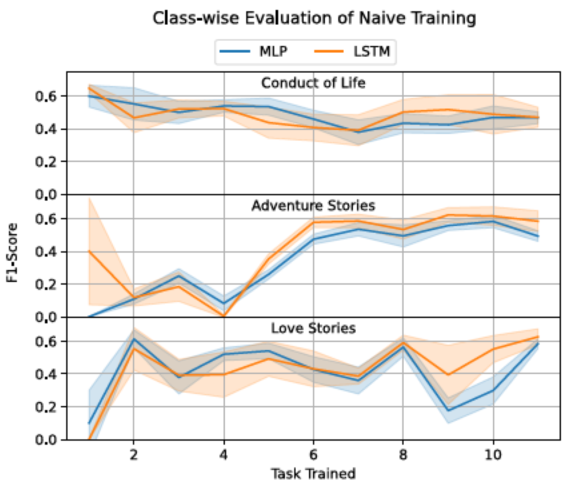
5.4.2 Analysis of Naive Training

The naive training experiment represents the CL baseline for all further analyses. Based on the average performance (Figure 5.7(a)), it is clearly visible that the cumulative trained models (dotted lines) outperform both naive trained models as expected. In numbers, the naive trained MLP has approximately a 0.14 lower F1-Score, and the naive trained LSTM performs 0.11 worse than the respective cumulative trained model. Furthermore, the curves of the naive trained models fluctuate, whereas the graph lines of the cumulative trained models are more stable. Under consideration of the class-wise performance (Figure 5.7(b)) the fluctuation can mainly be traced back to the class *Love Stories*, as the respective performance curves of both models show more fluctuation for the category *Love Stories* than the curves of the other classes.

In the task-wise analysis Figure 5.8, it can be seen that all tasks perform worse than in the cumulative training. This leads to the conclusion that the general loss in performance (Figure 5.7(a)) is spread over all tasks, and there is not a single task performing much worse. Comparing both models in this area, the performance of the MLP drops sharply, whereas the performance of the LSTM decreases more slowly. This observation coincides with the drop in the performance at tasks nine and ten visible in Figure 5.7. This drop can also be seen in the analysis of the average forgetting and BWT Figure 5.9). After a low, even slightly negative forgetting, which indicates an increase in the performance of previous tasks triggered by task eight, the forgetting rises again in task nine to approximately 0.12 for the MLP and to



(a) Average Performance



(b) Class-wise Performance

Figure 5.7: Performance of naive trained baseline models.
Sub-figure (a) shows the average performance (F1-Score) over all classes after each trained task. Detailed values of the average performance are shown in Table A.2.

0.04 for the LSTM. The values describe the average loss of the performance of the respective model for each task compared to the maximum performance achieved in each task during the previous training steps.

Generally, the average forgetting of the LSTM is with fluctuation between -0.05 and 0.05 largely lower than the forgetting of the MLP. At the beginning (until task five), the BWT of the MLP is higher compared to the LSTM. Looking at the task-wise analysis, it can be seen that tasks one to four have all improved from the first training until task five. For instance, task one improved from 0.2 at the beginning to almost 0.6 at task five. Task two improved from approximately 0.3 (when the model was first trained on task two) to 0.4 after task five, and so on. The LSTM has a drop in the performance of almost all tasks at task four, where it has a peak in the forgetting graph, resulting in a lower BWT at the beginning.

In a nutshell, the naive training showed that the MLP had a better performance at the beginning until task five before the LSTM outperformed the MLP after task eight. Both models have a partly fluctuating performance graph. However, the LSTM has a bigger standard deviation, which means the results between repeated runs with the same parameters deviate more than it is the case with the MLP.

5.4.3 Analysis of the Impact of Number Replayed Examples

The diagrams of the average performance after each trained task (Figure 5.10) show the performance graph for each number of replayed examples. The left diagram shows the results of the MLP compared to the on the right showing the results of the LSTM. The best performance is in large parts achieved by 15 replayed examples (red line). The performance after task one starts from 0.2 equally in all runs before the performance strongly raises until task six with a light drop at task four for the runs with fewer replayed examples. Until task six, the performance of all tasks except the naive training (blue line) is still comparative (F1-Scores between 0.5209 and 0.5532). After task six, the performance drops in all tasks, but the drop

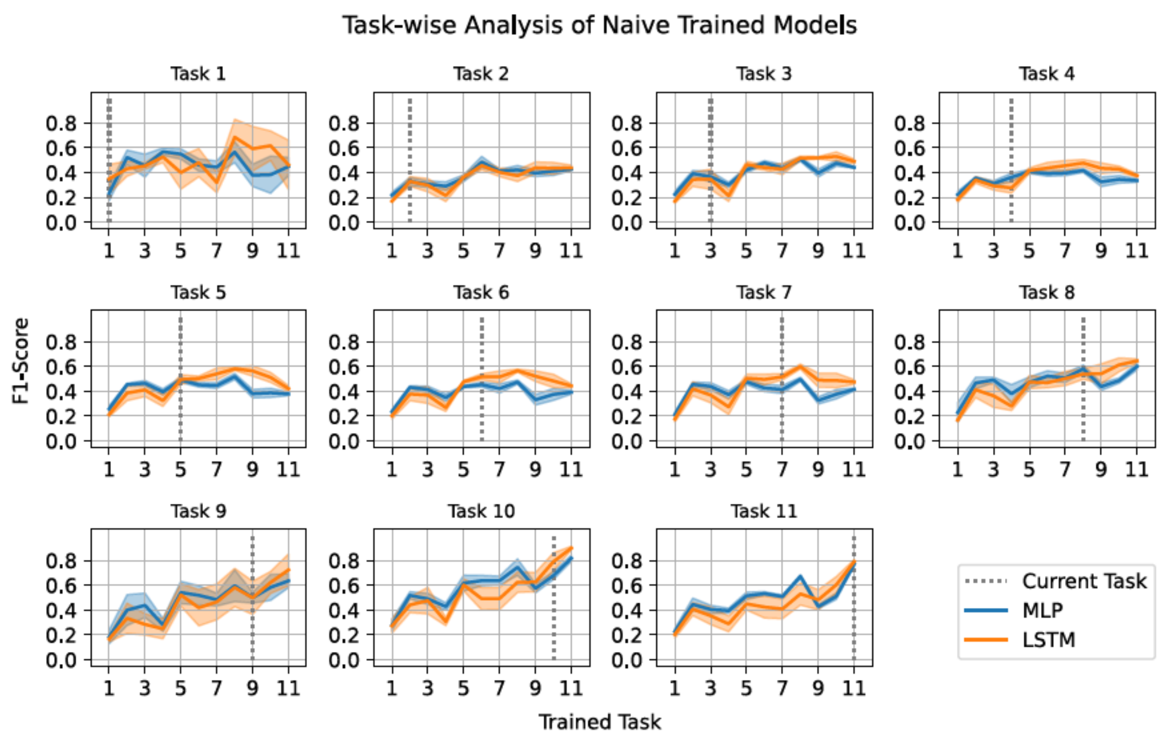


Figure 5.8: Task-wise performance (F1-Score) of naive trained baseline models.
The MLP and LSTM models were evaluated using F1-Score separately on each task. The vertical dotted line in each diagram marks the task where the models were first trained on the respective task. The semi-transparent areas show the standard deviation over five repeated runs.

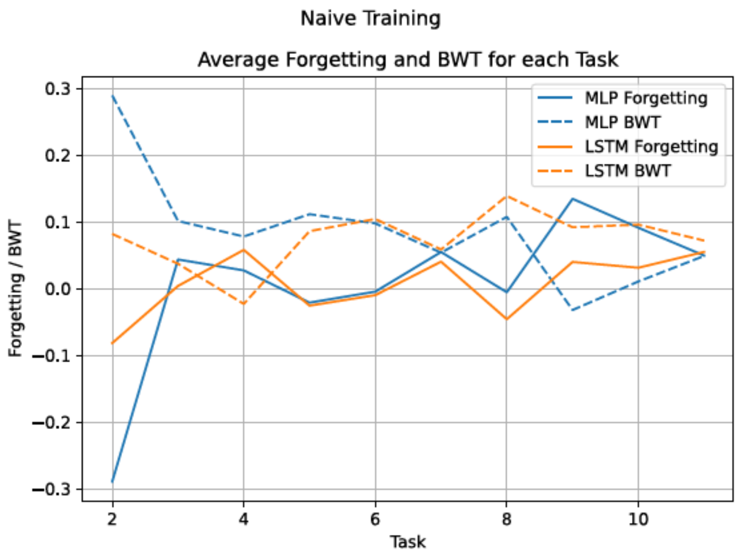


Figure 5.9: Average forgetting and BWT for naive trained baseline models.
Analysis of forgetting and BWT for the MLP and LSTM after each trained task. In general, a low forgetting and a high BWT are desirable. Both measures influence each other. Strong forgetting results in lower BWT and vice versa.

is more significant the smaller the number of replayed examples is. The maximum drop in all cases is reached at task nine, which was also seen in the naive training before. The drop of the run with 15 replayed examples is with a difference of 0.0279 much smaller than the drop of the run with one replayed example (0.1137). This shows that replay had a positive effect on overcoming the strong forgetting at task nine when the model first saw text from blurbs instead of texts from books before. After the last task, the results vary between 0.4902 (one replayed example) and 0.5808 (15 replayed examples).

The results of the LSTM differ from this in several ways. Whereas the difference in the performance of all MLP runs was low until task six, this is not the case for LSTM. The differences between the runs show bigger differences from the beginning on. Furthermore, the performance rises much slower during the first tasks. Whereas all MLP runs except zero replayed examples reach an F1-Score of more than 0.5 after task six, no run achieves this with the LSTM. The results after task eleven lie between 0.5238 (one replayed example) and 0.5669 (15 replayed examples), achieving results comparable to the MLP at the end. Surprising with the LSTM is the observation that the performance curve of the naive training (blue curve) is over large parts almost as good as the performance of the runs with ten and 15 replayed examples.

Notable differences between the MLP and LSTM also shows the analysis of the ranges between the best and worst performing class at each task (Figure 5.11). In the MLP, the range is over large parts lower and also the differences between the runs with varied numbers of replayed elements are smaller than in the LSTM. Only at task nine, where a drop in the performance graph occurred, all runs except for 15 replayed examples show a peak in the range. Interestingly, this peak is not as large in the LSTM at task nine. However, most LSTM runs show a significant peak at task seven.

The analysis of the forgetting and BWT (Figure 5.12) coincides with the previous results. The diagrams show the forgetting as solid lines and BWT as dotted lines. The color encodes the number of replayed examples. To maintain clarity, not all runs are shown in the diagrams. The forgetting in the MLP was negative, and the BWT high for all runs with replay at the beginning and closer to zero and more fluctuation from task seven on. In the LSTM, the opposite is the case. At the beginning until task seven, forgetting and BWT enormously fluctuate, but from task eight on, the forgetting curves of all runs with replay flatten around zero while having a BWT of more than 0.05 for all curves. The observations mirror the sharp rise in the performance at the beginning of the MLP and the slow increase of the LSTM. Contradictory looks the forgetting curve of the naive training (blue solid line) of the LSTM compared to the other forgetting curves, as it decreases from task one to three in the naive training but increases in all other cases. This is caused by the fact that the first task's test data contains only nine examples resulting in huge performance fluctuations when measuring the performance separately for each task, which is the case in the calculation of forgetting and BWT. However, when evaluating all test data up to the current task, the results on task one have just a small impact due to task one's low number of test examples. To cut it short, abrupt rises in the performance of task zero to task one, followed by a sharp drop to task three, which is valid for all classes, leads to the resulting forgetting curve. However, this fluctuation is not visible in the performance graph, as task zero has very few test examples and, therefore low impact on the overall performance.

