IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 17 February 2025, accepted 5 March 2025, date of publication 13 March 2025, date of current version 21 March 2025.

Digital Object Identifier 10.1109/ACCESS.2025.3551146

RESEARCH ARTICLE

Continual Learning With Quasi-Newton Methods

STEVEN VANDER EECKT AND HUGO VAN HAMME, (Senior Member, IEEE)
Department of Electrical Engineering ESAT-PSI, KU Leuven, 3001 Leuven, Belgium
Corresponding author: Steven Vander Eeckt (steven.vandereeckt@esat.kuleuven.be)

This work was supported by the Research Foundation Flanders (FWO) of the SBO Programme under Grant S004923N.

ABSTRACT Catastrophic forgetting remains a major challenge when neural networks learn tasks sequentially. Elastic Weight Consolidation (EWC) attempts to address this problem by introducing a Bayesian-inspired regularization loss to preserve knowledge of previously learned tasks. However, EWC relies on a Laplace approximation where the Hessian is simplified to the diagonal of the Fisher information matrix, assuming uncorrelated model parameters. This overly simplistic assumption often leads to poor Hessian estimates, limiting its effectiveness. To overcome this limitation, we introduce Continual Learning with Sampled Quasi-Newton (CSQN), which leverages Quasi-Newton methods to compute more accurate Hessian approximations. CSQN captures parameter interactions beyond the diagonal without requiring architecture-specific modifications, making it applicable across diverse tasks and architectures. Experimental results across four benchmarks demonstrate that CSQN consistently outperforms EWC and other state-of-the-art baselines, including rehearsal-based methods. CSQN reduces EWC's forgetting by 50% and improves its performance by 8% on average. Notably, CSQN achieves superior results on three out of four benchmarks, including the most challenging scenarios, highlighting its potential as a robust solution for continual learning.

INDEX TERMS Artificial neural networks, catastrophic forgetting, continual learning, quasi-Newton methods.

I. INTRODUCTION

Since the 2010s, Artificial Neural Networks (ANNs) have been able to match or even surpass human performance on a wide variety of tasks. However, when presented with a set of tasks to be learned sequentially—a setting referred to as Continual Learning (CL)—ANNs suffer from catastrophic forgetting [1]. Unlike humans, ANNs struggle to retain previously learned knowledge when extending their knowledge. Naively adapting an ANN to a new task generally leads to a deterioration in the network's performance on previous tasks.

Many CL methods have been proposed to alleviate catastrophic forgetting. One of the most well-known is Elastic Weight Consolidation (EWC) [2], which approaches CL from a Bayesian perspective. After training on a task, EWC uses Laplace approximation [3] to estimate a posterior distribution over the model parameters for that task. When training on the next task, this posterior is used via a regularization loss to prevent the model from catastrophically forgetting the previous task. To estimate the Hessian, which is needed in the Laplace approximation to measure the (un)certainty of the model parameters, EWC uses the Fisher Information Matrix (FIM). Furthermore, to simplify the computation, EWC assumes that the FIM is approximately diagonal. EWC thus uses the diagonal of the FIM to estimate the Hessian.

While EWC mitigates catastrophic forgetting, its effectiveness is limited by its assumption of an approximately diagonal FIM. This assumption oversimplifies parameter correlations, leading to poor Hessian estimates and suboptimal posterior approximations. These inaccuracies might cause EWC to struggle in complex scenarios with significant parameter interactions, severely impacting its performance in continual learning.

Hence, we propose Continual Learning with Sampled Quasi-Newton (CSQN), a method designed to overcome the limitations of EWC's diagonal Hessian approximation. CSQN leverages Quasi-Newton (QN) methods [4]. QN methods, being second-order optimization algorithms, involve (inverse) Hessian approximations that are updated at each iteration (without being explicitly computed) to find local optima. Two such methods are BFGS (Broyden, Fletcher, Goldfarb, Shanno) and SR1 (Symmetric Rank-1), which apply rank-2 and rank-1 updates, respectively, to the

The associate editor coordinating the review of this manuscript and approving it for publication was Mu-Yen Chen.

© 2025 The Authors. This work is licensed under a Creative Commons Attribution 4.0 License.
For more information, see https://creativecommons.org/licenses/by/4.0/

Hessian approximation at each iteration. Since our primary interest is in the Hessian approximations from QN methods, training the model with QN methods solely to obtain these approximations would be impractical. Therefore, we propose using Sampled Quasi-Newton (SQN) methods [5]. In SQN methods, the Hessian approximation is obtained by sampling around the current estimate of the local optimum, meaning that the Hessian is computed anew at each iteration without relying on previous ones. Furthermore, to enhance our method and make it a direct extension of EWC, we use EWC's Hessian estimate, namely the diagonal of the FIM, as the initial Hessian approximation. Using SQN, we then improve this Hessian approximation, which is no longer restricted to being diagonal. Similar to EWC, this improved Hessian approximation is used in the regularization loss to prevent forgetting previous tasks while learning new ones.

We evaluate our proposed method through four benchmarks, comparing it to the most relevant and competitive baselines. These include EWC, which our method extends and improves, as well as other CL methods, including those relying on the storage of past data. Our results show that CSQN consistently outperforms these baselines across all benchmarks, demonstrating superior performance after learning multiple tasks in each experiment. Additionally, we enhance our method by proposing mechanisms to reduce the memory requirements of CSQN and evaluate their impact. Our findings indicate that the performance loss associated with these memory-efficient versions is limited, suggesting that CSQN can achieve a balance between effectiveness and memory usage.

This paper is structured as follows. We start by reviewing related work in Section II. Next, in Section III, we explain our proposed method (Section III-E) after a short review of EWC (Section III-B) and (Sampled) Quasi-Newton methods (Section III-D). In Section IV, we present the experimental results and compare our method to EWC and other baselines. Section V discusses the strengths and challenges of the proposed CSQN method. Finally, Section VI concludes the paper.

II. RELATED WORK

Continual Learning (CL) methods are generally grouped into three main categories: regularization-based, rehearsal-based, and parameter-isolation methods [6]. Each of these categories tackles catastrophic forgetting using distinct approaches to retain knowledge of previously learned tasks.

For a more extensive overview of recent advancements and categorizations in Continual Learning, readers are encouraged to consult recent comprehensive reviews such as [7], [8]. The following discussion focuses on methods most relevant to our work.

A. REGULARIZATION-BASED METHODS

To prevent catastrophic forgetting, regularization-based methods regularize the learning of new tasks. Following [6], two classes of regularization-based methods are prior-focused methods and data-focused methods. EWC, using the diagonal of the FIM to estimate the importance of parameters to previous tasks, belongs to the prior-focused methods. Similar to EWC are Synaptic Intelligence (SI) [9] and Memory-Aware Synapses (MAS) [10], which compute the importance weights of the model parameters differently. While SI estimates the importance of parameters by accumulating each parameter's contribution to the reduction in the loss, MAS considers the gradients of the squared norm of the model's output with respect to each parameter. Similar work to EWC, MAS and SI includes [11], [12], [13].

Many works build on EWC's ideas. Online EWC [14] allows EWC to be used in an online way, without requiring task boundaries. Reference [15] proposes Riemanian Walk, which combines the regularization losses of Online EWC and SI. Moreover, to address EWC's assumption of an approximately diagonal FIM, R-EWC (Rotated-EWC) [16] rotates the parameter space layer-wise such that the corresponding FIM is indeed approximately diagonal. While R-EWC improves the performance of EWC when adapting one task to another, R-EWC applies the reparametrization only to the previous task and lacks a mechanism to combine the importance weights of all tasks learned so far. Meanwhile, [17] proposes to use Kronecker factored approximations of the FIM [18], which allows to compute and store the FIM efficiently for each layer, thus obtaining a block diagonal approximation of the FIM. Unlike EWC, their method considers interactions between model parameters within the same layer but not across layers. This method was extended by [19] to better accommodate batch normalization [19].

A second class of regularization-based methods is data-focused methods, whose main idea is knowledge distillation [20], to transfer knowledge from the previous to the current model. Examples include Learning Without Forgetting (LWF) [21], but also [22], [23]. Many rehearsal-based methods (discussed in Section II-B), relying on the storage of some past data, incorporate ideas from data-focused methods, e.g. [24], [25], [26].

Instead of using a regularization loss, other methods constrain the gradients when learning new tasks, which some reviews, such as [7], consider as a separate category of CL methods. Examples include Orthogonal Gradient Descent (OGD) [27], which updates the gradient used in Stochastic Gradient Descent (SGD) to make it orthogonal to the gradients of previous tasks; Gradient Projection Memory (GPM) [28], which partitions the gradient space into two orthogonal subspaces, forcing the gradient to lie in the subspace least harmful to previous tasks; as well as the work of [29], [30], [31], and [32].

B. REHEARSAL-BASED METHODS

Rehearsal-based methods assume access to a memory of representative samples of previous tasks. Most straightforward is Experience Replay (ER) [33], which trains the model jointly on a mini-batch from the new task and a mini-batch

from the memory. Gradient Episodic Memory (GEM) [34] overlaps with both rehearsal-based and regularization-based methods [35]. GEM introduces a regularization loss based on a set of representative samples, assuring SGD's gradient updates do not increase the loss of the representative samples. GEM has been extended in [36] and [37]. Similar to GEM is [38]. Other rehearsal-based methods which incorporate strategies from regularization-based methods are [24], [25], [26]. Rehearsal-based CL has been studied and analysed in [39].

Within the rehearsal-based methods, the pseudo-rehearsal methods, instead of requiring access to a memory of representative samples, train a generator to simulate data from previous tasks. Examples include [40], [41], [42].