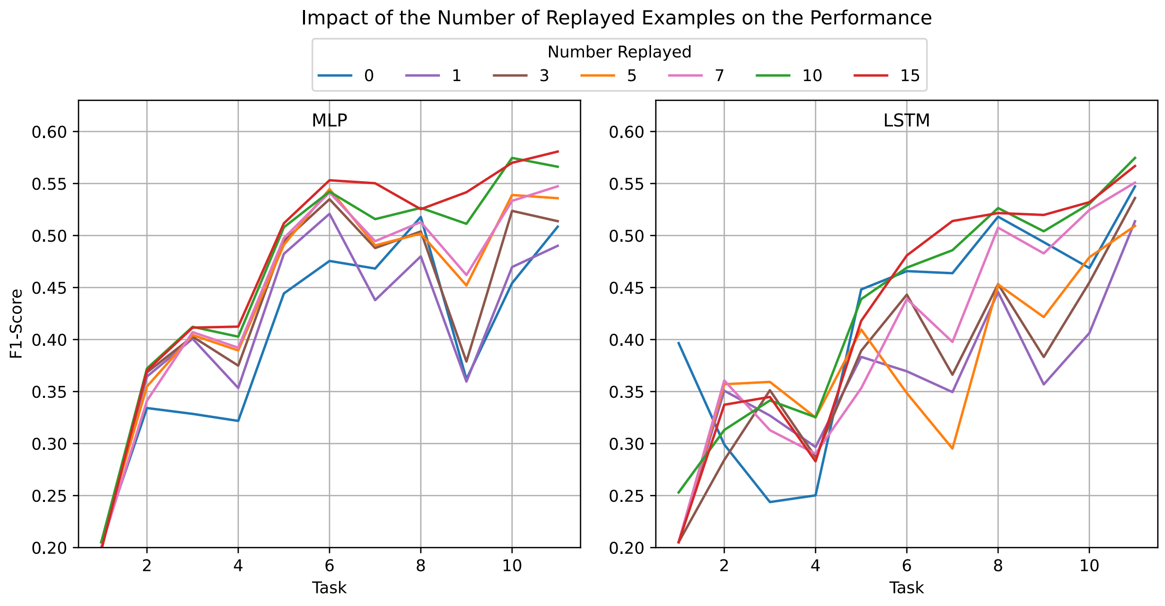


Figure 5.10: Impact of the number of replayed examples per ten new examples on the average performance.
The diagrams show the performance (F1-Score) of the MLP (left) and the LSTM (right) after each trained task for different numbers of replayed examples. Zero replayed examples is equivalent to naive training and included for the purpose of better comparison. Detailed values are shown in Tables A.3 and A.4

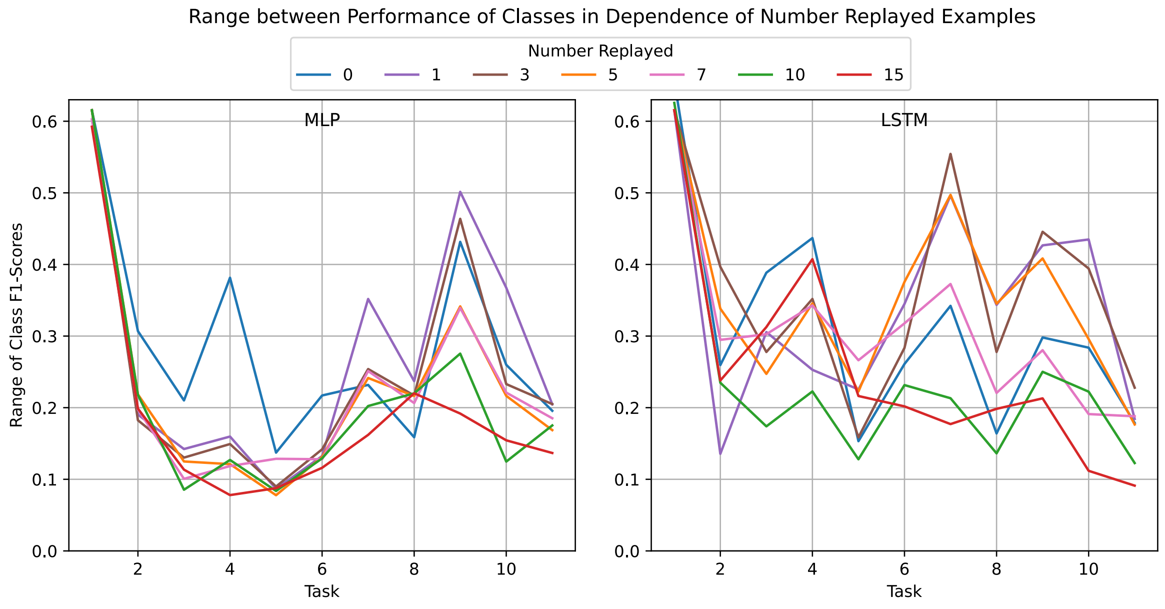


Figure 5.11: Impact of the number of replayed examples per ten new examples on the range of the performance of the classes.
The graphs represent the difference between the best and worst performing classes at each task for varied numbers of replayed elements. The range should ideally be low, as this indicates equal performances of all three classes.

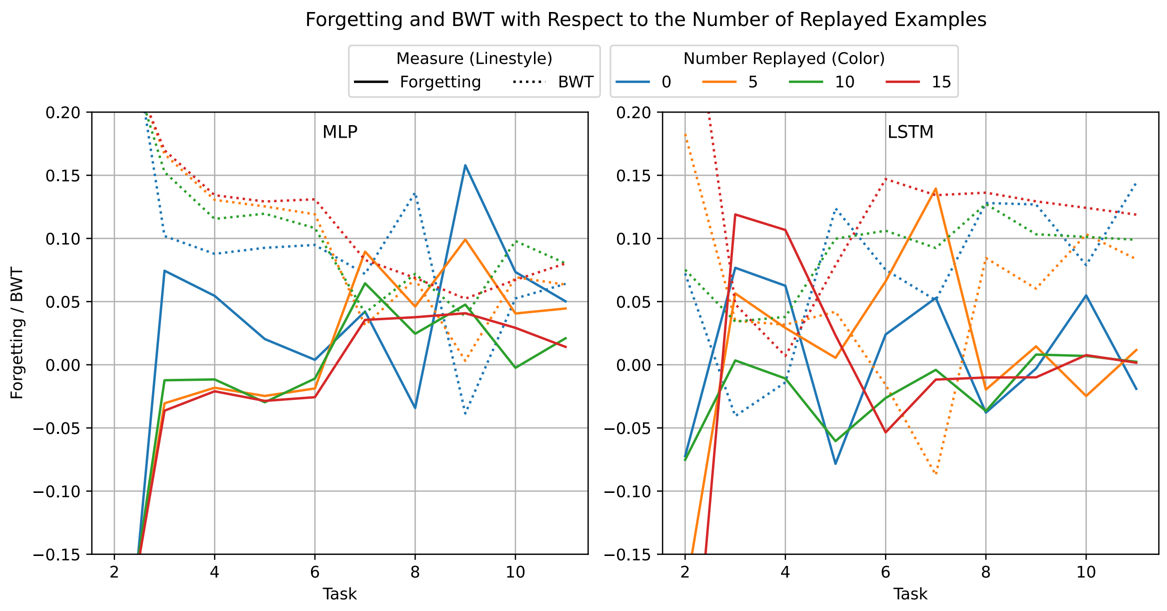


Figure 5.12: Analysis of forgetting and BWT depending on the number of replayed examples. The line style encodes the measure, which is either forgetting (solid lines) or BWT (dotted lines). By the colors results of runs using different numbers of replayed examples are compared. In order to maintain clarity, not all runs are plotted, but only these using 0, 5, 10, and 15 replayed examples.

The results of the MLP clearly confirm the hypothesis that the performance increases and the range between the class-wise performances decrease with an increasing number of replayed examples. This only applies to a limited extent to the LSTM. The naive training does not fit in the other results, as it achieves unexpectedly good performance with no replay coming close to the runs with ten and 15 replayed examples and outperforming the runs with less than ten replayed examples. This leads to the conclusion that replaying a small number of examples is inhibitory for the performance. A reason for the general slower increase in the performance of the LSTM could be the model size. With 68,291 trainable parameters, the LSTM is much more complex than the MLP with 15,303 trainable parameters. Due to the larger size of the LSTM, it is likely that it needs a larger amount of training data to fit its parameters properly and to stabilize. The diagram of the forgetting speaks for this besides the diagram of the performance. The fluctuation of the forgetting and BWT curves indicate high plasticity and low stability in the first tasks before the model seems to become more stable after task 7.

Based on this, it can be hypothesized that if the plasticity is too high, replay will have no major effect on model improvement. At the beginning of the training over the first tasks, the model saw only a relatively low number of different examples. If a selection from the seen examples is further replayed to the model, it can be imagined that the model overfits to the replayed examples due to high plasticity. This also explains the drop in the performance curves of the LSTM at task four, as the number of examples replayed plays no role here and does not prevent sudden drops in performance. The highest performance at this point achieve the runs with five and seven replayed examples, but upon others, the run with 15 replayed examples has one of the lowest performances and a deep performance drop at task four. Only later, from tasks five to six, the replay of more examples shows advantages regarding the performance itself and the prevention of sudden performance drops. That replay has a low effect on the performance during the first tasks coincides with the curves of the MLP. Until task six, the performance of all runs is still in the same range within 0.05 except for the naive training, which has a lower performance. However, in contrast to the LSTM, a higher number of replayed examples dampens sudden performance drops, as seen in task four. The reason why a higher number of replayed examples has a positive influence in the MLP but not in the LSTM in the early training phase at task four could therefore be explained by a higher plasticity of the LSTM due to a higher number of model parameters that must be adjusted.

What cannot be explained by the higher complexity of the LSTM and the associated larger amount of required training data is the high performance of the naive training. To find the reason for this, further analyses must be done. For example, a role may also play the optimizer for which Adam-optimizer was currently used or the applied loss function. Furthermore, an LSTM with a smaller hidden layer and accordingly a lower number of trainable parameters should lead to better results if the high complexity of the current LSTM would be the reason.

To summarize the effect of the number of replayed examples per ten new training examples, it can be said that a larger amount of replay generally leads to better results. However, the power of replay does not have a substantial effect starting right from the beginning, but it takes some time for replay to make a difference. In the present case, an advantage appeared from approximately the sixth task. Most of the benefits of replay were seen in the improvement of the average performance and in the ability to dampen sudden drops in the performance. When comparing MLP and LSTM, replay had a better effect on the results of the MLP, but both models improved compared to the naive training.

5.4.4 Analysis of the Impact of Number Training Iterations

Looking at the average performance (Figure 5.13), it can be seen that the number of training iterations leads to only minor differences in the case of the MLP. The highest performance of 0.6090 after task eleven is achieved with five iterations. Despite the run with one iteration, all other runs achieve only slightly lower F1-Scores. The differences in earlier tasks also show equal results. Besides the run with five iterations with the best performance over large parts, the difference to the other runs with at least three iterations is minimal.

The results of the LSTM show more differences between the runs. From task seven on again, the run with five iterations achieves the best F1-Scores resulting in 0.6045 after task eleven. Only slightly worse perform the runs with seven and nine iterations, followed by the run with 15 iterations. The run with three iterations performs about 0.1 worse than the best run between tasks five and nine. In the last tasks, the performance improved a bit. The run with only one iteration has the lowest performance, with large parts having an F1-Score lower than 0.4.

In the diagrams of both models, it is visible at task one that the curves of the runs with one and three iterations start with an F1-Score lower than 0.25, and all other runs achieve a much better performance of over 0.35 for the MLP and over 0.4 for the LSTM. On the one hand side, this shows that the networks have more training steps to fit the weights due to more training iterations. On the other hand side, the high performance seems to come from an overfitting

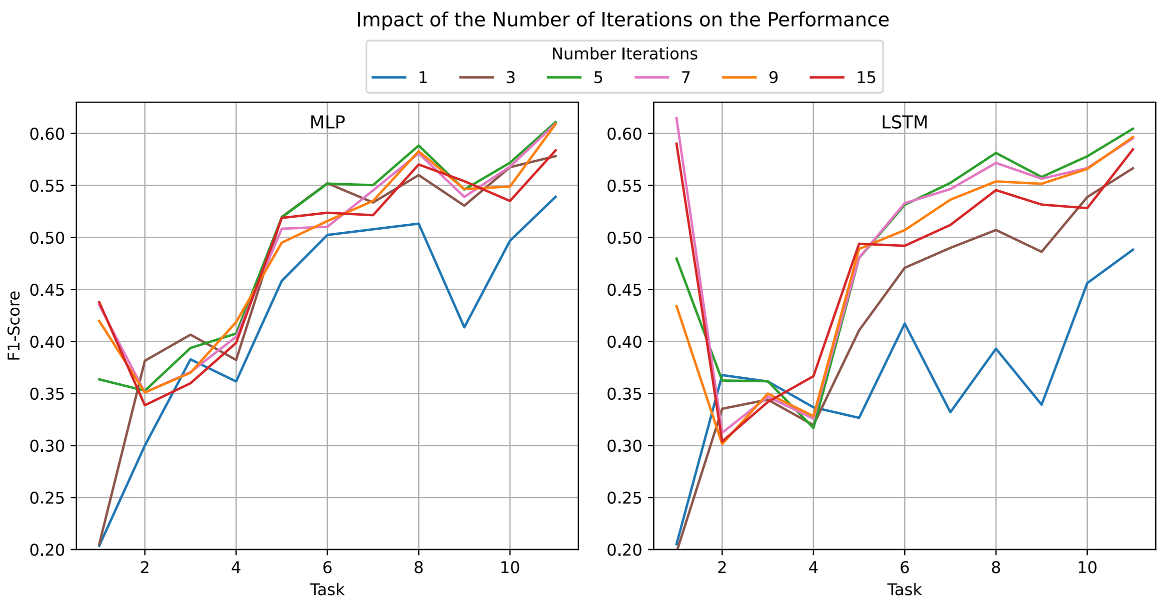


Figure 5.13: Impact of the number of training iterations on the average performance.
The number of training iterations per batch was varied between one and 15. The diagrams show the respective performance graphs for the MLP and LSTM. Detailed values are listed in Tables A.5 and A.6.

to the majority class, as the range between the class with highest and lowest performance (Figure 5.14) is also big at task one. Another indicator of overfitting is the performance drop after task one in both models.

Regarding the ranges for the MLP (Figure 5.14), contrary to the performance, bigger differences between the runs are visible around tasks three to eight. The smallest ranges in this part have the runs with three and five training iterations lying between 0.1 and 0.2 after task three. This strengthens the observation that five iterations lead to the best results for the MLP. In contrast, at the LSTM, the range between the best and worse performing class decreases over time. It starts with bigger ranges at tasks three and four compared to the MLP, but it improves to lower ranges than the MLP has from task 8 to the end when comparing the curves with the lowest ranges in both models. In the LSTM, the lowest ranges are achieved with 15 iterations at the beginning before the runs with five and seven iterations further improve and achieve the lowest ranges. Also the F1-Score performance of the run with 15 iterations was higher compared to the other runs, showing that the system might benefit from a higher number of iterations at the beginning, which could be reduced after a certain number of tasks or training experiences. Generally, analysis of the ranges confirms in the case of the LSTM that one is too less for the complex model, and also three iterations are too less to be on a level with the runs using a higher number of training iterations.

In accordance with the best performance of the MLP when using five training iterations are also the results of the analysis of forgetting ans BWT (Figure 5.15). Compared to the runs with a higher number of iterations (orange and red graphs), it is visible that the green graph has the highest BWT while preserving forgetting over large parts being lower or equal to the other curves. This shows that the model adapts more to the current training data with more iterations. However, it loses generalization simultaneously, which is visible in a lower BWT of the orange and red graphs compared to the green graph. An unexpected course has the blue curve using one iteration in the diagram of the MLP, as it shows a high BWT and

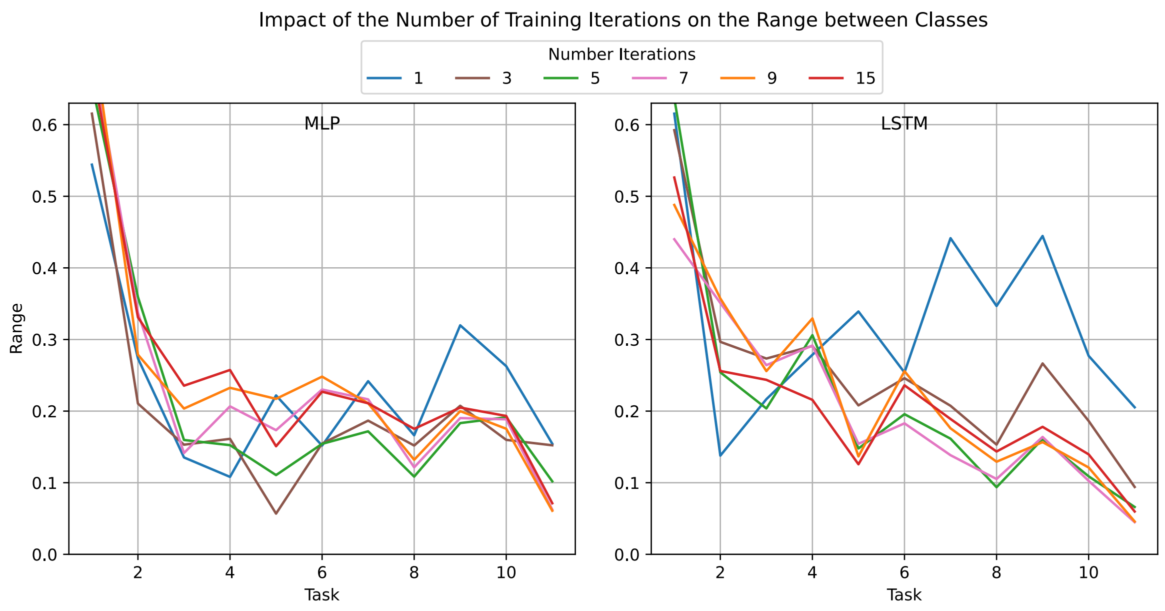


Figure 5.14: Impact of the number of training iterations on the range of the performance of the classes.
Analysis of the difference (range) in the performance (F1-Score) of the best and worst performing class at each task for the MLP and LSTM.

a low forgetting until task eight. This is caused by the fact that the performance of the run using one iteration is very low (almost 0.20) at the beginning but heavily increasing by about 0.3 until task six. In contrast, the runs with five or more iterations start with significantly better performances between 0.35 and 0.45 at task one and increase to 0.50 to 0.55 at task six, which is an improvement of maximal 0.2. The higher improvement of the run with one iteration results in a higher BWT and lower forgetting at the beginning. However, in the last tasks, it can be seen that the blue graph in Figure 5.15 shows expected results (high forgetting and low BWT).

For the LSTM, the analysis of forgetting and BWT (Figure 5.15) is more straightforward. As expected, the curve with one iteration has a high forgetting and a low BWT. The best performing run with five iterations has over large parts the lowest forgetting and a high BWT. As with the MLP, the results show that too many iterations (more than five to seven in this case) have a negative effect on the generalization, which in turn affects the performance.

In summary, about the number of training iterations it was found for both tested models, the MLP and LSTM, that it influences the general performance as well as the fluctuation of the performance curve.

Regarding the general performance, it was found, that five iterations led to the best results. For the LSTM, a number of five to seven iterations showed the best results. If the number of iterations is too low, it is observed that the performance is clearly lower than the best runs achieved, which indicates that the fitting process of the weights could not be finished. A higher number of iterations than the optimal values result in a slightly lower performance compared to the best runs. This is likely caused by decreasing generalization due to better adaption to current training data.

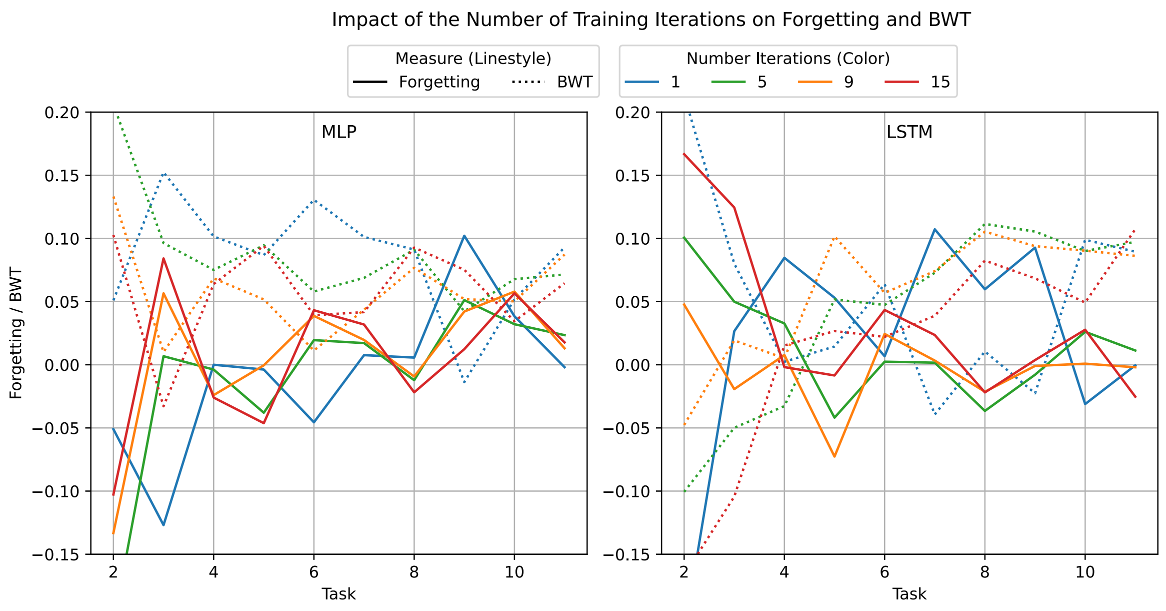


Figure 5.15: Analysis of forgetting and BWT depending on the number of training iterations. Forgetting and BWT computed from the F1-Score results of the MLP and LSTM models. The solid lines depict the forgetting and the dotted lines represent the BWT.

Furthermore, the results showed that a higher number of iterations reduce fluctuation and sudden performance drops in the performance curve up to a certain degree, which means it contributes to stabilizing the performance curve. If the number of iterations is too low, the fluctuation is higher. However, a higher number of iterations than in the case with the highest performance is unlikely to improve the results significantly further.

In the case of the LSTM, it was also visible that 15 iterations resulted in a smaller range between the class performances around tasks four and five (Figure 5.14), which also positively affects the overall performance. However, it is unclear whether this observation can be generalized, as nine iterations showed no improvements to the run with five iterations.

5.4.5 Analysis of Buffer Balancing

The performance curves of runs with different balancing combinations (Figure 5.16) show that for both models, the MLP and LSTM, only marginal differences, whereas the differences in the LSTM are slightly higher, which may be because of the higher number of parameters. However, regarding the performance, neither the MLP results nor the LSTM results show that one balancing strategy works visibly better than another.

When looking at the ranges between the best and worst performing classes (Figure 5.17), the diagram of the LSTM shows that the ranges in the random selection with no explicit balancing are higher than the ranges of the other tested balancing strategies from tasks six to nine. As the average performance is equally high as in the other runs, this indicates that there is at least one worse but also one very good performing class. In the area from task six to nine, the task-balanced run (blue curve) shows the lowest ranges for the LSTM, followed by class- and task-balanced (orange curve). This shows that class balancing has a less positive impact on the ranges than task balancing. This could be caused by the data distribution. As

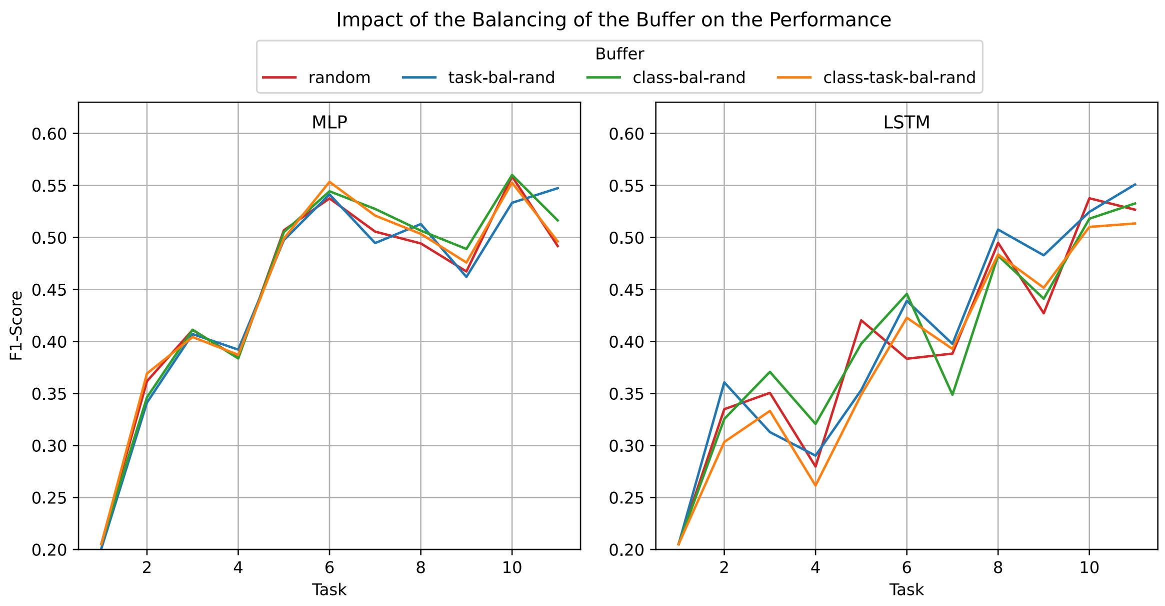


Figure 5.16: Impact of balancing the buffer on the average performance.
The curves show the average F1-Score over five repeated runs for each buffer constellation for the MLP and the LSTM after each trained task.

the classes are balanced in the dataset (similar number of examples for each class in each task), the random buffer is, by the law of large numbers, theoretically also balanced if the buffer size (respectively the number of selected elements) is big enough. This seems to be true for the classes which are balanced over all tasks, as the run with class balancing showed no significant improvement. However, as the number of texts in the tasks differs very much, the buffer is probably too small to make the law of large numbers apply. This means the proportion of elements in the buffer from the different tasks cannot be achieved by random sampling with this buffer size. The observation also coincides with the number of examples in tasks six to nine (Figure 5.2), where the difference was mainly visible. In tasks five, six, and seven, the random buffer is likely flooded with books from those tasks as they have with around 1,000 texts per task and class much more training examples than the tasks before. The following tasks eight to ten have with less than 200 examples per task and class again much fewer examples. This might flood the buffer with examples from tasks five to seven, while the law of large numbers does not apply due to a small task size in tasks eight to ten. The task-balanced buffer, in contrast, manages to keep the proportion of examples per task correct.

For the MLP, the diagram of the ranges does not show significant differences between the tested buffer constellations. This indicates that the MLP is less influenced by replay than the LSTM, which is, in turn, caused by the much lower number of trainable parameters in the MLP. The lighter model better fits to the data already in the training of each task, and therefore replay has not the function to improve the model but to prevent forgetting.