C. PARAMETER-ISOLATION METHODS

Parameter-isolation methods expand the network with each new task, to avoid interference between tasks. Examples include [14], [43], [44], [45], [46]. The former proposes Progress & Compress (P&C) [14], which uses online EWC in the 'progress' stage. Another method on the intersection between parameter-isolation and regularization-based methods is AR1 [35], which combines Copy Weight with Re-init (CWR) [47] with SI and EWC.

III. CONTINUAL LEARNING WITH SAMPLED QUASI-NEWTON

In this section, we introduce our proposed Continual Learning (CL) method, Continual Learning with Sampled Quasi-Newton (CSQN). To provide the necessary context, we first formulate the problem and review Elastic Weight Consolidation (EWC), a foundational regularization-based method in CL. Next, we discuss the limitations of EWC and the motivation for CSQN, highlighting the need for more accurate Hessian approximations to overcome EWC's shortcomings. Finally, we provide an overview of (Sampled) Quasi-Newton methods, which form the foundation of our approach.

A. PROBLEM FORMULATION

We have a sequence of T tasks, each represented by data $\mathcal{D}_1, \mathcal{D}_2, \ldots, \mathcal{D}_T$. For each $(\boldsymbol{x}, y) \in \mathcal{D}_i$, $\boldsymbol{x} \in \mathbb{R}^d$ is the model input, and y is the corresponding class label. The tasks are learned in sequence and, during training of any task t, the model has only access to $\mathcal{D}_t$. The goal is to learn task t without forgetting the knowledge extracted from $\mathcal{D}_1, \mathcal{D}_2 \ldots, \mathcal{D}_{t-1}$, i.e. if $\boldsymbol{\theta} \in \mathbb{R}^N$ are the model's parameters, and $\boldsymbol{\theta}^{(t)}$ are the parameters after learning the first t tasks in sequence, then the objective is:

$$\boldsymbol{\theta}^{(T)} = \arg\max_{\boldsymbol{\theta}} P(\boldsymbol{\theta}|\mathcal{D}_1, \mathcal{D}_2, \ldots, \mathcal{D}_T) \tag{1}$$

where $\boldsymbol{\theta}^{(t)}$ is obtained by training $\boldsymbol{\theta}^{(t-1)}$ on $\mathcal{D}_t$ without access to $\mathcal{D}_1, \mathcal{D}_2, \ldots, \mathcal{D}_{t-1}$.

We denote by $f_{\boldsymbol{\theta}}(\cdot)$ the model with parameters $\boldsymbol{\theta}$. $\boldsymbol{p}_{\boldsymbol{\theta}}(y|\boldsymbol{x})$ is its softmax output for class y given input $\boldsymbol{x}$, while $\mathcal{L}_t^{ce}(\boldsymbol{\theta})$ is its cross-entropy loss on task t (thus computed on $\mathcal{D}_t$).

B. ELASTIC WEIGHT CONSOLIDATION

EWC approaches CL from a Bayesian perspective. When learning a second task, the optimal parameters $\boldsymbol{\theta}^{(2)}$ are found as follows:

$$\begin{aligned}\boldsymbol{\theta}^{(2)} &= \arg\max_{\boldsymbol{\theta}} \log P(\boldsymbol{\theta}|\mathcal{D}_1, \mathcal{D}_2) \\ &= \arg\max_{\boldsymbol{\theta}} \log P(\mathcal{D}_2|\boldsymbol{\theta}) + \log P(\boldsymbol{\theta}|\mathcal{D}_1)\end{aligned} \tag{2}$$

This is obtained by applying Bayes' rule to $P(\boldsymbol{\theta}|\mathcal{D}_1, \mathcal{D}_2)$ (the objective from Equation 1), assuming that $\mathcal{D}_1$ and $\mathcal{D}_2$ are independent [2]. The first term on the right-hand side of Equation 2 corresponds to optimizing the cross-entropy loss for $\mathcal{D}_2$, i.e. $\mathcal{L}_2^{ce}(\boldsymbol{\theta})$; the second term, the posterior after training on task 1, is intractable. However, it can be estimated using the Laplace approximation [3]:

$$\log P(\boldsymbol{\theta}|\mathcal{D}_1) = -\frac{\lambda}{2}(\boldsymbol{\theta} - \boldsymbol{\theta}^{(1)})^T \boldsymbol{\Lambda} (\boldsymbol{\theta} - \boldsymbol{\theta}^{(1)}) \tag{3}$$

In Equation 3, $\boldsymbol{\Lambda}$ represents the Hessian of $\mathcal{L}_1^{ce}(\boldsymbol{\theta}^{(1)})$. The Laplace approximation models the posterior $P(\boldsymbol{\theta}|\mathcal{D}_1)$ as a normal distribution $N(\boldsymbol{\theta}^{(1)}, \boldsymbol{\Lambda}^{-1})$. To estimate the Hessian $\boldsymbol{\Lambda}$, EWC utilizes the Fisher Information Matrix (FIM), defined as:

$$\boldsymbol{F}_{\boldsymbol{\theta}}^{(t)} = \mathbb{E}_{(\boldsymbol{x},y)\sim\mathcal{D}_t}\left[\left(\frac{\partial \log \boldsymbol{p}_{\boldsymbol{\theta}}(y|\boldsymbol{x})}{\partial \boldsymbol{\theta}}\right)\left(\frac{\partial \log \boldsymbol{p}_{\boldsymbol{\theta}}(y|\boldsymbol{x})}{\partial \boldsymbol{\theta}}\right)^T\right] \tag{4}$$

To simplify, EWC approximates the FIM by its main diagonal. Combining Equations 2, 3 and the assumption that the FIM is diagonal, the loss during training task 2 becomes the following:

$$\mathcal{L}_2(\boldsymbol{\theta}) = \mathcal{L}_2^{ce}(\boldsymbol{\theta}) + \frac{\lambda}{2}\sum_{i=1}^{N} \Omega_i^{(1)} \left(\boldsymbol{\theta}_i - \boldsymbol{\theta}_i^{(1)}\right)^2 \tag{5}$$

With $\boldsymbol{\Omega}^{(1)} \in \mathbb{R}^N$ the diagonal of $\boldsymbol{F}_{\boldsymbol{\theta}^{(1)}}^{(1)}$, $\boldsymbol{\theta}_i$ the ith parameter of the model $f_{\boldsymbol{\theta}}(\cdot)$ and λ a hyper-parameter, determining the weight of the regularization. $\Omega_i^{(1)}$ can be considered an estimate of how important $\theta_i^{(1)}$ is to $f_{\boldsymbol{\theta}^{(1)}}$ for task 1.

To extend EWC to more than two tasks, a regularization loss can be introduced for each term. However, [48] suggests combining these regularization losses in one term. When learning task $t+1$, one then obtains:

$$\mathcal{L}_{t+1}(\boldsymbol{\theta}) = \mathcal{L}_{t+1}^{ce}(\boldsymbol{\theta}) + \frac{\lambda}{2}\sum_{i=1}^{N}\left(\sum_{j=1}^{t}\Omega_i^{(j)}\right)\left(\boldsymbol{\theta}_i - \boldsymbol{\theta}_i^{(t)}\right)^2 \tag{6}$$

Therefore, it suffices to store $\sum_{j=1}^{t}\boldsymbol{\Omega}^{(j)}$ and $\boldsymbol{\theta}^{(t)}$ in order to learn task $t+1$ with EWC.

C. LIMITATIONS OF EWC AND MOTIVATION FOR CSQN

While EWC mitigates catastrophic forgetting, its effectiveness is limited by its assumption that FIM is approximately diagonal. This assumption oversimplifies parameter correlations, and in many cases the diagonal of the FIM captures only a small portion of the total information. As a result, EWC's Hessian approximation is often inaccurate, leading to suboptimal posterior estimation and reduced performance in continual learning tasks.

A method that overcomes the diagonal FIM assumption while maintaining computational feasibility is needed. By leveraging more accurate Hessian approximations, such a method could better estimate the posterior distribution, effectively preserving knowledge across tasks and addressing the shortcomings of EWC. This is the motivation behind Continual Learning with Sampled Quasi-Newton (CSQN), which we propose as a solution to these challenges.

D. (SAMPLED) QUASI-NEWTON METHODS

To improve EWC's poor Hessian approximation, we explore the use of Quasi-Newton (QN) methods. Before diving into our approach, we first review QN and Sampled Quasi-Newton (SQN) methods to provide the necessary background.

1) QUASI-NEWTON METHODS

Given a function $f(\cdot)$ to optimize with respect to $\boldsymbol{x} \in \mathbb{R}^n$. Let $f_k = f(\boldsymbol{x}_k)$ with $\boldsymbol{x}_k$ be the current solution at iteration k and $\nabla \boldsymbol{f}_k$ its gradient. In order to iteratively optimize f, QN methods consider the quadratic model:

$$m_k(\boldsymbol{p}) = f_k + \nabla \boldsymbol{f}_k^T \boldsymbol{p} + \frac{1}{2}\boldsymbol{p}^T \boldsymbol{B}_k \boldsymbol{p} \tag{7}$$

with $\boldsymbol{B}_k$ an estimate of the Hessian at the current solution $\boldsymbol{x}_k$. QN methods move to the next iteration by taking a step in the direction of the quadratic model's minimizer, $\boldsymbol{p}^* = -\boldsymbol{B}_k^{-1}\nabla \boldsymbol{f}_k$. Next, the Hessian estimate is updated and the process is repeated until convergence.