5.4.6 Analysis of Selection Strategy

This experiment (buffer constellation sub-experiment 2) was designed to get insights into the selection strategy to select examples stored in the buffer. It is analyzed without any balancing at this point. The MLP and LSTM performance curves (Figure 5.18) clearly show

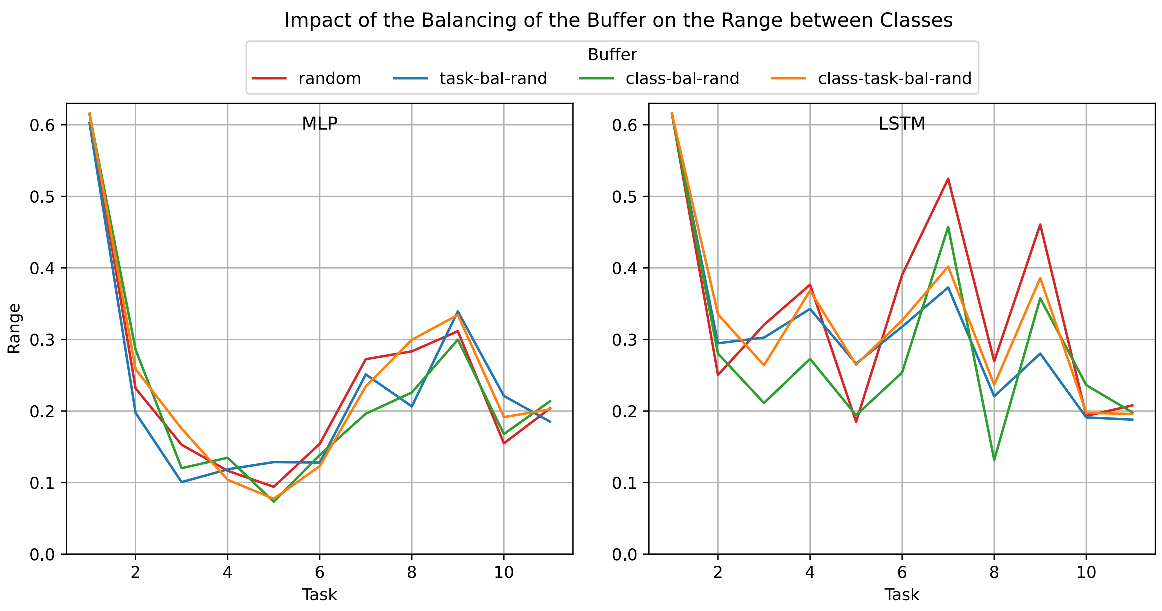


Figure 5.17: Impact of balancing the buffer on the range of the performance (F1-Score) of the classes. The diagrams show the range between the best and worst performing classes after training on each task using the F1-Score as the performance measure. The results are averaged over five repeated runs each.

that the classification-based approach achieves the lowest performance. The difference to the other approaches becomes, in particular, visible from around task seven. Interestingly, in the MLP, the performance sharply drops from over 0.5 to almost 0.25 from tasks six to seven. Looking at the range between classes (Figure 5.19) at this point, it can be seen that the range is with almost 0.5 very high. This indicates that at least one class performs poorly, whereas the model adapted well to another class. Once the model comes in a strong imbalance regarding the performance of the classes, the problem worsens. The reason is that, followed by the performance imbalance, many elements of the worse performing class are added to the buffer, which leads to a shift in the prediction imbalance. The worse performing class from before will likely improve due to many elements of this class added to the buffer, whereas the foremost good performing class loses performance. This results in an unstable model.

Another explanation for the observation is that the tasks immediately before task seven contain many falsely classified outliers. Due to the strategy, the false classified outliers are added to the buffer and replayed, leading to an adaption of the model to fit the outliers. In turn, the representative examples of the data, which were correctly classified before, were then falsely classified. Subsequently, in task seven, the representative examples of the data are falsely classified, which makes them adding to the buffer resulting in an increased performance due to its replay in task eight again. The selection of false classified examples thus leads to fluctuation in the performance curve.

In the LSTM, this observation is not visible as clearly, which is caused again by the fact that the LSTM has a much higher number of trainable parameters, and subsequently, it adapts slower to new data. This also means that it adapts slower to probably replayed outliers which negatively influences the MLP. Nevertheless, the classification-based selection also achieves the lowest performance with the LSTM.

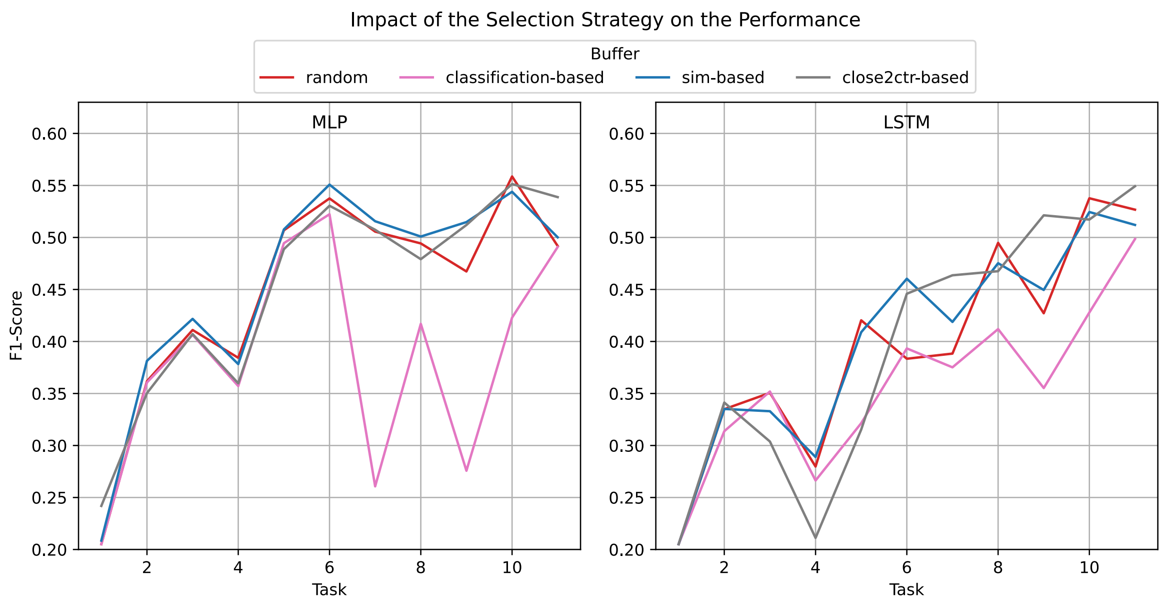


Figure 5.18: Impact of the example selection strategy on the average performance. Comparison of strategies of how to select examples to store in the buffer. Classification-based selects in the first line false classified examples, similarity-based (sim-based) aims to maintain a wide diversity of samples in the buffer by dropping similar elements, and the close-to-center (close2ctr) strategy selects elements based on their distance to the feature mean of the elements in the buffer.

The other tested strategies (random, similarity-based, and close-to-center) achieved comparable results with minor differences in both models. The close-to-center selection strategy shows a slight advantage in case of the LSTM from tasks six to eleven, as the curve is more stable than the others. However, in the beginning, the close-to-center selection is not as stable, which becomes visible by a sharp performance drop at task four. The reason for this could be that the buffer contains elements that are not that representative for the later elements. As new elements are selected based on their Euclidean distance to the feature mean of all elements in the buffer, this leads to a gap between the elements in the buffer and the current training data. After task four, the buffer content adapts to the current data, which stabilizes the model.

Looking further at the ranges between classes (Figure 5.19), the curves mainly confirm the results that the differences between the random, similarity-based, and close-to-center strategy are only marginal. A slight advantage shows the close-to-center strategy in the last tasks of the LSTM training, as it is the only strategy where the range constantly reduces and has less fluctuation regarding the ranges than the other strategies.

The strategy analysis comes to the result that it is essential that the buffer contains a high percentage of representative examples of the respective tasks and classes. This is ensured in the first line in the close-to-center strategy. But also, the random approach and the similarity-based approach (aiming at diversity) still work well. In the case of the random approach, it lies in the nature of the random selection, that it selects examples proportional to the distribution of the underlying data. In the similarity-based strategy, among others, also the main representative examples are selected, as the strategy selects examples across the whole feature space. However, the diversity does not result in a better performance such it can be said that the examples near the feature mean have the most significant impact.

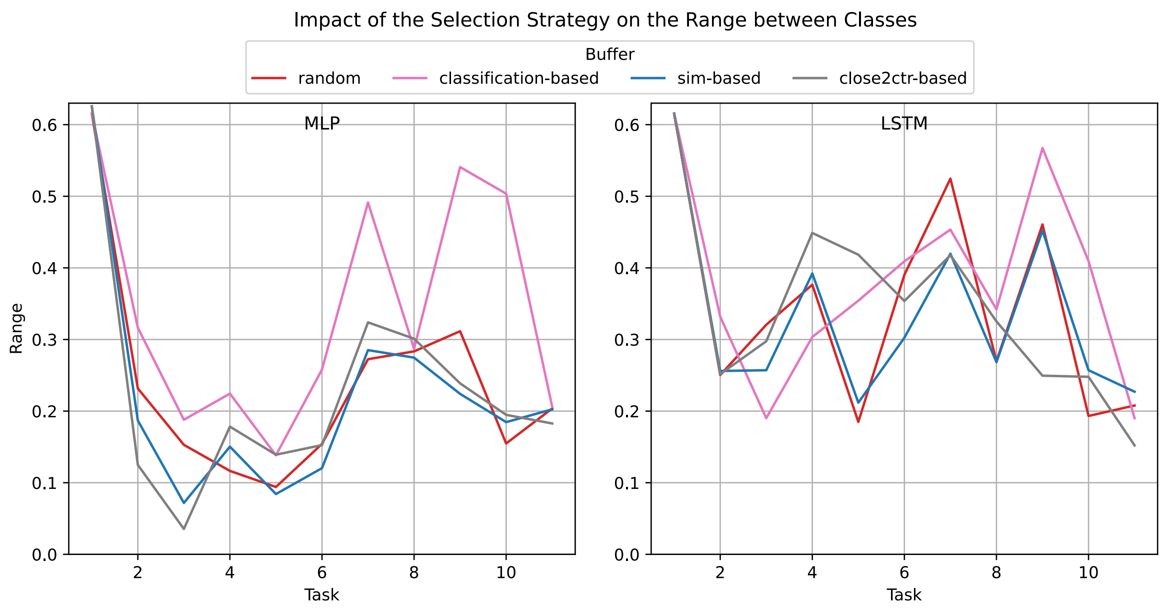


Figure 5.19: Impact of the balancing in combination with selection strategy on the range of the performance of the classes.
Comparison of strategies of how to select examples to store in the replay buffer regarding the range between the best and worst performing class for the MLP and LSTM.

5.4.7 Analysis of the Combination of Balancing and Selection Strategies

In the analyses before, selection strategies and buffer balancing were separately analyzed. It was seen that classification-based selection works poorly, and task-balancing leads to a slight improvement compared to the other balancing strategies. However, the buffer constellation could make a difference if balancing and specific strategies are combined. For example, it is conceivable that a class-balanced approach combined with the close-to-center selection strategy results in an improvement of the performance, whereas both constellations separately do not show improved results.

However, the results of the seven compared constellations (Figure 5.20) show that also the combination of balancing and selection strategies leads just to minor differences. For the MLP, the most significant differences are visible around tasks eight to eleven. But as all curves fluctuate in this area, it cannot be clearly said which constellation has the best results. The most negligible fluctuation has the task-balanced close-to-center (task-bal-close2ctr) curve. Looking at the ranges-diagram for the MLP (Figure 5.21), the task-balanced close-to-center constellation has, in particular during tasks seven and ten with just slightly more than 0.2 a constant range whereas all other constellations have at least one peak up almost 0.3 or higher. This could neither achieve the task-balancing nor the close-to-center strategy separately, which shows that the size of the ranges slightly benefits from this combination.

For the LSTM, the performance diagram shows that the curves of the task-balanced close-to-center and the task-balanced similarity-based constellations show the most stable curves after a drop in task four while preserving performance in the upper range compared to the other constellations. The stable curve of the task-balanced close-to-center approach was expected as the task-balancing and the close-to-center strategy separately also showed the best results. In contrast, the task-balanced and the similarity-based strategy showed

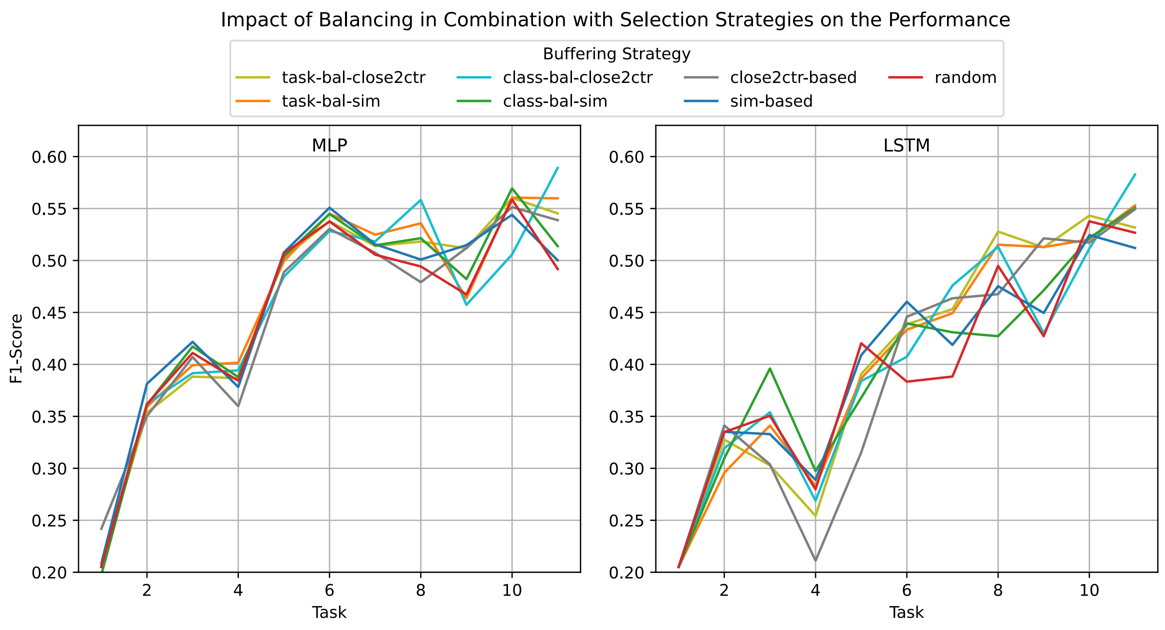


Figure 5.20: Impact of balancing in combination with different selection strategies on the average performance.
The diagram shows the performance curve (F1-Score) averaged over five repeated runs each for different buffer constellations combining balancing and selection strategies. Task-balancing and class-balancing are combined with the close-to-center and similarity strategies. For better comparison, both strategies' curves are also plotted without balancing.

fluctuation when used separately, but in combination, the fluctuations cancel each other out, resulting in a stable performance curve. This means the constellation benefits from the combination. Even though both constellations have good results regarding the average performance, they do not show consistently low ranges (Figure 5.21). Compared to the other constellations in the LSTM, the ranges are mostly in the lower ranges, but compared to the MLP, they are most time higher. In particular between tasks two and six, the MLP achieves much better results in this point, which can be traced back to the faster adaption of the MLP due to a smaller number of parameters.