QN methods differ in how this Hessian estimate is computed. Two well-known QN methods are BFGS and SR1, which, respectively, apply a rank-two and rank-one update to the Hessian estimate at each iteration. For BFGS, the Hessian estimate at iteration k is computed as follows [4]:

$$\boldsymbol{B}_k = \boldsymbol{B}_0 - \begin{bmatrix}\boldsymbol{B}_0\boldsymbol{S}_k & \boldsymbol{Y}_k\end{bmatrix}\begin{bmatrix}\boldsymbol{S}_k^T\boldsymbol{B}_0\boldsymbol{S}_k & \boldsymbol{L}_k \\ \boldsymbol{L}_k^T & -\boldsymbol{D}_k\end{bmatrix}^{-1}\begin{bmatrix}\boldsymbol{S}_k^T\boldsymbol{B}_0 \\ \boldsymbol{Y}_k^T\end{bmatrix} \tag{8}$$

with $\boldsymbol{S}_k = [\boldsymbol{s}_0, \boldsymbol{s}_1 \ldots, \boldsymbol{s}_{k-1}]$ and $\boldsymbol{Y}_k = [\boldsymbol{y}_0, \boldsymbol{y}_1 \ldots, \boldsymbol{y}_{k-1}]$ where $\boldsymbol{s}_i = \boldsymbol{x}_{i+1} - \boldsymbol{x}_i$ and $\boldsymbol{y}_i = \nabla \boldsymbol{f}_{i+1} - \nabla \boldsymbol{f}_i$. Moreover, $\boldsymbol{D}_k$ is a diagonal matrix with $(D_k)_{i,i} = \boldsymbol{s}_i^T\boldsymbol{y}_i$ and $\boldsymbol{L}_k$ is a lower triangular matrix with values $(L_k)_{i,j} = \boldsymbol{s}_i^T\boldsymbol{y}_j$ only when $i > j$, otherwise $(L_k)_{i,j} = 0$. Finally, $\boldsymbol{B}_0$ is an initial estimate for the Hessian, commonly $\boldsymbol{B}_0 = \gamma \boldsymbol{I}$ with $\boldsymbol{I}$ the $n \times n$ identity matrix.

For SR-1, the Hessian estimate at iteration k is computed as follows [4]:

$$\boldsymbol{B}_k = \boldsymbol{B}_0 + (\boldsymbol{Y}_k - \boldsymbol{B}_0\boldsymbol{S}_k)\big(\boldsymbol{D}_k + \boldsymbol{L}_k + \boldsymbol{L}_k^T - \boldsymbol{S}_k^T\boldsymbol{B}_0\boldsymbol{S}_k\big)^{-1} \times (\boldsymbol{Y}_k - \boldsymbol{B}_0\boldsymbol{S}_k)^T \tag{9}$$

While it is not guaranteed that the Hessian estimates produced by SR1 are positive definite, SR1 has proven to yield better Hessian estimates than BFGS [4].

Furthermore, to limit the memory requirements of the QN methods, only the M most recent $\boldsymbol{s}$ and $\boldsymbol{y}$ vectors are stored in $\boldsymbol{S}_k$ and $\boldsymbol{Y}_k$, respectively.

2) SAMPLED QUASI-NEWTON METHODS

To further improve QN methods, [5] proposes Sampled Quasi-Newton methods. While in QN methods $\boldsymbol{S}$ and $\boldsymbol{Y}$ depend on $(\boldsymbol{s}, \boldsymbol{y})$ pairs from the current and previous $M-1$ iterations, SQN methods "forget the past" and, at each iteration, build the $\boldsymbol{S}$ and $\boldsymbol{Y}$ matrices from scratch [5]. This has some advantages. **First**, as illustrated in [5], this yields better Hessian approximations. **Second**, since $\boldsymbol{S}$ and $\boldsymbol{Y}$ are computed from scratch at each iteration, they immediately contain M components (i.e. M columns), unlike in traditional QN methods where this is only the case after M iterations. Once $\boldsymbol{S}$ and $\boldsymbol{Y}$ are sampled, SQN methods proceed in the same way as their QN counterparts.

To obtain $\boldsymbol{S}_k$ and $\boldsymbol{Y}_k$ at iteration k from scratch, SQN methods sample points $\tilde{\boldsymbol{x}}_i \in \mathbb{R}^n$ with $i = 1, .., M$ in the neighborhood of $\boldsymbol{x}_k$. Then, these methods compute $\boldsymbol{s}_i = \boldsymbol{x}_k - \tilde{\boldsymbol{x}}_i$ and $\boldsymbol{y}_i = \nabla \boldsymbol{f}(\boldsymbol{x}_k) - \nabla \boldsymbol{f}(\tilde{\boldsymbol{x}}_i)$ and, similar to QN methods, store $\boldsymbol{s}_i$ and $\boldsymbol{y}_i$ in, respectively, $\boldsymbol{S}_k = [\boldsymbol{s}_1, \boldsymbol{s}_2, \ldots, \boldsymbol{s}_M]$ and $\boldsymbol{Y}_k = [\boldsymbol{y}_1, \boldsymbol{y}_2, \ldots, \boldsymbol{y}_M]$.

The process of obtaining $\boldsymbol{S}$ and $\boldsymbol{Y}$ by sampling is summarized in Algorithm 1. $\tilde{\boldsymbol{x}}_i$ is sampled from $N(\boldsymbol{x}_k, \boldsymbol{\Sigma})$ with $\boldsymbol{\Sigma}$ a diagonal matrix. Moreover, note that $\boldsymbol{y}$ is computed using $\nabla^2 \boldsymbol{f}(\boldsymbol{x}_k)\boldsymbol{s}$ instead of $\boldsymbol{y} = \nabla \boldsymbol{f}_k - \nabla \boldsymbol{f}(\tilde{\boldsymbol{x}}_i)$. The advantage of the former case is that it is invariant to the scale of $\boldsymbol{\Sigma}$, whereas in the latter case, the scale of $\boldsymbol{\Sigma}$ must be carefully chosen.

Algorithm 1 Computation of $\boldsymbol{S}$ and $\boldsymbol{Y}$ in SQN Methods [5]

```
function Compute_SY(x_k, M, Σ)
    Initialize S, Y ← [ ], [ ]
    for i = 1, ..., M do
        Sample x̃_i ~ N(x_k, Σ)
        s ← x_k − x̃_i
        y ← ∇²f(x_k)s        ▷ or: y ← ∇f(x_k) − ∇f(x̃_i)
        S, Y ← [S s], [Y y]
    end for
    return S, Y
end function
```

To assure that the Hessian approximations for BFGS are positive definite, an $(\boldsymbol{s}, \boldsymbol{y})$ pair may only be added to $\boldsymbol{S}$ and $\boldsymbol{Y}$ if $\boldsymbol{s}^T\boldsymbol{y} > \kappa\|\boldsymbol{s}\|^2$ [5]. On the other hand, in order for Equation 9 to be well defined for SR1, $|\boldsymbol{s}^T(\boldsymbol{y} - \boldsymbol{B}\boldsymbol{s})| \geq \kappa||\boldsymbol{s}||^2$ must hold

for all $(\boldsymbol{s}, \boldsymbol{y})$ pairs [5]. Therefore, only $(\boldsymbol{s}, \boldsymbol{y})$ pairs that satisfy the required condition(s) are added to $\boldsymbol{S}$ and $\boldsymbol{Y}$. However, [5] shows that, if κ is sufficiently small (e.g. $\kappa = 1e\text{-}8$), the condition is rarely violated.

E. CSQN

We combine EWC and SQN to develop our proposed continual learning method: Continual Learning with Sampled Quasi-Newton (CSQN).

1) ADAPTING THE MODEL TO A SECOND TASK

Given Equation 3, we consider SQN methods to find an estimate of the Hessian of $\mathcal{L}_1^{ce}(\boldsymbol{\theta}^{(1)})$, which, unlike in EWC, does not assume the model parameters to be independent.

SQN methods obtain $\boldsymbol{S}$ and $\boldsymbol{Y}$ from scratch at each iteration. This enables us to approximate the Hessian without needing to optimize the ANN using an SQN method. This is a third advantage of using SQN over QN, in addition to the two given in Section III-D2. Given $\boldsymbol{\theta}^{(1)}$, we thus use Algorithm 1 to sample $\boldsymbol{S}^{(1)}$ and $\boldsymbol{Y}^{(1)}$. Next, Equation 8 or 9 returns an estimate of the Hessian, $\boldsymbol{B}^{(1)}$, which is then used in Equation 3, resulting in the following loss function to adapt the model to the second task:

$$\mathcal{L}_2(\boldsymbol{\theta}) = \mathcal{L}_2^{ce}(\boldsymbol{\theta}) + \frac{\lambda}{2}(\boldsymbol{\theta} - \boldsymbol{\theta}^{(1)})^T \boldsymbol{B}^{(1)} (\boldsymbol{\theta} - \boldsymbol{\theta}^{(1)}) \tag{10}$$

Dropping superscripts for convenience, note that $\boldsymbol{B}$ is computed from $\boldsymbol{S}$, $\boldsymbol{Y}$ and $\boldsymbol{B}_0$. It thus suffices to store these three objects, combined $2M + 1$ vectors of size N.