5.4.8 Run with Optimized Parameters

In the experiments before, multiple parameters influencing continuous training were analyzed. In this run, all the parameters were chosen as optimal according to the previous analyses. It should be noted that the combination of separately optimized parameters must not necessarily lead to the optimal result. To get more insights into how the parameters influence each other further analyses are necessary.

According to the results of the previous experiments, the parameters for both models were chosen equally using five training iterations, 15 replayed examples per each ten new training examples, and a task-balanced buffer using the close-to-center selection strategy.

The performance results after each task and its comparison to the cumulative trained models are shown in Figure 5.22. It can be seen that the continuously trained MLP consistently achieves a higher performance than the LSTM, except for the first two tasks. The standard deviations of both models are equally high, which means the results between repeated runs

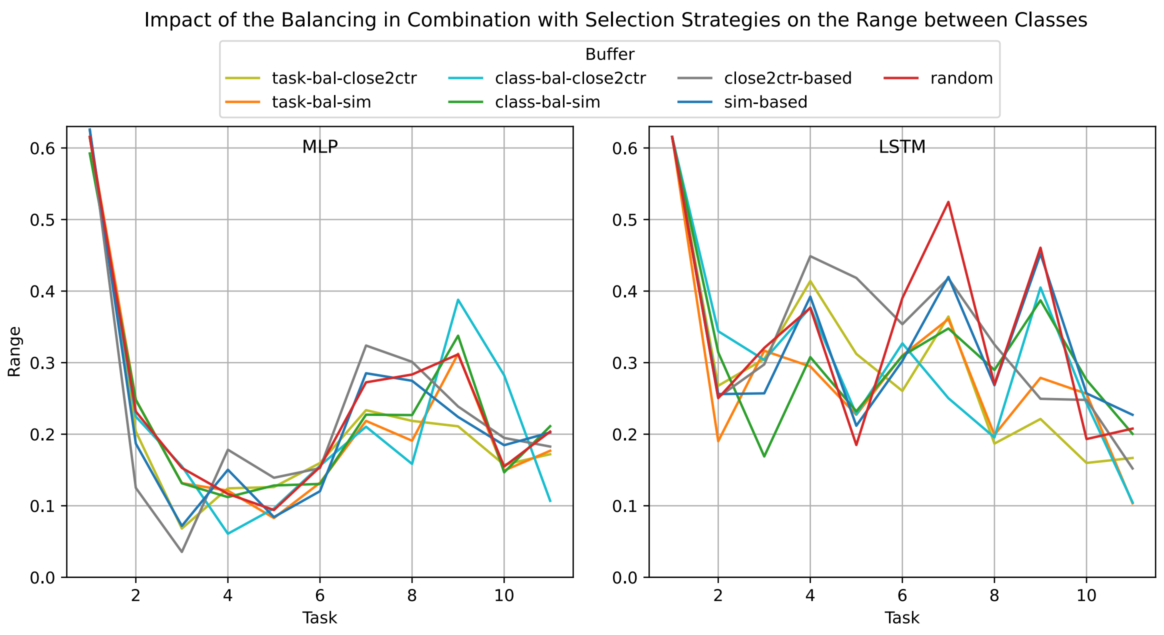


Figure 5.21: Impact of the buffer balancing in combination with selection strategy on the range of the performance of the classes.
The diagram visualizes the results of different combinations of buffer balancing (task-balanced, class-balanced, and unbalanced) and selection strategies (close-to-center, and similarity-based). The run called random (red) uses no balancing and random selection of elements to store in the buffer.

with the same parameter constellation vary approximately by the same amount. With a maximum standard deviation of 0.231 for the MLP and 0.303 for the LSTM, the variation is low, showing that influences of random processes (in particular the initialization) are largely eliminated.

Compared to the cumulative trained models, the results of the continuously trained MLP are only slightly lower than those of the cumulative trained MLP. When ignoring the results after task one, the difference is at maximum –0.0524 in task five and on average –0.0222 (negative means the continuous trained model performs worse than the cumulative trained model). For the LSTM, the difference is higher. At maximum, it is –0.1263 in task four and on average –0.0800, again ignoring the results after task one, as this is not representative and would distort the results. Even though the LSTM achieved slightly higher results in the cumulative training than the MLP, the opposite is true for continuous training. This leads to the conclusion that the less complex MLP is more suitable for continuous training than the complex LSTM. The higher number of trainable parameters in the LSTM slows down the process of the adaption to new data because more parameters must be fitted, which in turn requires more data.

In both continuously trained models, the range between the performance of the classes (Figure 5.23(a)) is very low compared to the previous experiments. Over large parts, the range is smaller than 0.2 for both models, whereas the LSTM has a higher standard deviation. Furthermore, the LSTM takes longer to stabilize visible at the higher range until task four. That both models are stable without much fluctuation is also confirmed by the curves of forgetting and BWT (Figure 5.23). Apart from the beginning of the curves, at task two, both models show a consistent but low forgetting only slightly greater than zero over most parts. This means,

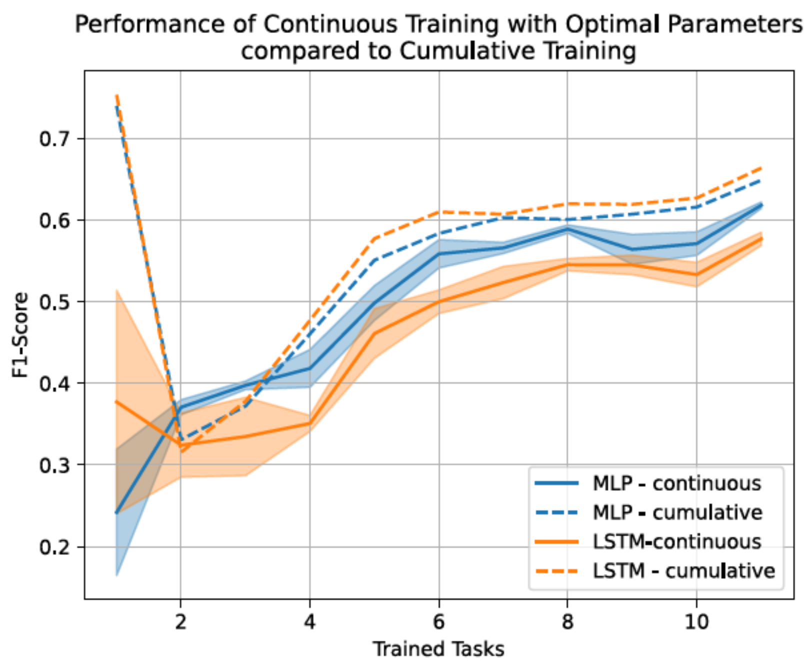
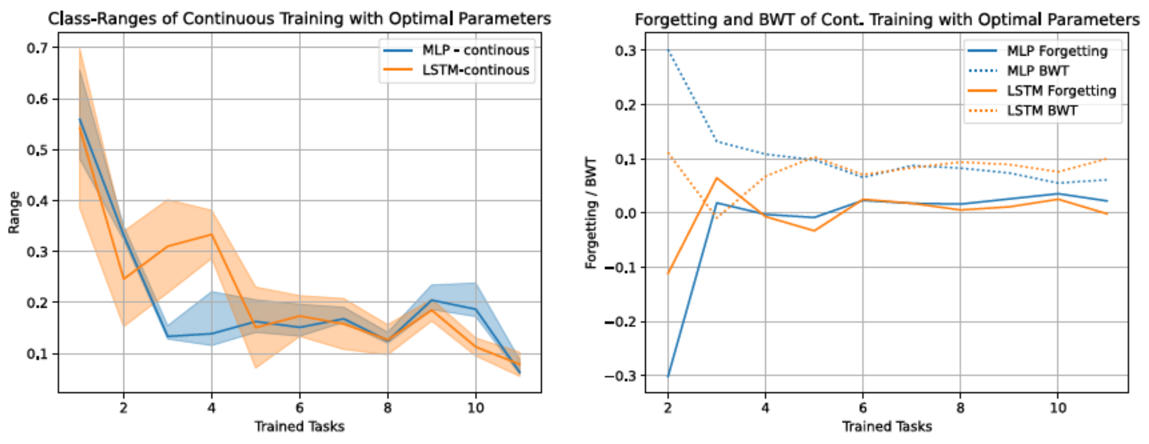


Figure 5.22: Comparison of cumulative and continuous training with separately optimized parameters.
The solid lines show the results of the continuously trained models compared to the cumulative trained models (dashed lines), which can be seen as an upper bound for the performance. The semi-transparent areas depict the standard deviation over five repeated runs. Detailed performance values for the continuously trained models are listed in Table A.7.

on average over all previously learned tasks, the models have lost only little knowledge compared to the maximum knowledge they ever had on each task. Additionally, the BWT is, except at task three for the LSTM, the BWT is always higher than the forgetting. Both BWT-curves are most of the time just below 0.1, whereas the BWT of the MLP slightly decreases at the end. The ratio of forgetting and BWT for both models means that learning new tasks helps to improve the performance of previously learned tasks on average by the value of the BWT compared to each task's performance directly after training it. As the forgetting is slightly positive most of the time, this means that even though some previous tasks improve, the model can not always keep maximum achieved performances for all tasks. However, considerable losses in the performance of tasks should be observed, as the forgetting is generally low.

5.5 Summary

In the previous experiments, the possible impact of four parameters on domain-incremental CL was analyzed. The parameters are the number of examples to replay for each ten new training examples, the number of training iterations for each batch, the balancing of the buffer, and the selection strategy to select elements to store in the buffer. The experiments were designed to find the impact of each of the parameters on the result. Because the buffer balancing and the selection strategy are more related as they both control the elements to store in the buffer, an experiment combining both parameters was done in Section 5.4.7. The conducted analyses can not necessarily be used to make conclusions about how the



(a) Class-Ranges (b) Forgetting and BWT

Figure 5.23: Class ranges and forgetting metrics for the run with optimized parameters. Diagram (a) shows the ranges between the best and worst performing classes for each task. In (b), forgetting and BWT are depicted.

parameters influence each other, as the parameters were mainly analyzed separately. This means, theoretically, the advantages of two parameter choices might cancel each other out when used together, for example. Such cross-relations cannot be ruled out.

For the number of examples to replay, it was found that it significantly impacts the model's stability while preserving sufficient plasticity. This, in turn, positively affects the average performance. The more examples replayed, the less fluctuation was observed in the performance curve, and the average F1-Score increased. The improvement in the performance was clearly visible for the MLP but only slight for the LSTM. Sudden performance drops could be eliminated for large parts in both models. Furthermore, a high number of replayed examples partly reduced the range between the performance of the classes. In the analysis, the best results were achieved with a number of 15 replayed examples, which was the highest number tested.

The main impact of the number of training iterations was seen in the performance. The highest performance was achieved with five iterations. However, in the LSTM, the difference to seven iterations was minimal and therefore equivalent. In the MLP, the differences between the tested numbers of iterations were generally smaller compared to the LSTM. It was found that for the LSTM, more than three iterations are necessary to fit it to new data properly. Too many iterations led to overfitting on the new data resulting in a lower performance compared to the results of the optimal number of iterations. Because of its lower complexity, the MLP adapted faster to new data. All runs with more than one iteration achieved comparable results in this case.

Regarding balancing of the buffer, a task-balanced buffer, a class-balanced buffer, and both combined were compared with an unbalanced buffer. The differences were marginal between all buffers. The task-balanced buffer was found to have a slight advantage regarding the resulting performance and lower, less fluctuation ranges between the classes' performances. This impact was not found in the MLP, however.