a: REDUCING MEMORY REQUIREMENTS FOR SR1

As SR1 with M $(\boldsymbol{s}, \boldsymbol{y})$ pairs results in a rank-M (without the initial Hessian estimate $\boldsymbol{B}_0$) approximation of the Hessian, it should suffice to store $M + 1$ instead of $2M + 1$ vectors. To this end, we compute $\boldsymbol{X} \in \mathbb{R}^{N \times M}$ and $\boldsymbol{A} \in \mathbb{R}^{N \times N}$ such that we can write Equation 9 as $\boldsymbol{B} = \boldsymbol{B}_0 + \boldsymbol{X}\boldsymbol{A}^{-1}\boldsymbol{X}^T$. Next, we apply the Cholesky factorization to $\boldsymbol{A}^{-1}$, such that $\boldsymbol{A}^{-1} = \boldsymbol{L}\boldsymbol{L}^T$, which allows us to write $\boldsymbol{B} = \boldsymbol{B}_0 + \boldsymbol{Z}\boldsymbol{Z}^T$ with $\boldsymbol{Z} = \boldsymbol{X}\boldsymbol{L}$. However, the Cholesky factorization requires $\boldsymbol{A}$ to be positive definite. If $\boldsymbol{A}$ is not positive definite, which may be problematic because in that case the regularization loss of Equation 10 might be negative, we compute the Eigenvalue Decomposition (EVD) of $\boldsymbol{A}^{-1}$: $\boldsymbol{A}^{-1} = \boldsymbol{V}\boldsymbol{\Gamma}\boldsymbol{V}^T$. Next, we obtain $\boldsymbol{\Gamma}'$ by setting the negative eigenvalues in $\boldsymbol{\Gamma}$ to zero, after which we compute the QR factorization of $(\boldsymbol{V}\sqrt{\boldsymbol{\Gamma}'})^T$ such that $\boldsymbol{A}^{-1} \approx \boldsymbol{V}\boldsymbol{\Gamma}'\boldsymbol{V}^T = (\boldsymbol{Q}\boldsymbol{R})^T\boldsymbol{Q}\boldsymbol{R} = \boldsymbol{R}^T\boldsymbol{R} = \boldsymbol{L}\boldsymbol{L}^T$ with $\boldsymbol{L} = \boldsymbol{R}^T$. Consequently, we find $\boldsymbol{Z} = \boldsymbol{X}\boldsymbol{L}$ and it suffices to store $\boldsymbol{Z} \in \mathbb{R}^{N \times M}$ and $\boldsymbol{B}_0 \in \mathbb{R}^N$. Moreover, it is assured that the regularization loss for CSQN with SR1 will be positive. Algorithm 2 summarizes the procedure for obtaining $\boldsymbol{Z}$ from $\boldsymbol{S}$ and $\boldsymbol{Y}$.

Algorithm 2 Computation of $\boldsymbol{Z}$ From $\boldsymbol{S}$ and $\boldsymbol{Y}$

function Compute_Z_from_SY($\boldsymbol{S}$, $\boldsymbol{Y}$)
  Compute $\boldsymbol{X}, \boldsymbol{A}$ from Eq. 9 s.t. $\boldsymbol{B} = \boldsymbol{B}_0 + \boldsymbol{X}\boldsymbol{A}^{-1}\boldsymbol{X}^T$
  Compute $\boldsymbol{A}^{-1} = \boldsymbol{V}\boldsymbol{\Gamma}\boldsymbol{V}^T$ ▷ EVD
  if $\boldsymbol{A}^{-1}$ is positive definite **then**
    Compute $\boldsymbol{A}^{-1} = \boldsymbol{L}\boldsymbol{L}^T$ ▷ Cholesky factorization
    return $\boldsymbol{Z} = \boldsymbol{X}\boldsymbol{L}$
  else
    Set $\boldsymbol{\Gamma}' = \max(\boldsymbol{\Gamma}, \boldsymbol{0})$ ▷ Element-wise max(·)
    $(\boldsymbol{V}\sqrt{\boldsymbol{\Gamma}'})^T = \boldsymbol{Q}\boldsymbol{R}$ ▷ QR factorization
    return $\boldsymbol{Z} = \boldsymbol{X}\boldsymbol{R}^T$
  end if
end function

b: FURTHER IMPROVING CSQN WITH DIAGONAL OF FIM

To further improve CSQN, we use $\boldsymbol{\Omega}^{(1)}$, the diagonal of the FIM and the Hessian approximation of EWC, in two ways. **First**, as covariance matrix $\boldsymbol{\Sigma}$ to compute $\boldsymbol{S}$ and $\boldsymbol{Y}$ (see Algorithm 1), we set $\boldsymbol{\Sigma} = (\boldsymbol{\Omega}^{(1)} + \epsilon \max(\boldsymbol{\Omega}^{(1)})\boldsymbol{I})^{-1}$ with $\max(\boldsymbol{\Omega}^{(1)})$ the maximum of $\boldsymbol{\Omega}^{(1)}$ and $\boldsymbol{I}$ the $N \times N$ identity matrix. The idea is that knowing the Hessian leads to better sampling, and the diagonal of the FIM is an initial estimate of the Hessian. This has proven to work well in our experiments. However, note that this introduces a new hyperparameter ϵ to avoid numerical problems when some elements of $\boldsymbol{\Omega}^{(1)}$ are (close to) zero. **Second**, we take $\boldsymbol{\Omega}^{(1)}$ as our initial estimate of the Hessian, i.e. $\boldsymbol{B}_0^{(1)} = \boldsymbol{\Omega}^{(1)}$. This means that the regularization loss of CSQN can be split into two terms: one term is related to $\boldsymbol{B}_0$ and therefore equal to EWC's regularization loss, while the second term is related to $\boldsymbol{S}$ and $\boldsymbol{Y}$ and the corresponding rank-$2M$ or rank-M Hessian approximations. Consequently, CSQN directly extends EWC. (Note that Algorithm 1 requires computing the gradient of $\mathcal{L}_1^{ce}(\boldsymbol{\theta}^{(1)})$, which is also required to compute $\boldsymbol{\Omega}^{(1)}$. Therefore, for CSQN, computing $\boldsymbol{\Omega}^{(1)}$ does not entail an extra forward or backward pass over the task's data.)

2) EXTENDING CSQN TO MULTIPLE TASKS

To adapt a model trained on t tasks to a $(t + 1)$th task, we follow the reasoning of [48] and consider the following loss:

$$\mathcal{L}_{t+1}(\boldsymbol{\theta}) = \mathcal{L}_{t+1}^{ce}(\boldsymbol{\theta}) + \frac{\lambda}{2}\left(\boldsymbol{\theta} - \boldsymbol{\theta}^{(t)}\right)^T \left(\sum_{j=1}^{t} \boldsymbol{B}^{(j)}\right) \left(\boldsymbol{\theta} - \boldsymbol{\theta}^{(t)}\right) \tag{11}$$

3) LIMITING MEMORY REQUIREMENTS

A disadvantage of our method, as currently proposed, is that the memory requirements increase linearly with the number of tasks. To overcome this, a straightforward approach is to use Algorithm 2 to compute $\boldsymbol{Z}^{(i)}$ for each task i. These matrices are then concatenated into one large matrix $\boldsymbol{Z}^{(\leq t)}$, after which Singular Value Decomposition (SVD) is applied to reduce the number of columns to M (for SR1) or $2M$ (for BFGS). This is done after each task, and thus the number of columns of $\boldsymbol{Z}^{(\leq t)}$ is constant as t increases. To compensate for the reduction of regularization loss, as we reduce the number

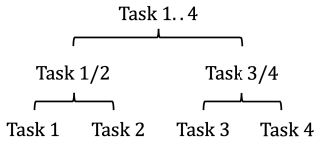


FIGURE 1. Illustration of the binary tree of tasks for the BTREE method with $T = 4$. After learning task 2, the Z matrices of tasks 1 and 2 are concatenated and reduced with SVD to form a single Z matrix of size $(2)M$. Following task 3, no reduction is applied. After task 4 is learned, the Z matrices of tasks 3 and 4 are concatenated and reduced. In a second step, the Z matrix from tasks 1 and 2, and the Z matrix from tasks 3 and 4 are further reduced into a single Z matrix of size $(2)M$.

of columns of $\boldsymbol{Z}^{(\leq t)}$ from M_b to M_a, we multiply the reduced $\tilde{\boldsymbol{Z}}^{(\leq t)}$ by $\sqrt{M_b}/\sqrt{M_a}$. We call this strategy **CT**.

One problem of this strategy might be that at task $t+1$, when $\tilde{\boldsymbol{Z}}^{(\leq t)}$ and $\boldsymbol{Z}^{(t+1)}$ are concatenated and reduced with SVD, $\tilde{\boldsymbol{Z}}^{(\leq t)}$ represents t times more tasks than $\boldsymbol{Z}^{(t+1)}$, yet both contain the same number of columns. Consequently, one can expect $\tilde{\boldsymbol{Z}}^{(\leq t+1)}$ to contain less and less information about older tasks as t increases. To overcome this, we group the tasks into a binary tree. All tasks start at the bottom of the tree and are grouped two-by-two. Once a task and its neighbour in the tree have both been learned by the model, their $\boldsymbol{Z}$ matrices are concatenated and reduced with SVD into a $\boldsymbol{Z}$ matrix of size M for SR1 and $2M$ for BFGS. The resulting matrix moves up the tree and the process is repeated. Therefore, unlike in the CT strategy, any two $\boldsymbol{Z}$ matrices that are concatenated and reduced represent an equal number of tasks. With this strategy, referred to as **BTREE**, the number of components to store is not constant, but only increases with $\log_2(T)$. This method is illustrated for $T = 4$ in Fig. 1.

Finally, we also consider using CSQN for the most recent task and EWC for older tasks. Thus, we only store the $\boldsymbol{Z}$ matrix (for SR1), or $\boldsymbol{X}$ and $\boldsymbol{A}$ matrices (for BFGS) of the most recent task; for older tasks, we discard these matrices and only regularize through $\boldsymbol{B}_0$, the initial Hessian, which is the diagonal of the FIM that is used in EWC. For example, in Equation 11, this means that $\boldsymbol{B}^{(t)}$ is computed using Algorithm 1 and Equations 8 or 9, while $\boldsymbol{B}^{(j)} = \boldsymbol{\Omega}^{(j)}$ for $j < t$. This means that CSQN 'protects' the most recent task, while EWC 'protects' older tasks. We refer to this strategy as **MRT** (Most Recent Task). While one might expect that this would mean that we need two regularization weights λ, i.e. one for the most recent task (CSQN) and one for the older tasks (EWC), we found that this is not the case.

An overview of the three proposed strategies to limit the memory requirements of CSQN is shown in Table 1.

4) OVERVIEW

Algorithm 3 summarizes our method, CSQN. Note that *reduce*$(\cdot)$ could be any user-defined function to reduce $\boldsymbol{Z}^{(\leq t)}$, including those proposed in Section III-E3. If it is defined, or if it is not (because reducing the number of components is not desired) but the considered QN method is SR1, then Algorithm 2 is used to compute $\boldsymbol{Z}^{(t)}$ (Line 13). In the former case, by applying *reduce*$(\cdot)$ to $\boldsymbol{Z}^{(t)}$ and $\tilde{\boldsymbol{Z}}^{(<t)}$, $\tilde{\boldsymbol{Z}}^{(\leq t)}$ is obtained (Line 16). In the latter case, $\tilde{\boldsymbol{Z}}^{(\leq t)}$ is simply obtained by concatenating $\boldsymbol{Z}^{(1)}, \ldots, \boldsymbol{Z}^{(t-1)}$. In either case, $\tilde{\boldsymbol{Z}}^{(\leq t)}$ is then used in Line 6 in the next iteration (of the for loop) to compute the regularization loss to learn task $t+1$. Finally, if it is not desired to reduce the number of components and the considered method is BFGS, then it is not required to compute $\boldsymbol{Z}^{(t)}$, and it suffices to store $\boldsymbol{S}^{(t)}$ and $\boldsymbol{Y}^{(t)}$. To learn task $t+1$ during the next iteration, then, $\boldsymbol{S}^{(1)}, \ldots, \boldsymbol{S}^{(t)}$, $\boldsymbol{Y}^{(1)}, \ldots, \boldsymbol{Y}^{(t)}$ and Equation 8 are used.

TABLE 1. Overview of methods for limiting CSQN's memory requirements. Last column describes how the memory requirements increase with the number of tasks T (Linear: linearly with T; Log: logarithmically with T; Constant: constant, no increase).

Name	Method	Memory
None	Store and maintain all $(2)M+1$ vectors per task	Linear
CT	Apply SVD to reduce $[\tilde{\boldsymbol{Z}}^{(<t)}\ \boldsymbol{Z}^{(t)}]$ to $(2)M+1$ columns after each task	Constant
BTREE	Order tasks in binary-tree and apply CT to pairs of tasks, as shown in Fig. 1.	Log
MRT	Apply CSQN only to the most recent task, using EWC for older tasks	Constant

Algorithm 3 Continual Learning With SQN

1: Initialize $\boldsymbol{\theta}$, select M, ϵ, *method* (BFGS or SR1), define *reduce*$(\cdot)$ to reduce $\boldsymbol{Z}$ (if desired), choose λ.
2: Set $\boldsymbol{B}_0 \leftarrow \boldsymbol{0}$
3: **for** $t = 1, \ldots, T$ **do**
4: *# Use* $\boldsymbol{B}_0$, *and* $\tilde{\boldsymbol{Z}}^{(\leq t)}$ *with Eq. 9 for SR1*
5: *# or* $\{\boldsymbol{S}^{(j)}\}_{j=1}^{(t-1)}$ *and* $\{\boldsymbol{Y}^{(j)}\}_{j=1}^{(t-1)}$ *with Eq. 8 for BFGS*
6: $\boldsymbol{\theta}^{(t)} \leftarrow$ train $f_{\boldsymbol{\theta}}$ with loss $\mathcal{L}_t(\boldsymbol{\theta})$ from Eq. 11 on $\mathcal{D}_t$
7: Compute $\nabla \mathcal{L}^{ce}{}_t(\boldsymbol{\theta}^{(t)})$ and $\boldsymbol{\Omega}^{(t)}$
8: $\boldsymbol{\Sigma} \leftarrow (\boldsymbol{\Omega}^{(t)} + \epsilon \max(\boldsymbol{\Omega}^{(t)})\boldsymbol{I})^{-1}$
9: $\boldsymbol{S},\ \boldsymbol{Y} \leftarrow$ Compute_SY($\boldsymbol{\theta}^{(t)},\ M,\ \boldsymbol{\Sigma}$)
10: $\boldsymbol{B}_0 \leftarrow \boldsymbol{B}_0 + \boldsymbol{\Omega}^{(t)}$
11: *# For SR1, or to reduce,* $\boldsymbol{Z}$ *must be computed*
12: **if** *method* == SR1 **or** *reduce*$(\cdot)$ is defined **then**
13: $\boldsymbol{Z}^{(t)} \leftarrow$ Compute_Z_from_SY($\boldsymbol{S},\ \boldsymbol{Y}$)
14: $\tilde{\boldsymbol{Z}}^{(\leq t)} \leftarrow [\tilde{\boldsymbol{Z}}^{(<t)}\ \ \boldsymbol{Z}^{(t)}]$
15: **if** *reduce*$(\cdot)$ is defined **then**
16: Compute $\tilde{\boldsymbol{Z}}^{(\leq t)} \leftarrow$ *reduce*$(\tilde{\boldsymbol{Z}}^{(\leq t)})$
17: **end if**
18: Store $\boldsymbol{B}_0, \tilde{\boldsymbol{Z}}^{(\leq t)}$
19: **else**
20: Store $\boldsymbol{B}_0, \boldsymbol{S}^{(t)}, \boldsymbol{Y}^{(t)}$
21: **end if**
22: **end for**
23: **result** model $f_{\boldsymbol{\theta}^{(T)}}$

IV. EXPERIMENTS

We evaluate CSQN on several continual learning benchmarks and compare its performance against strong baselines.

Datasets. We consider four benchmarks: Rotated MNIST [34], Split CIFAR-10/100 [9], Split TinyImageNet [49], and Vision Datasets [17]. In Rotated MNIST, following [27], MNIST [50] images for task t are rotated by an angle of $\gamma(t-1)$. The MNIST training set is split into 55,000 training and 5,000 validation images. For Split CIFAR-10/100, the CIFAR-100 data [51] is divided into T subsets with $100/T$ non-overlapping classes, with CIFAR-10 as the first task, resulting in $T+1$ tasks. We split the CIFAR-10 and CIFAR-100 training sets into 45,000 training and 5,000 validation images. In Split TinyImageNet, the 200 classes are split into T tasks with non-overlapping classes, with each task further divided into 90% training and 10% validation data. Finally, in the Vision Datasets benchmark, tasks are composed of five datasets: MNIST, CIFAR-10, SVHN [52], FashionMNIST [53], and notMNIST [54], with training data split into 95% training and 5% validation. In the Rotated MNIST experiments, the entire model is shared, whereas in Split CIFAR-10/100, Split TinyImageNet, and Vision Datasets experiments, the classification layer is task-specific, meaning each task learns its own classification layer.

Baselines. We compare CSQN to the following methods:

(a) EWC, as explained in Section III-B;
(b) Memory-Aware Synapses (MAS) [10]: similar to EWC but computes $\Omega^{(t)}$ differently, without relying on class ground truth labels;
(c) Kronecker-Factored approximated Laplace (KF) [17]: uses Kronecker-factored approximations of the FIM to estimate the posterior with Laplace approximation;
(d) Learning without Forgetting (LWF) [21]: uses knowledge distillation between $f_{\boldsymbol{\theta}^{(t)}}$ and $f_{\boldsymbol{\theta}}$ (the current model) on data from task $t+1$ when learning task $t+1$;
(e) Orthogonal Gradient Descent (OGD) [27]: updates the gradient in SGD such that it is orthogonal to gradients of the previous tasks (we consider the OGD-GTL variant) – to this end, it stores K gradients for each task;
(f) Gradient Projection Memory (GPM) [28]: splits the gradient space into a subspace with high interference and one with low interference with previous tasks. Before performing an optimizer step with SGD, GPM then makes sure that the gradient update lies in the latter subspace;
(g) Experience Replay (ER) [33]: a rehearsal-based method which, at each iteration, samples a mini-batch from a (small) memory of previous samples to rehearse previous tasks;
(h) Dark Experience Replay++ (DER++) [25]: a strong rehearsal-based continual learning method that combines experience replay with knowledge distillation [20] to stabilize learning across tasks;
(i) Average-Gradient Episodic Memory (A-GEM) [36]: a rehearsal-based method that, at each iteration, samples a mini-batch from memory to ensure that the gradients of the new task and old tasks do not interfere. If interference is detected, the method updates the new task's gradient to make it orthogonal to those of the old tasks;
(j) Fine-Tuning: naïvely adapts the model to new tasks without using any CL algorithm - considered the worst-case scenario.

We first compare CSQN to EWC, MAS, and KF—similar regularization-based methods—on the Rotated MNIST and Split CIFAR-10/100 experiments. For the Split TinyImageNet and Vision Datasets experiments, we include all baselines to evaluate CSQN against a broader set of strong methods. Our baseline selection prioritizes methods that are broadly applicable across different architectures and CL settings. In particular, we focus on methods that are effective in scenarios where the primary challenge is maintaining a stable feature representation across tasks. This aligns with our experimental setting, where new tasks involve either the same classes as previous tasks (commonly referred to as domain-incremental learning) or new classes (called class-incremental learning), but with a newly trained classification layer.