In close relation to buffer balancing is the strategy to select examples to store in the buffer. Three strategies implementing different ideas were compared with random selection. The classification-based strategy selecting primarily false classified examples showed by far the worst results. The differences between the other strategies were smaller. In the MLP, it is not possible to say which one is the best because of their minor differences in the performance as well as in the class ranges. For the LSTM, a difference is seen in the stability of the performance curves. Whereas the random and similarity-based (aiming at diversity) strategy show fluctuating performance curves, the one of the close-to-center selection strategy was much more stable without sudden drops after an initial training time until task four.

As balancing and the selection strategy both influence the examples stored in the buffer and are therefore particularity related, balancing was also analyzed in combination with different selection strategies. Again, in the MLP, the differences were minimal, leading to the result that the buffer does impact the results only on a low scale with the present balanced dataset. In the LSTM, the task-balanced constellations (task-balanced close-to-center and task-balanced similarity-based) showed partly a higher performance and were more stable regarding the performance than the other constellations, including the unbalanced buffer with random selection.

Finally, it was tested to what extent the parameters considered to be optimal beforehand improve the result. These were 15 replayed examples, five training iterations, and a task-balances buffer using the close-to-center selection strategy.

For the MLP, it was found that the combination of these parameter choices outperformed all previous results of the continuously trained models. The results in the experiment of the number of iterations are almost as high, but the final parameterization leads to a more stable model. This is seen in the performance curve, which shows no sudden performance drops, in the forgetting and BWT curves, and also the curve of the class ranges shows only a slight jump at the end. Therefore it can be concluded that the number of iterations has an impact on the average performance. However, the other parameters, likely at big parts the number of replayed elements, stabilize the model and prevent performance drops. The results of the continuously trained models come very close to the cumulative trained MLP, which only has an average difference of –0.0222 ignoring outlier values at task one.

The performance of the LSTM using the mentioned parameter choice is slightly lower than in the experiment of the number of iterations. However, the curve is more stable in the concluding experiment, which is also confirmed by the analyses of the ranges, forgetting, and BWT. In particular, forgetting and BWT are more stable and consistent than in the experiment analyzing the number of iterations. It can be concluded that, in this case, the performance has decreased in favor of higher stability. This mirrors the stability-plasticity-trade-off. Compared to the cumulative training, the difference is with –0.0800 on average (ignoring outliers at task one) higher than in the MLP.

In a nutshell, the final parameter choice improved the performance, stabilized the models, and reduced the differences in the performance of the classes compared to naive continuous training. The MLP could almost achieve the performance of the cumulative trained model, whereas the performance of the LSTM was slightly lower. This leads to the conclusion that the MLP is more suitable in this case, as it is less complex and, therefore, more flexible and adapts faster. The more complex LSTM needs more training data to adapt to new tasks, finally leading to lower performance.

6 Conclusion

This work consists of two parts. First, the challenges of CL were explained, including dataset shifts, CF, and the stability-plasticity dilemma. All three challenges are related, making it difficult to balance the prevention of learned knowledge (stability) and learning new tasks (plasticity). In this context, the main approaches were introduced: replay-based, regularization-based, and architectural methods. It was found that replay-based approaches are an intuitive way to prevent CF by storing selected examples in a buffer and replaying them later when learning on new data. Even though or because of its simplicity, replay is used in many approaches, and it is also state-of-the-art, often achieving good results. Furthermore, it can be well combined with other strategies.

Due to its relevance, a replay-based approach was practically implemented and analyzed regarding influential parameters in the second part of this work. A considerable proportion of work in the area of CL is done for image classification, however CL for text classification was rarely done. Nevertheless, CL in general became more popular in the recent three to five years resulting in a rising variability in the research. Furthermore, replay-based approaches have been examined, but rarely in terms of influential hyperparameters such as the number of examples replayed, the number of training iterations, and buffer constellations.

For these reasons, a data-incremental text classification task was chosen as the problem to solve in this study. The task was to classify book excerpts and blurbs into three given categories. Thereby the system starts learning from old books and progressively continues training with newer books. Therefore, the system needs to adapt to the books' changing language and writing styles over time.

For this task, a dataset based on books of the Gutenberg-Project and blurbs of the Goodreads web page was created first. It was divided into eleven tasks by successive time intervals. In all analyses, an MLP and LSTM using a replay-based approach were compared. The baseline was set by a naive approach without using replay or any other strategy to address CF and the stability-plasticity dilemma. On the other hand, cumulative trained models using all data at once were used as the upper bound of the achievable performance.

Results showed that replay improved the results compared to naive training in the case of the MLP clearly (approx. 0.11 higher F1-Score) and moderately for the LSTM (approx. 0,03 higher F1-Score). In contrast to the cumulative training, where the LSTM performed slightly better, the MLP clearly outperformed the LSTM in continuous training. The MLP showed only a slightly lower performance (-0.03), whereas the performance of the LSTM was approximately 0.09 worse than in the cumulative training. This is explained by the higher complexity of the LSTM compared to the MLP. Accordingly, a higher number of parameters must be fitted in the training process, which makes it adapt slower to new tasks.

For the analyzed parameters regarding research question two, it was found that the number of replayed examples and the number of training iterations have the most impact on the result. For the number of replayed examples, the results showed the higher the number of replayed examples, the better the results. At the highest, 15 replayed examples per ten new training examples were tested. It improved the performance and stabilized the model. For the number of training iterations, a number of five to seven iterations lead to the best results. If the number is too low, the model cannot acquire enough knowledge about the task. If the number of iterations is too high on the other side, the model is too plastic and overfits on new tasks while losing knowledge about old tasks. Regarding the buffer, in the present experiments, only marginal differences could be found with a slight advantage of a task-balanced buffer. Similarly, the selection strategy had only a minor impact. With the exception of the false-classification strategy, which resulted in a much lower performance than the other tested strategies, the differences were small. However, a slightly better and more stable performance was observed with the close-to-center selection strategy.

To cut it short, the analyses of replay-based CL on a data-incremental text classification task could clearly improve the results regarding the average performance and stability of the models compared to naive training. The better adaption of the MLP showed that it was in the present application more suitable than the LSTM.

7 Future Work

CL is a research topic that is still at its beginnings and got just recently a big boost. Therefore, also during the experiments and their analyses, multiple points for potential improvements and observations to be analyzed in more detail came up. Regarding the analyses, it would be helpful to further examine the class-wise and task-wise performance in more detail and evaluate in smaller intervals than task-wise to detect possible anomalies which cannot be seen in a task-wise evaluation (De Lange et al., 2023). In particular, for the analyses of the buffer constellation, this might show deeper insights into the impact of a class- or task-balanced buffer. Thereby, the experiments should also be done on (class-) unbalanced data to find out how a balanced buffer may increase the results in this case. This is, in particular, important for real-world applications, as data is often unbalanced.

In this context, it could also be helpful to track the distribution shifts in the data, which can then be used to draw conclusions regarding selection strategies deciding which elements to store in the buffer. Both are interrelated, as the selection strategy should ideally select elements such that the buffer also represents the distribution shift. Furthermore, the sampling strategy of examples from the buffer, which are replayed in each batch, can likely be improved. Even though regularly random sampling is used (Aljundi, Belilovsky, et al., 2019), it can be expected that sampling of elements to replay based on the new training data improves the results. For example, it is conceivable that replaying elements from the buffer that are more distant to the feature mean of the new training data might improve the results as this prevents overfitting to the new data and forgetting of previous knowledge.

Regarding the buffer-related parameters also the impact of the buffer size should be analyzed to minimize computational and memory resources without a loss in the performance.

Generally, after this work, it is also unclear how the analyzed parameters influence each other and which cross relations there are. In particular, for the LSTM, the run with the separately optimized parameters performed slightly worse than the best run in the experiment conducting the number of training iterations. This shows there are cross relations between parameters canceling advantages of each of the parameter choices out.

Finally, feature engineering was not done in the present case except for the word embeddings. More exhaustive experiments in this area and regarding hyper-parameters of the models, such as learning rate, optimizer, or loss function, could improve the results. However, this was not the scope of this work in the first order, but to get insights about influential replay parameters.

Appendix A: Performance Tables

A.1 Cumulative Training

Task	SVM		MLP		LSTM	
	Mean	Std	Mean	Std	Mean	Std
1	0.6000	0.0000	0.7396	0.1403	0.7531	0.1682
2	0.3782	0.0000	0.3310	0.0105	0.3157	0.0050
3	0.4110	0.0000	0.3721	0.0192	0.3781	0.0201
4	0.4566	0.0000	0.4606	0.0081	0.4771	0.0114
5	0.5923	0.0000	0.5507	0.0048	0.5772	0.0148
6	0.6080	0.0000	0.5834	0.0105	0.6098	0.0045
7	0.6214	0.0000	0.6029	0.0119	0.6070	0.0112
8	0.6264	0.0000	0.6004	0.0049	0.6197	0.0133
9	0.6277	0.0000	0.6070	0.0040	0.6191	0.0094
10	0.6318	0.0000	0.6157	0.0082	0.6268	0.0066
11	0.6549	0.0000	0.6486	0.0047	0.6635	0.0050

Table A.1: F1-Scores of cumulative trained baseline models.

The table lists the numerical values of the diagram in Figure 5.5(a). The column "Mean" shows the mean of the F1-Scores over the five runs. Column "Std" shows the standard deviation. The respective Experiment is described in Section 5.4.1.

A.2 Naive Training

Avg. F1-Score Task	MLP	LSTM
1	0.2051	0.3966
2	0.3342	0.2991
3	0.3285	0.2438
4	0.3218	0.2502
5	0.4443	0.4482
6	0.4756	0.4658
7	0.4683	0.4638
8	0.5179	0.5181
9	0.3619	0.4937
10	0.4542	0.4688
11	0.5085	0.5473

Table A.2: Average performance (F1-Scores) of naive trained MLP and LSTM.

Average F1-Score values over five repeated runs for the MLP and LSTM each. The respective experiment is described in Section 5.4.2. The diagram visualizing the results is shown in Figure 5.7(a).

A.3 Impact of Number Replayed Examples

Number Replayed Task	0	1	3	5	7	10	15
1	0.2051	0.2051	0.2051	0.2051	0.2009	0.2051	0.1974
2	0.3342	0.3645	0.3685	0.3549	0.3413	0.3722	0.3704
3	0.3285	0.4008	0.4028	0.4044	0.4071	0.4122	0.4115
4	0.3218	0.3534	0.3749	0.3895	0.3922	0.4028	0.4124
5	0.4443	0.4824	0.4947	0.4911	0.4975	0.5079	0.5118
6	0.4756	0.5209	0.5349	0.5447	0.5414	0.5422	0.5532
7	0.4683	0.4379	0.4880	0.4905	0.4946	0.5158	0.5503
8	0.5179	0.4800	0.5039	0.5019	0.5129	0.5266	0.5253
9	0.3619	0.3596	0.3788	0.4520	0.4621	0.5113	0.5416
10	0.4542	0.4697	0.5237	0.5390	0.5333	0.5746	0.5699
11	0.5085	0.4902	0.5138	0.5359	0.5474	0.5661	0.5808

Table A.3: F1-Scores of MLP with varied number of replayed examples.

Performance (F1-Scores) for the MLP of the experiment analyzing the impact of the number of replayed examples per training batch containing ten new examples (Section 5.4.3). The respective diagram is shown in Figure 5.10.

Number Replayed Task	0	1	3	5	7	10	15
1	0.3966	0.2051	0.2051	0.2051	0.2051	0.2530	0.2051
2	0.2991	0.3508	0.2843	0.3570	0.3607	0.3131	0.3373
3	0.2438	0.3269	0.3513	0.3593	0.3129	0.3415	0.3448
4	0.2502	0.2967	0.2871	0.3251	0.2902	0.3253	0.2829
5	0.4482	0.3833	0.3892	0.4097	0.3532	0.4389	0.4179
6	0.4658	0.3695	0.4432	0.3485	0.4390	0.4691	0.4809
7	0.4638	0.3495	0.3662	0.2952	0.3977	0.4859	0.5140
8	0.5181	0.4456	0.4533	0.4530	0.5076	0.5264	0.5216
9	0.4937	0.3568	0.3832	0.4216	0.4829	0.5041	0.5198
10	0.4688	0.4063	0.4549	0.4791	0.5246	0.5305	0.5321
11	0.5473	0.5138	0.5361	0.5093	0.5509	0.5746	0.5669

Table A.4: F1-Scores of LSTM with varied number of replayed examples.

Performance (F1-Scores) for the LSTM of the experiment analyzing the impact of the number of replayed examples per training batch containing ten new examples (Section 5.4.3). The respective diagram is shown in Figure 5.10.

A.4 Impact of Number of Training Iterations

Number Replayed Task	**1**	**3**	**5**	**7**	**9**	**15**
1	0.2036	0.2051	0.3636	0.4351	0.4197	0.4378
2	0.2999	0.3814	0.3526	0.3511	0.3509	0.3387
3	0.3828	0.4065	0.3937	0.3705	0.3700	0.3599
4	0.3616	0.3823	0.4076	0.4045	0.4186	0.3987
5	0.4583	0.5197	0.5193	0.5084	0.4950	0.5188
6	0.5024	0.5521	0.5518	0.5103	0.5158	0.5238
7	0.5078	0.5336	0.5503	0.5452	0.5352	0.5214
8	0.5133	0.5600	0.5884	0.5809	0.5829	0.5701
9	0.4136	0.5306	0.5461	0.5390	0.5466	0.5541
10	0.4968	0.5676	0.5719	0.5681	0.5490	0.5351
11	0.5391	0.5782	0.6109	0.6094	0.6090	0.5837

Table A.5: F1-Scores of MLP with varied number of training iterations.