Evaluation Metrics. To evaluate the performance of the methods, we consider the average accuracy (ACC) and backward transfer (BWT), as in [34]. The former is simply the average of the accuracies of all tasks after the last task has been learned. The latter is defined as follows, assuming that R_{ij} is the accuracy on task j after learning task i and that T tasks are learned:

$$\text{BWT} = \frac{1}{T-1}\sum_{i=1}^{T-1}(R_{Ti} - R_{ii}) \tag{12}$$

Consequently, negative BWT indicates forgetting.

Models and training. For the MNIST benchmarks, we consider a Multi-Layer Perceptron (MLP) with two hidden layers of 256 neurons and ReLU activations and a dropout [55] of 25%. We train the model for 10 epochs for each task with Adam [56] ($\beta_1 = 0.9$, $\beta_2 = 0.999$) with a learning rate of 0.001. For the Split CIFAR-10/100 and Split TinyImageNet experiments, we consider ResNet-18 [57]. For Split CIFAR-10/100, the first task (on the CIFAR-10 dataset) is trained for 15 epochs and subsequent (CIFAR-100 tasks) are trained for 10 epochs, while for the Split TinyImageNet experiments, the number of epochs is set to 5. The optimizer is the same as for the MNIST experiments, except for OGD, GPM and A-GEM, for which, instead of Adam, we use SGD with the same learning rate and momentum of 0.9. Finally, for the Vision Datasets experiments, we use a similar LeNet [58] as in [17], with two convolutional layers with convolutions of 5×5 and 20 and 50 channels, respectively, followed by a fully-connected layer with 500 neurons. However, we add another fully-connected layer of 84 neurons, to reduce the number of neurons in the classification layer, which is task-specific and therefore not subject to regularization. Each of the five tasks is trained for 50 epochs using the same optimizers as for the MNIST experiments.

TABLE 2. Average accuracy (ACC) and backward transfer (BWT) in % on, respectively, the Rotated MNIST, Split CIFAR-10/100, Split TinyImageNet and Vision Datasets experiments. Best average performance (ACC) per experiment is in bold.

Model	*MNIST*		*CIFAR*		*ImageNet*		*Vision*	
	ACC	BWT	ACC	BWT	ACC	BWT	ACC	BWT
Fine-Tuning	58.66	-41.15	50.52	-22.87	23.89	-28.37	33.00	-73.00
EWC	79.29	-17.12	61.42	-2.58	44.55	+0.02	70.62	-9.74
MAS	82.06	-12.48	60.84	-2.42	42.55	+0.02	73.78	-8.96
KF	**94.66**	-1.92	61.05	+0.26	45.01	+0.72	77.55	-1.67
LWF	-	-	-	-	11.03	-43.99	75.71	-9.45
OGD	-	-	-	-	44.99	-3.36	47.41	-50.66
GPM	-	-	-	-	40.31	+0.35	77.03	-4.07
ER	-	-	-	-	37.84	-8.02	77.51	-11.96
A-GEM	-	-	-	-	30.19	-12.04	67.55	-24.65
DER++	-	-	-	-	40.25	-7.33	77.09	-11.01
CSQN-S (10)	85.49	-8.21	65.82	-2.54	46.07	+0.33	77.95	-6.69
CSQN-S (20)	86.22	-7.46	**66.39**	-1.06	45.47	+0.35	**79.23**	-4.77
CSQN-B (10)	85.37	-8.45	62.69	-0.28	**46.49**	+0.74	78.04	-2.30
CSQN-B (20)	85.96	-7.96	64.32	-5.06	46.02	+0.43	77.98	-6.77

TABLE 3. Average accuracy (ACC), backward transfer (BWT) and forward transfer (FWT) in % on, respectively, the Rotated MNIST, Split CIFAR-10/100, Split TinyImageNet and Vision Datasets experiments with different reduction strategies applied to CSQN-S (20). 'None' is CSQN-S (20) from Table 2. Last column describes how the memory requirements increase with the number of tasks T (Linear: linearly with T; Log: logarithmically with T; Constant: constant, no increase).

Method	*MNIST*		*CIFAR*		*ImageNet*		*Vision*		Memory
	ACC	BWT	ACC	BWT	ACC	BWT	ACC	BWT	
None	86.22	-7.46	66.39	-1.06	45.47	+0.35	79.23	-4.77	Linear
CT	82.44	-6.91	64.06	-1.79	42.41	-2.21	74.95	-6.65	Constant
BTREE	85.48	-8.36	65.86	-1.48	45.53	+0.26	75.97	-3.37	Log
MRT	84.76	-9.57	63.73	-0.63	47.94	+0.77	75.36	-5.88	Constant

Implementation. For our method, CSQN, we consider $\epsilon = 1e\text{-}4$ and $\kappa = 1e\text{-}12$ in all experiments. We consider CSQN with both BFGS (referred to as CSQN-B) and SR1 (CSQN-S) with $M = 10$ and $M = 20$ (reported between brackets). For the rehearsal-based baselines – ER, DER++ and A-GEM – we randomly sample a memory set from the training data after training each task and add it to the 'global' memory set, whose size increases with the number of tasks. For the hyperparameter selection with respect to λ (α and β for DER++), the regularization weight, for both our method and the baselines, as well as for the other hyperparameters for the baselines, we refer to Appendix A. All experiments are done in PyTorch [59].[1] All reported results are averages of five runs. The standard deviation of these five runs is, for all experiments, reported in Appendix B.

[1]For our code, refer to *github.com/StevenVdEeckt/csqn*

A. ROTATED MNIST

For the Rotated MNIST experiments, we consider $T = 20$ and $\gamma = 5°$.

As shown in Table 2, CSQN outperforms EWC and MAS in terms of ACC after 20 tasks, for all four settings. The differences between these settings are relatively small, but increasing the number of components from $M = 10$ to $M = 20$ seems to slightly improve the performance of CSQN with both BFGS and SR1. Additionally, CSQN with SR1 slightly outperforms CSQN with BFGS. Specifically, CSQN-S 20 improves the performance of EWC and MAS by 8.7% and 5.1%, respectively. Figure 2 shows the average accuracy after each task, clearly illustrating that CSQN consistently performs better than EWC and MAS, particularly after 10 tasks, where the gap becomes wider.

However, while CSQN outperforms EWC, it underperforms KF, which reaches an impressive average accuracy of

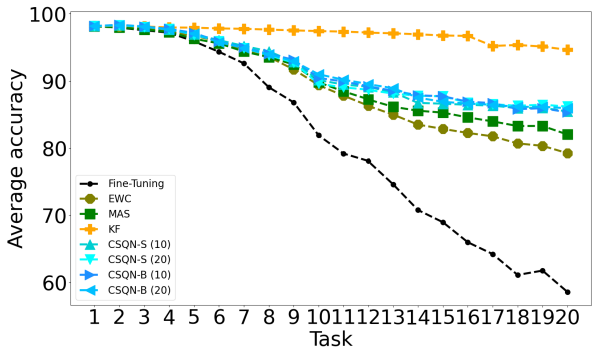


FIGURE 2. Average accuracy in % after each task for Rotated MNIST.

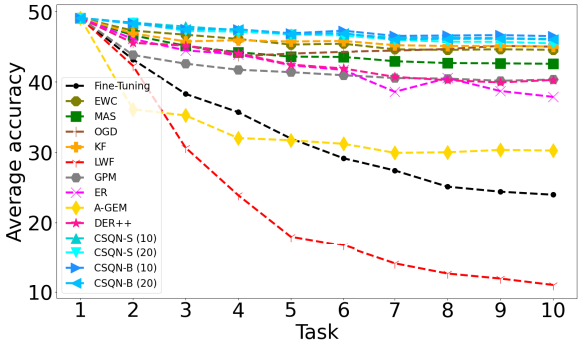


FIGURE 4. Average accuracy in % after each task for Split TinyImageNet.

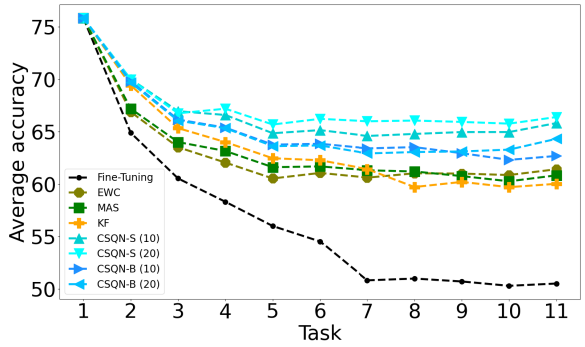


FIGURE 3. Average accuracy in % after each task for Split CIFAR-10/100 experiments.

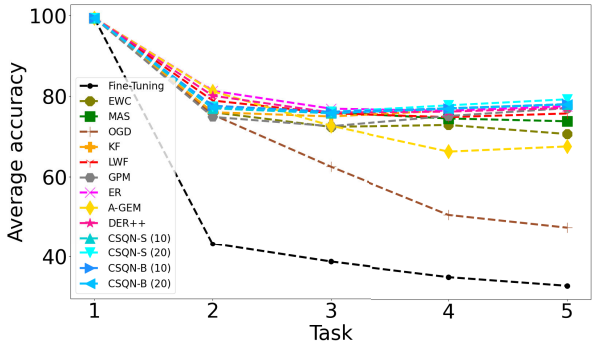


FIGURE 5. Average accuracy in % after each task for Vision Datasets experiments.

over 94%—approximately 10% higher than the best CSQN setting and 19% higher than EWC. This result is consistent with [17], which also found KF to perform considerably better than EWC in similar experiments on the MNIST dataset with a similar model. Given its excellent performance, it is no surprise that KF attains the highest backward transfer and experiences significantly less forgetting than CSQN, and especially MAS and EWC. Indeed, Figure 2 shows that KF is able to learn the tasks with almost zero forgetting.

Note that since the Rotated MNIST tasks are ordered on increasing angle γ, the results could vary if tasks were presented in a different sequence.

B. SPLIT CIFAR-10/100

We split the CIFAR-100 dataset into $T = 10$ tasks, thus obtaining a sequence of 11 tasks. Table 2 shows the final results for the Split CIFAR-10/100 experiments.

Similar conclusions can be drawn as for the Rotated MNIST experiments: CSQN substantially outperforms EWC and MAS across all CSQN settings. As observed in the Rotated MNIST experiments, CSQN with SR1 performs better than CSQN with BFGS, and having more components (larger M) also proves advantageous. However, the differences between settings are more pronounced. For instance, CSQN-S (20) improves EWC's performance by 8.1%, whereas CSQN-B (10) only achieves a 2.1% improvement. Nevertheless, all CSQN configurations (CSQN-S (20), CSQN-B (10), CSQN-S (10), and CSQN-B (20)) outperform KF, unlike in the Rotated MNIST experiments. While KF once again achieves the highest backward transfer, even a positive value (indicating improved performance on old tasks when learning new ones), it struggles to learn new tasks effectively and underperforms compared to EWC.

Figure 3, which displays the average accuracy after training each task, shows that KF outperforms EWC until the seventh task, after which it experiences a drop in performance. All CSQN settings consistently outperform EWC, MAS, and KF, with CSQN-S (20) being the best performing method after each task.

C. SPLIT TINYIMAGENET

For the Split TinyImageNet experiments, we consider $T = 10$, resulting in a sequence of ten tasks. Additionally, we introduce the baselines LWF, OGD, GPM, ER, and A-GEM. The results are presented in Table 2.

Once again, all four CSQN settings outperform EWC, MAS, and KF. As shown in Figure 4, this holds true after learning each of the ten tasks. Interestingly, compared to previous experiments, CSQN with BFGS outperforms CSQN with SR1, and more components (higher M) prove detrimental rather than beneficial to performance. The CSQN settings with $M = 10$ achieve approximately the same backward transfer as those with $M = 20$ while demonstrating higher average performance. This suggests that the higher M

settings may be over-regularizing, protecting parameters or prohibiting directions in the parameter space unnecessarily, which hinders the model from effectively learning new tasks.

Comparing CSQN to the other baselines (LWF, OGD, GPM, and the rehearsal-based methods ER, A-GEM and DER++), CSQN consistently outperforms all of them, as these baselines also underperform compared to KF. Figure 4 demonstrates this superiority after learning each task. With the exception of GPM, these baselines fail to achieve the high (or even positive) backward transfer of CSQN, KF, and even EWC and MAS, indicating that they suffer from severe forgetting compared to the latter methods.

D. VISION DATASETS

Finally, we consider the Vision Datasets experiments, with results presented in Table 2.

CSQN outperforms EWC, MAS, and KF, with CSQN-S (20) achieving the best performance among the CSQN settings, similar to the Rotated MNIST and Split TinyImageNet experiments. From the four experiments, we conclude that CSQN with SR1 generally performs slightly better than CSQN with BFGS, especially considering that in terms of memory requirements, CSQN-B (10) is equivalent to CSQN-S (20), not CSQN-B (20). CSQN-S (20) also achieves the highest backward transfer among the CSQN settings, though it remains lower than KF's. However, KF struggles to learn the tasks as effectively as CSQN. Figure 5 shows that the gap between CSQN and KF remains roughly consistent after each task, while the gap between CSQN and EWC or MAS widens as more tasks are learned.

Although LWF, GPM, ER and DER++ achieve much higher performance than EWC and MAS, they are still outperformed by CSQN. LWF, ER and DER++ are effective at learning new tasks but suffer from much higher forgetting compared to CSQN. GPM achieves a backward transfer comparable to CSQN (better than CSQN-S (10) and CSQN-B (20) but worse than CSQN-S (20)) but fails to learn new tasks as effectively as CSQN. Finally, A-GEM and OGD suffer from catastrophic forgetting and perform significantly worse than all other baselines and CSQN.

Interestingly, as shown in Figure 5, A-GEM performs well after the first two tasks, achieving higher average accuracy than CSQN—with only ER performing better. However, as more tasks are added, A-GEM's performance declines relative to that of the other methods. A similar trend is observed for ER, which performs best after the second and third tasks, but is eventually outperformed by CSQN.

E. SENSITIVITY ANALYSIS OF RANK M IN CSQN

We conduct a sensitivity analysis of the rank M in CSQN on Split CIFAR-10/100 (Figure 6) and Vision Datasets (Figure 7). Full results can be found in Appendix B. Note that the setting with $M = 0$ corresponds to EWC.

On **Split CIFAR-10/100**, even the lowest tested rank of $M = 1$ already provides a significant advantage over EWC, demonstrating that CSQN can improve upon traditional regularization-based methods even with minimal Hessian information. As M increases, performance continues to improve for SR-1, while for BFGS, noticeable gains only appear at $M = 20$. Across all values of $M > 0$, CSQN with SR-1 consistently outperforms its BFGS counterpart.

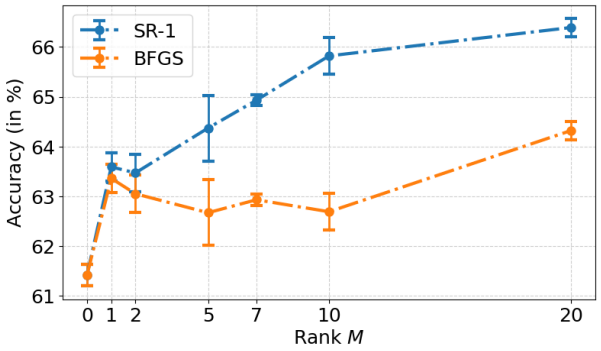


FIGURE 6. Sensitivity of CSQN with respect to rank M on Split CIFAR-10/100.

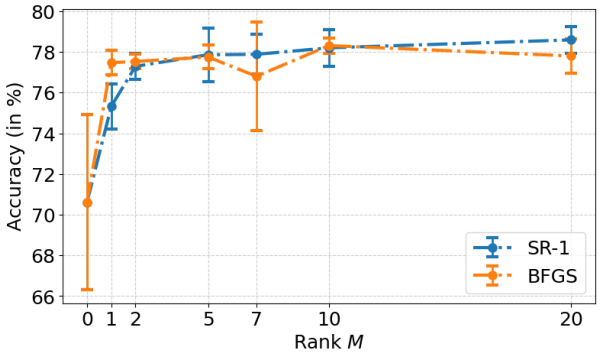


FIGURE 7. Sensitivity of CSQN with respect to rank M on Vision Datasets.

To ensure fair comparisons, the regularization weight λ from Equation 11 was selected from a predefined set of five values for each M using a validation set (see Appendix A, Table 5). However, it is possible that the actual optimal λ lies between two of these values. This could explain why, for BFGS, increasing M occasionally leads to (slight) degradation in performance. Nevertheless, since the impact of an exact λ tuning would be small, we conclude that increasing M from $M = 1$ to $M = 10$ with BFGS in this experiment does not result in meaningful gains.

Similar trends are observed on the **Vision Datasets**. With $M = 1$, CSQN already surpasses EWC, confirming that even a low-rank approximation effectively mitigates catastrophic forgetting. From $M = 2$ onward, increasing the rank M leads to limited improvements. At $M = 1$, CSQN with BFGS already outperforms all baselines except KF and ER; from $M = 5$ onward, both BFGS and SR-1 variants of CSQN outperform all baselines. Notably, the difference between BFGS and SR-1 remains small across all tested values of M. Initially, at $M = 1$ and $M = 2$, BFGS slightly outperforms SR-1, but at $M = 20$, SR-1 surpasses BFGS.

These findings indicate that even a low-rank approximation with $M = 1$ is already sufficient to provide substantial improvements over EWC. Furthermore, setting $M = 5$ suffices for CSQN to outperform all other baselines.

F. LIMITING MEMORY REQUIREMENTS

As discussed in Section III-E3, a disadvantage of CSQN is its memory requirements, which increase linearly with T, the number of tasks. To address this, we proposed three strategies to mitigate this increase, summarized in Table 1, and applied these strategies to CSQN with SR1 and $M = 20$ in all four experiments. The results are presented in Table 3.

The CT strategy, which uses SVD to ensure memory requirements remain constant, still outperforms EWC and MAS on the Rotated MNIST, Split CIFAR-10/100, and Vision Datasets experiments. Additionally, on the latter two, similar to the non-reduced CSQN, it also outperforms KF. However, on Split TinyImageNet, CT suffers from increased forgetting and underperformance even compared to EWC and MAS. On average, the performance degradation of CT compared to non-reduced CSQN is between 4% and 7% across the four experiments.

Compared to CT, the BTREE strategy – where Z matrices of the tasks are organized in a binary tree and reduced two-by-two – performs better, with less than 1% degradation on the Rotated MNIST and Split CIFAR-10/100 experiments. On Split TinyImageNet, no performance degradation is observed. Consequently, BTREE outperforms EWC and MAS on all experiments, KF on Split CIFAR-10/100 and Split TinyImageNet, and all baselines on Split TinyImageNet. Only on Vision Datasets does BTREE experience a larger performance degradation of 4% compared to non-reduced CSQN.