Performance (F1-Scores) for the MLP of the experiment analyzing the impact of the number of training iterations per training batch (Section 5.4.4). The respective diagram is shown in Figure 5.13.

Number Replayed Task	**1**	**3**	**5**	**7**	**9**	**15**
1	0.2051	0.1974	0.4797	0.6146	0.4341	0.5902
2	0.3677	0.3353	0.3625	0.3122	0.3013	0.3039
3	0.3615	0.3441	0.3618	0.3469	0.3499	0.3418
4	0.3367	0.3196	0.3168	0.3258	0.3282	0.3666
5	0.3266	0.4107	0.4802	0.4804	0.4890	0.4940
6	0.4172	0.4708	0.5314	0.5330	0.5071	0.4920
7	0.3320	0.4900	0.5525	0.5465	0.5363	0.5120
8	0.3930	0.5072	0.5813	0.5717	0.5539	0.5455
9	0.3392	0.4862	0.5581	0.5565	0.5516	0.5316
10	0.4561	0.5388	0.5781	0.5667	0.5659	0.5282
11	0.4882	0.5665	0.6045	0.5952	0.5965	0.5847

Table A.6: F1-Scores of LSTM with varied number of training iterations.

Performance (F1-Scores) for the LSTM of the experiment analyzing the impact of the number of training iterations per training batch (Section 5.4.4). The respective diagram is shown in Figure 5.13.

A.5 Run with Optimized Parameters

Task	MLP F1	MLP Std	LSTM F1	LSTM Std
1	0.2421	0.0774	0.3772	0.1368
2	0.3707	0.0092	0.3244	0.0392
3	0.3975	0.0052	0.3349	0.0479
4	0.4183	0.0231	0.3509	0.0097
5	0.4984	0.0213	0.4610	0.0303
6	0.5585	0.0173	0.4999	0.0144
7	0.5658	0.0068	0.5234	0.0197
8	0.5888	0.0051	0.5453	0.0075
9	0.5640	0.0184	0.5449	0.0119
10	0.5710	0.0144	0.5331	0.0149
11	0.6180	0.0039	0.5769	0.0081

Table A.7: F1-Scores and standard deviation of the runs with supposed optimum parameters for MLP and LSTM.

Results of the run with optimized parameters using five iterations, 15 replayed examples per ten new training examples and a task-balanced buffer with close-to-center selection strategy. The respective diagram is shown in Figure 5.22.

Bibliography

Aljundi, R., Belilovsky, E., Tuytelaars, T., Charlin, L., Caccia, M., Lin, M., & Page-Caccia, L. (2019). Online continual learning with maximal interfered retrieval. *Advances in Neural Information Processing Systems 32 (NeurIPS 2019)*, *32*. Retrieved March 23, 2023, from https://proceedings.neurips.cc/paper/2019/hash/15825aee15eb335cc13f9b559f166ee8-Abstract.html

Aljundi, R., Lin, M., Goujaud, B., & Bengio, Y. (2019). Gradient based sample selection for online continual learning. *Proceedings of the 33rd International Conference on Neural Information Processing Systems*, (1058), 11817–11826. Retrieved March 20, 2023, from https://proceedings.neurips.cc/paper_files/paper/2019/file/e562cd9c0768d5464b64cf61da7fc6bb-Paper.pdf

Ataei, M., Erdogd, M., Kocak, S. A., Ben-David, S., Saleh, S., Pesaranghader, A., Alberts-Scherer, A., Sanchez, G., Pouryazdian, S., Ghazi, A., Nguyen, J., Khayrat, K., & Zhao, B. (2021). *Understanding dataset shift and potential remedies* (tech. rep.). Vector Institute. https://vectorinstitute.ai/wp-content/uploads/2021/08/ds_project_report_final_august9.pdf

Awasthi, A., & Sarawagi, S. (2019). Continual learning with neural networks: A review. *Proceedings of the ACM India Joint International Conference on Data Science and Management of Data*, 362–365. https://doi.org/10.1145/3297001.3297062

Benavides Prado, D. (2020). Beyond catastrophic forgetting in continual learning: An attempt with SVM. *Proceedings of the 37th International Conference on Machine Learning*. Retrieved March 8, 2023, from https://openrepository.aut.ac.nz/items/58f78c7a-55f8-43e6-863b-d86cedc6fa5e

Biesialska, M., Biesialska, K., & Costa-jussà, M. R. (2020). Continual lifelong learning in natural language processing: A survey. *Proceedings of the 28th International Conference on Computational Linguistics*, 6523–6541. https://doi.org/10.18653/v1/2020.coling-main.574

Bird, S., Loper, E., & Klein, E. (2009). *Natural language processing with python*. O'Reilly Media Inc.

Bittencourt, M. M., Silva, R. M., & Almeida, T. A. (2019). Ml-mdltext: A multilabel text categorization technique with incremental learning. *2019 8th Brazilian Conference on Intelligent Systems (BRACIS)*, 580–585. https://doi.org/10.1109/BRACIS.2019.00107

Blitzer, J., Dredze, M., & Pereira, F. (2007). Biographies, bollywood, boom-boxes and blenders: Domain adaptation for sentiment classification. *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, 440–447. Retrieved March 6, 2023, from https://aclanthology.org/P07-1056

Blitzer, J., McDonald, R., & Pereira, F. (2006). Domain adaptation with structural correspondence learning. *Proceedings of the 2006 Conference on Empirical Methods in Natural Language Processing*, 120–128. Retrieved June 12, 2023, from https://aclanthology.org/W06-1615

Bojanowski, P., Grave, E., Joulin, A., & Mikolov, T. (2017). Enriching word vectors with subword information [type: article], (arXiv:1607.04606). https://doi.org/https://doi.org/10.48550/arXiv.1607.04606

Capuano, N., Greco, L., Ritrovato, P., & Vento, M. (2021). Sentiment analysis for customer relationship management: An incremental learning approach. *Applied Intelligence*, *51*(6), 3339–3352. https://doi.org/10.1007/s10489-020-01984-x

Chaudhry, A., Dokania, P. K., Ajanthan, T., & Torr, P. H. S. (2018). Riemannian walk for incremental learning: Understanding forgetting and intransigence. In V. Ferrari, M. Hebert, C. Sminchisescu, & Y. Weiss (Eds.), *Computer vision – ECCV 2018* (pp. 556–572). Springer International Publishing. https://doi.org/10.1007/978-3-030-01252-6_33

Chaudhry, A., Ranzato, M., Rohrbach, M., & Elhoseiny, M. (2019). Efficient lifelong learning with a-GEM. *International Conference on Learning Representations*. Retrieved April 18, 2023, from https://openreview.net/forum?id=Hkf2_sC5FX

Chen, Z., & Liu, B. (2018). *Lifelong machine learning*. Springer International Publishing. https://doi.org/10.1007/978-3-031-01581-6

Chen, Z., Ma, N., & Liu, B. (2015). Lifelong learning for sentiment classification. *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, 750–756. https://doi.org/10.3115/v1/P15-2123

Cossu, A., Ziosi, M., & Lomonaco, V. (2021). Sustainable artificial intelligence through continual learning. *Proceedings of the 1st International Conference on AI for People: Towards Sustainable AI*. https://doi.org/http://dx.doi.org/10.4108/eai.20-11-2021.2314097

D'Autume, C. d. M., Ruder, S., Kong, L., & Yogatama, D. (2019). Episodic memory in lifelong language learning. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, & R. Garnett (Eds.), *Advances in neural information processing systems* (Vol. 32). Curran Associates, Inc. https://proceedings.neurips.cc/paper/2019/file/f8d2e80c1458ea2501f98a2cafadb397-Paper.pdf

De Lange, M., Aljundi, R., Masana, M., Parisot, S., Jia, X., Leonardis, A., Slabaugh, G., & Tuytelaars, T. (2021). A continual learning survey: Defying forgetting in classification tasks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 1–1. https://doi.org/10.1109/tpami.2021.3057446

De Lange, M., & Tuytelaars, T. (2021). Continual prototype evolution: Learning online from non-stationary data streams [ISSN: 2380-7504]. *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, 8230–8239. https://doi.org/10.1109/ICCV48922.2021.00814

De Lange, M., van de Ven, G. M., & Tuytelaars, T. (2023). Continual evaluation for lifelong learning: Identifying the stability gap. Retrieved March 30, 2023, from https://openreview.net/forum?id=Zy350cRstc6

Fernando, C., Banarse, D., Blundell, C., Zwols, Y., Ha, D., Rusu, A. A., Pritzel, A., & Wierstra, D. (2017). Pathnet: Evolution channels gradient descent in super neural networks. https://doi.org/10.48550/ARXIV.1701.08734

French, R. M. (1999). Catastrophic forgetting in connectionist networks. *Trends in Cognitive Sciences*, *3*(4), 128–135. https://doi.org/10.1016/S1364-6613(99)01294-2

Gama, J., Sebastião, R., & Rodrigues, P. P. (2013). On evaluating stream learning algorithms. *Machine Learning*, *90*(3), 317–346. https://doi.org/10.1007/s10994-012-5320-9

Gepperth, A., & Hammer, B. (2016). Incremental learning algorithms and applications. *The European Symposium on Artificial Neural Networks*. Retrieved March 7, 2023, from https://www.semanticscholar.org/paper/Incremental-learning-algorithms-and-applications-Gepperth-Hammer/796daa298ba0355808554913a072eccb5dd752a3

Goodreads LLC. (2023). *About goodreads*. Retrieved July 21, 2023, from https://www.goodreads.com/about/us

Goodrich, B., & Arel, I. (2014). Unsupervised neuron selection for mitigating catastrophic forgetting in neural networks [ISSN: 1558-3899]. *2014 IEEE 57th International Midwest Symposium on Circuits and Systems (MWSCAS)*, 997–1000. https://doi.org/10.1109/MWSCAS.2014.6908585

Google Cloud. (2023, July 19). *Google Cloud Guides: Machine learning workflow*. Retrieved July 23, 2023, from https://cloud.google.com/ai-platform/docs/ml-solutions-overview

Grossberg, S. (1982). How does a brain build a cognitive code? In S. Grossberg (Ed.), *Studies of mind and brain: Neural principles of learning, perception, development, cognition, and motor control* (pp. 1–52). Springer Netherlands. https://doi.org/10.1007/978-94-009-7758-7_1

Hassabis, D., Kumaran, D., Summerfield, C., & Botvinick, M. (2017). Neuroscience-inspired artificial intelligence. *Neuron*, *95*(2), 245–258. https://doi.org/10.1016/j.neuron.2017.06.011

Hayes, T. L., Kemker, R., Cahill, N. D., & Kanan, C. (2018). New metrics and experimental paradigms for continual learning [ISSN: 2160-7516]. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, 2112–21123. https://doi.org/10.1109/CVPRW.2018.00273

Hinton, G., Vinyals, O., & Dean, J. (2015). Distilling the knowledge in a neural network [type: article]. *NIPS Deep Learning and Representation Learning Workshop (2015)*, (arXiv:1503.02531). https://doi.org/10.48550/arXiv.1503.02531

Ho, S., Liu, M., Du, L., Gao, L., & Xiang, Y. (2023). Prototype-guided memory replay for continual learning. *IEEE Transactions on Neural Networks and Learning Systems*, 1–11. https://doi.org/10.1109/TNNLS.2023.3246049

Hsu, Y.-C., Liu, Y.-C., Ramasamy, A., & Kira, Z. (2018). Re-evaluating continual learning scenarios: A categorization and case for strong baselines. *NeurIPS Continual learning Workshop*. https://marcpickett.com/cl2018/CL-2018_paper_16.pdf

Hu, H., Sener, O., Sha, F., & Koltun, V. (2020). Drinking from a firehose: Continual learning with web-scale natural language. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 1–12. https://doi.org/10.1109/TPAMI.2022.3218265

Huang, Y., Zhang, Y., Chen, J., Wang, X., & Yang, D. (2021). Continual learning for text classification with information disentanglement based regularization. *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 2736–2746. https://doi.org/10.18653/v1/2021.naacl-main.218

Hurtado, J., Salvati, D., Semola, R., Bosio, M., & Lomonaco, V. (2023). Continual learning for predictive maintenance: Overview and challenges. *Intelligent Systems with Applications*, *19*, 200251. https://doi.org/https://doi.org/10.1016/j.iswa.2023.200251

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., & Hadsell, R. (2017). Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences*, *114*, 3521–3526. https://doi.org/10.1073/pnas.1611835114

Kumaran, D., Hassabis, D., & McClelland, J. L. (2016). What learning systems do intelligent agents need? complementary learning systems theory updated. *Trends in Cognitive Sciences*, *20*(7), 512–534. https://doi.org/10.1016/j.tics.2016.05.004

Lesort, T., Lomonaco, V., Stoian, A., Maltoni, D., Filliat, D., & Díaz-Rodríguez, N. (2020). Continual learning for robotics: Definition, framework, learning strategies, opportunities and challenges. *Information Fusion*, *58*, 52–68. https://doi.org/https://doi.org/10.1016/j.inffus.2019.12.004

Li, Z., & Hoiem, D. (2016). Learning without forgetting. In B. Leibe, J. Matas, N. Sebe, & M. Welling (Eds.), *Computer vision – ECCV 2016* (pp. 614–629). Springer International Publishing. https://doi.org/10.1007/978-3-319-46493-0_37

Library of Congress. (2022). *Introduction to library of congress subject headings (lcsh 44)*. Library of Congress. Retrieved May 9, 2023, from https://www.loc.gov/aba/publications/FreeLCSH/LCSH44-Main-intro.pdf

Lomonaco, V., Pellegrini, L., Cossu, A., Carta, A., Graffieti, G., Hayes, T. L., Lange, M. D., Masana, M., Pomponi, J., van de Ven, G. M., Mundt, M., She, Q., Cooper, K., Forest, J., Belouadah, E., Calderara, S., Parisi, G. I., Cuzzolin, F., Tolias, A. S., . . . Maltoni, D. (2021). Avalanche: An end-to-end library for continual learning. *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. https://doi.org/10.1109/cvprw53098.2021.00399

Lopez-Paz, D., & Ranzato, M. (2017). Gradient episodic memory for continual learning. *Proceedings of the 31st International Conference on Neural Information Processing Systems*, 6470–6479. Retrieved April 18, 2023, from https://dl.acm.org/doi/10.5555/3295222.3295393

Mallya, A., Davis, D., & Lazebnik, S. (2018). Piggyback: Adapting a single network to multiple tasks by learning to mask weights. In V. Ferrari, M. Hebert, C. Sminchisescu, & Y. Weiss (Eds.), *Computer vision – eccv 2018* (pp. 72–88). Springer International Publishing. https://doi.org/https://doi.org/10.1007/978-3-030-01225-0_5

Maltoni, D., & Lomonaco, V. (2019). Continuous learning in single-incremental-task scenarios. *Neural Networks*, *116*, 56–73. https://doi.org/10.1016/j.neunet.2019.03.010

Marcus, M. P., Santorini, B., & Marcinkiewicz, M. A. (1993). Building a large annotated corpus of english: The penn treebank. *Computational Linguistics*, *19*(2), 313–330. Retrieved July 13, 2023, from https://aclanthology.org/J93-2004

McClelland, J. L., McNaughton, B. L., & O'Reilly, R. C. (1995). Why there are complementary learning systems in the hippocampus and neocortex: Insights from the successes and failures of connectionist models of learning and memory. *Psychological Review*, *102*(3), 419–457. https://doi.org/10.1037/0033-295X.102.3.419

McCloskey, M., & Cohen, N. J. (1989, January 1). Catastrophic interference in connectionist networks: The sequential learning problem. In G. H. Bower (Ed.), *Psychology of learning and motivation* (pp. 109–165, Vol. 24). Academic Press. https://doi.org/10.1016/S0079-7421(08)60536-8

Mermillod, M., Bugaiska, A., & BONIN, P. (2013). The stability-plasticity dilemma: Investigating the continuum from catastrophic forgetting to age-limited learning effects. *Frontiers in Psychology*, *4*. Retrieved July 20, 2023, from https://www.frontiersin.org/articles/10.3389/fpsyg.2013.00504

Minaee, S., Kalchbrenner, N., Cambria, E., Nikzad, N., Chenaghlu, M., & Gao, J. (2021). Deep learning–based text classification: A comprehensive review. *ACM Computing Surveys*, *54*(3), 1–40. https://doi.org/10.1145/3439726

Mirzadeh, S. I., Farajtabar, M., Pascanu, R., & Ghasemzadeh, H. (2020). Understanding the role of training regimes in continual learning. *Advances in Neural Information Processing Systems*, *33*, 7308–7320. Retrieved July 10, 2023, from https://proceedings.neurips.cc//paper/2020/hash/518a38cc9a0173d0b2dc088166981cf8-Abstract.html

Mundt, M., Lang, S., Delfosse, Q., & Kersting, K. (2022). CLEVA-compass: A continual learning evaluation assessment compass to promote research transparency and comparability. *International Conference on Learning Representations*. https://openreview.net/forum?id=rHMaBYbkkRJ

Next Move Strategy Consulting. (2023, January 15). *Artificial intelligence (ai) market size worldwide in 2021 with a forecast until 2030 (in million u.s. dollars) [graph]*. Statista. Retrieved July 24, 2023, from https://www.statista.com/statistics/1365145/artificial-intelligence-market-size/

Otter, D. W., Medina, J. R., & Kalita, J. K. (2021). A survey of the usages of deep learning for natural language processing. *IEEE Transactions on Neural Networks and Learning Systems*, *32*(2), 604–624. https://doi.org/10.1109/TNNLS.2020.2979670

Pan, S. J., & Yang, Q. (2010). A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering*, *22*(10), 1345–1359. https://doi.org/10.1109/TKDE.2009.191

Parisi, G. I., Kemker, R., Part, J. L., Kanan, C., & Wermter, S. (2019). Continual lifelong learning with neural networks: A review. *Neural Networks*, *113*, 54–71. https://doi.org/10.1016/j.neunet.2019.01.012

Quiñonero-Candela, J., Sugiyama, M., Schwaighofer, A., & Lawrence, N. D. (Eds.). (2008, December 12). *Dataset shift in machine learning*. The MIT Press. https://doi.org/10.7551/mitpress/9780262170055.001.0001

Ramalho, T., & Garnelo, M. (2018). Adaptive posterior learning: Few-shot learning with a surprise-based memory module. *International Conference on Learning Representations*. Retrieved June 8, 2023, from https://openreview.net/forum?id=ByeSdsC9Km

Ratcliff, R. (1990). Connectionist models of recognition memory: Constraints imposed by learning and forgetting functions. *Psychological Review*, *97*(2), 285–308. https://doi.org/10.1037/0033-295x.97.2.285

Rebuffi, S.-A., Kolesnikov, A., Sperl, G., & Lampert, C. H. (2017). iCaRL: Incremental classifier and representation learning. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. https://doi.org/10.1109/cvpr.2017.587

Robins, A. (1995). Catastrophic forgetting, rehearsal and pseudorehearsal. *Connection Science*, *7*, 123–146. https://doi.org/10.1080/09540099550039318

Rodríguez, N. D., Lomonaco, V., Filliat, D., & Maltoni, D. (2018). Don't forget, there is more than forgetting: New metrics for continual learning. Retrieved May 13, 2023, from https://marcpickett.com/cl2018/CL-2018_paper_5.pdf

Rosenblatt, F. (1957). *The perceptron - a perceiving and recognizing automaton* (tech. rep. No. 85-460-1). Cornell Aeronautical Laboratory. Ithaca, New York. https://blogs.umass.edu/brain-wars/files/2016/03/rosenblatt-1957.pdf

Rusu, A. A., Rabinowitz, N. C., Desjardins, G., Soyer, H., Kirkpatrick, J., Kavukcuoglu, K., Pascanu, R., & Hadsell, R. (2016). Progressive neural networks [version: 1 type: article], (arXiv:1606.04671). https://doi.org/10.48550/arXiv.1606.04671

Serra, J., Suris, D., Miron, M., & Karatzoglou, A. (2018). Overcoming catastrophic forgetting with hard attention to the task. *Proceedings of the 35th International Conference on Machine Learning*, 4548–4557. Retrieved April 28, 2023, from https://proceedings.mlr.press/v80/serra18a.html

Shan, G., Xu, S., Yang, L., Jia, S., & Xiang, Y. (2020). Learn#: A novel incremental learning method for text classification. *Expert Systems with Applications*, *147*, 113198. https://doi.org/10.1016/j.eswa.2020.113198

Shin, H., Lee, J. K., Kim, J., & Kim, J. (2017). Continual learning with deep generative replay. *Proceedings of the 31st International Conference on Neural Information Processing Systems*, 2994–3003. Retrieved May 30, 2023, from https://dl.acm.org/doi/10.5555/3294996.3295059

Sodhani, S., Faramarzi, M., Mehta, S. V., Malviya, P., Abdelsalam, M., Janarthanan, J., & Chandar, S. (2022). An introduction to lifelong supervised learning. https://doi.org/10.48550/ARXIV.2207.04354

Sprechmann, P., Jayakumar, S. M., Rae, J. W., Pritzel, A., Badia, A. P., Uria, B., Vinyals, O., Hassabis, D., Pascanu, R., & Blundell, C. (2018). Memory-based parameter adaptation. *International Conference on Learning Representations*. Retrieved July 14, 2023, from https://openreview.net/forum?id=rkfOvGbCW

Stanford University. (2023, April 23). *Global total corporate artificial intelligence (ai) investment from 2015 to 2022 (in billion u.s. dollars) [graph]*. Statista. Retrieved July 24, 2023, from https://www.statista.com/statistics/941137/ai-investment-and-funding-worldwide/

Struß, J. M., Siegel, M., Ruppenhofer, J., Wiegand, M., & Klenner, M. (2019). Overview of germeval task 2, 2019 shared task on the identification of offensive language. *Proceedings of the 15th Conference on Natural Language Processing (KONVENS 2019)*, 354–365. Retrieved May 29, 2023, from https://konvens.org/proceedings/2019/papers/germeval/GermEvalSharedTask2019Iggsa.pdf

Sun, F.-K., Ho, C.-H., & Lee, H.-Y. (2020). Lamol: Language modeling is all you need for lifelong language learning. *International Conference on Learning Representations*. https://openreview.net/forum?id=PxD6JH09Eme

Sun, J., Wang, S., Zhang, J., & Zong, C. (2020). Distill and replay for continual language learning. *Proceedings of the 28th International Conference on Computational Linguistics*, 3569–3579. https://doi.org/10.18653/v1/2020.coling-main.318

Thrun, S., & Mitchell, T. M. (1995). Lifelong robot learning. *Robotics and Autonomous Systems*, *15*(1), 25–46. https://doi.org/10.1016/0921-8890(95)00004-Y

Tiwari, R., Killamsetty, K., Iyer, R., & Shenoy, P. (2022). GCR: Gradient coreset based replay buffer selection for continual learning [ISSN: 2575-7075]. *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 99–108. https://doi.org/10.1109/CVPR52688.2022.00020

Toneva, M., Sordoni, A., des Combes, R. T., Trischler, A., Bengio, Y., & Gordon, G. J. (2018). An empirical study of example forgetting during deep neural network learning. *International Conference on Learning Representations*. Retrieved June 8, 2023, from https://openreview.net/forum?id=BJlxm30cKm

van de Ven, G. M., & Tolias, A. S. (2019, April 15). Three scenarios for continual learning [type: article]. https://doi.org/10.48550/arXiv.1904.07734

van de Ven, G. M., Tuytelaars, T., & Tolias, A. S. (2022). Three types of incremental learning. *Nature Machine Intelligence*, *4*(12), 1185–1197. OriginalPaper. https://doi.org/10.1038/s42256-022-00568-3

Wang, L., Zhang, X., Su, H., & Zhu, J. (2023, January 31). A comprehensive survey of continual learning: Theory, method and application [type: article]. https://doi.org/10.48550/arXiv.2302.00487

Wang, Z., Zhan, Z., Gong, Y., Yuan, G., Niu, W., Jian, T., Ren, B., Ioannidis, S., Wang, Y., & Dy, J. (2022). SparCL: Sparse continual learning on the edge. In A. H. Oh, A. Agarwal, D. Belgrave, & K. Cho (Eds.), *Advances in neural information processing systems*. https://openreview.net/forum?id=n0dD3d54Wgf

Weaver, S., Baird, L., & Polycarpou, M. (1998). Preventing unlearning during online training of feedforward networks [ISSN: 2158-9860]. *Proceedings of the 1998 IEEE International Symposium on Intelligent Control (ISIC) held jointly with IEEE International Symposium on Computational Intelligence in Robotics and Automation (CIRA) Intell*, 359–364. https://doi.org/10.1109/ISIC.1998.713688

Widmer, G., & Kubat, M. (1993). Effective learning in dynamic environments by explicit context tracking. In P. B. Brazdil (Ed.), *Machine learning: ECML-93* (pp. 227–243). Springer. https://doi.org/10.1007/3-540-56602-3_139

Xiao, Z., Du, Z., Wang, R., Gan, R., & Li, J. (2023). Online continual learning with declarative memory. *Neural Networks*. https://doi.org/10.1016/j.neunet.2023.03.025

Zenke, F., Poole, B., & Ganguli, S. (2017). Continual learning through synaptic intelligence. *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, 3987–3995.

Zhang, W., Deng, Y., Liu, B., Pan, S. J., & Bing, L. (2023, May 24). *Sentiment analysis in the era of large language models: A reality check* (arXiv:2305.15005) (type: article). arXiv. https://doi.org/10.48550/arXiv.2305.15005

Zhang, X., Zhao, J., & LeCun, Y. (2015). Character-level convolutional networks for text classification. *Advances in Neural Information Processing Systems*, *28*. Retrieved February 24, 2023, from https://proceedings.neurips.cc/paper/2015/hash/250cf8b51c773f3f8dc8b4be867a9a02-Abstract.html

Statutory Declaration in Lieu of an Oath

I – Christoph Demus – do hereby declare in lieu of an oath that I have composed the presented work independently on my own and without any other resources than the ones given.

All thoughts taken directly or indirectly from external sources are correctly acknowledged.

This work has neither been previously submitted to another authority nor has it been published yet.

Mittweida, 30. July 2023	[redacted]
Location, Date	Christoph Demus