Finally, the MRT strategy, which applies CSQN to the last task while regularizing older tasks using EWC, also maintains constant memory requirements like CT. However, MRT exhibits less performance degradation than CT, with less than 2% on Rotated MNIST, and between 4% and 5% on Split CIFAR-10/100 and Vision Datasets. Interestingly, on Split TinyImageNet, MRT results in a 5.5% improvement, making CSQN-S (20) with MRT the best performer among all baselines and CSQN settings.

In conclusion, both BTREE and MRT offer effective ways to limit CSQN's memory requirements, resulting in either only logarithmic growth with T (the number of tasks) or constant memory usage, respectively, with minimal performance degradation. Consequently, these strategies achieve competitive results, outperforming all baselines on Split CIFAR-10/100 and Split TinyImageNet, while many of these baselines (e.g., KF, OGD, ER, A-GEM) still face linear growth in memory usage as the number of tasks increases.

G. COMPARISON BETWEEN CSQN AND KF

Both CSQN and KF are based on EWC's idea of using Laplace approximation to estimate the posterior for previous tasks. EWC uses the diagonal of the FIM in the Laplace approximation, which implicitly assumes that model parameters are independent. CSQN and KF both attempt to overcome this limitation. KF employs Kronecker-factored approximations of the FIM [18], allowing the FIM to be efficiently and compactly obtained per layer. As a result, KF's Hessian approximation is block-diagonal and accounts for interactions between parameters within the same layer. CSQN, on the other hand, uses Hessian approximations obtained through QN methods. To derive these approximations, CSQN utilizes the concept of SQN methods, which involves sampling around the current model parameters. Consequently, CSQN's Hessian approximation is no longer diagonal, allowing interactions between any two parameters to be considered. Another advantage of CSQN is its ease of implementation across different network architectures, as obtaining the Hessian approximation only requires sampling around the current model parameters and computing $M + 1$ gradients. However, a drawback of CSQN is that its Hessian estimate is a very low-rank approximation, as M is typically much smaller compared to N.

In the experiments, we find that KF outperforms CSQN on the Rotated MNIST experiments with the MLP, while CSQN surpasses KF on the Split CIFAR-10/100 and Split TinyImageNet experiments with ResNet-18, as well as the more challenging Vision Datasets experiments with LeNet. For KF, these results are consistent with [17], where the difference between KF and EWC was significantly larger on the MNIST experiments using MLP compared to, for instance, the Vision Datasets experiments with the LeNet architecture. Overall, CSQN outperforms KF in three out of the four experiments.

In terms of memory requirements, KF is slightly more memory-efficient than CSQN. For example, using ResNet-18, if we express memory requirements in terms of the number of models, KF needs to store slightly more than 8 models per task, while CSQN requires $M + 1$ models per task (e.g., $M = 10$ or $M = 20$). Using LeNet, the differences are more pronounced, with KF requiring 3.4 models per task compared to CSQN's M. For both KF and CSQN, memory requirements increase linearly with the number of tasks T. However, as shown in Section IV-F, we can mitigate this for CSQN with minimal performance degradation. When using the reduced versions of CSQN from Section III-E3 – of which the results on the experiments were discussed in Section IV-F – CSQN soon becomes more memory-efficient than KF after a small number of tasks, as memory requirements either increase logarithmically (for BTREE) or remain constant (for CT or MRT). Moreover, CSQN with reduction strategies BTREE or MRT was still able to outperform KF on the Split CIFAR-10/100 and Split TinyImageNet experiments.

V. DISCUSSION

Continual Learning with Sampled Quasi-Newton (CSQN) stands out for its simplicity in implementation and adaptability. Unlike methods such as Gradient Projection Memory (GPM) and Kronecker-factored (KF) approximations, which

TABLE 4. Average accuracy in % on, respectively the Rotated MNIST, Split CIFAR-10/100 and Vision Datasets experiments. These results are the average of five runs. The average of these five runs is reported, as well as the standard deviation.

Model	*MNIST*	*CIFAR*	*ImageNet*	*Vision*
	ACC	ACC	ACC	ACC
EWC	79.28 ± 0.46	61.42 ± 0.22	44.55 ± 0.60	70.62 ± 4.31
MAS	82.06 ± 1.10	60.84 ± 0.85	42.55 ± 0.49	73.78 ± 3.73
KF	94.65 ± 1.05	61.05 ± 0.23	45.01 ± 0.46	77.55 ± 1.09
LWF	-	-	11.03 ± 0.17	75.71 ± 0.35
OGD	-	-	44.99 ± 0.45	47.41 ± 1.77
GPM	-	-	40.31 ± 0.18	77.03 ± 0.18
ER	-	-	37.84 ± 1.34	77.51 ± 0.33
A-GEM	-	-	30.19 ± 1.19	67.55 ± 0.65
DER++	-	-	40.25 ± 0.40	77.09 ± 0.12
CSQN-S (1)	-	63.59 ± 0.28	-	75.32 ± 1.11
CSQN-S (2)	-	63.47 ± 0.38	-	77.29 ± 0.65
CSQN-S (5)	-	64.37 ± 0.66	-	77.86 ± 1.32
CSQN-S (7)	-	64.93 ± 0.11	-	77.88 ± 0.98
CSQN-S (10)	85.49 ± 1.29	65.82 ± 0.37	46.07 ± 0.25	77.95 ± 1.25
CSQN-S (20)	86.22 ± 1.31	66.39 ± 0.18	45.47 ± 0.26	79.23 ± 0.36
CSQN-B (1)	-	63.36 ± 0.43	-	77.47 ± 0.61
CSQN-B (2)	-	63.05 ± 0.51	-	77.53 ± 0.37
CSQN-B (5)	-	62.67 ± 0.16	-	77.75 ± 0.58
CSQN-B (7)	-	62.93 ± 0.13	-	76.80 ± 2.66
CSQN-B (10)	85.37 ± 1.09	62.69 ± 0.29	46.49 ± 0.30	78.04 ± 0.27
CSQN-B (20)	85.96 ± 0.75	64.32 ± 0.49	46.02 ± 0.43	77.98 ± 1.23
CSQN-S (20) CT	82.44 ± 1.08	64.06 ± 0.36	42.41 ± 1.07	74.95 ± 0.66
CSQN-S (20) BTREE	85.48 ± 0.44	65.82 ± 0.37	45.53 ± 0.20	75.97 ± 0.96
CSQN-S (20) LAST	84.76 ± 1.56	63.73 ± 0.22	47.94 ± 0.35	75.35 ± 2.89

require architecture-specific implementations and may not generalize to all neural network types, CSQN is straightforward to implement and can be easily applied to diverse architectures. Its only requirement is the ability to compute gradients, making it a versatile solution for continual learning tasks. Despite being simple to implement, CSQN effectively reduces forgetting and improves performance across tasks. In its Hessian approximation—derived through sampled gradients— CSQN accounts for parameter interactions beyond the layer level, unlike block-diagonal approximations used in KF. Performance might be further improved by increasing the number of components M, although this comes with trade-offs in computational and memory costs.

A. STRENGTHS OF CSQN

CSQN has several strengths, which can be summarized as follows:

- **Ease of Implementation:** Implementing CSQN requires only the ability to compute gradients, making it simple to integrate into existing workflows without additional complexity.
- **Adaptability Across Architectures:** CSQN can be applied to any architecture, including feedforward networks, convolutional architectures, and transformers [60]. Unlike methods such as GPM or KF, it does not rely on architecture-specific definitions, ensuring broad applicability across diverse neural network models.
- **Balance of Implementation Simplicity and Performance:** While simple to implement, CSQN achieves substantial improvements in retaining task knowledge and reducing forgetting compared to baseline methods. In three out of four benchmark experiments, it outperformed all other methods.

B. CHALLENGES OF CSQN

Despite its strengths, CSQN is not without limitations. The most significant challenge lies in its memory requirements. The need to store $M + 1$ vectors (for SR-1; $2M +$

1 for BFGS) for each task can make CSQN infeasible for large models or scenarios with limited memory resources. While our proposed memory reduction strategies, BTREE and MRT (Section 1), successfully overcome the linear increase in memory requirements of CSQN, the overall memory demands remain high. Further research is needed to develop approaches that further reduce memory usage while preserving CSQN's strong performance.

In certain scenarios, the memory requirements of CSQN can be mitigated by selectively applying it to parts of the model that are most critical for retaining knowledge. For instance, in monolingual Automatic Speech Recognition tasks, it has been identified that the encoder in encoder-decoder models is crucial for forgetting [61]. Applying CSQN exclusively to the encoder, while using more lightweight methods for other parts of the model, could achieve a balance between memory efficiency and performance.

It is also worth noting that every continual learning method comes with its own set of disadvantages. For instance, rehearsal-based methods, while effective, require storing past data. This is often not permissible where privacy concerns prohibit data retention.

C. EXTENSIONS TO OTHER LEARNING PARADIGMS

While CSQN is designed for continual learning, its Hessian-based regularization could also be beneficial in other learning paradigms. For example, in transfer learning, it could help improve generalization by preserving beneficial representations while adapting to new tasks. In semi-supervised learning, it might enhance stability when incorporating unlabeled data by constraining parameter updates. In robust learning, it could help mitigate sensitivity to distribution shifts or adversarial perturbations. Additionally, CSQN's Hessian approximation might be useful for model merging, where having an estimate of the Hessian could facilitate a smoother integration of two models. Exploring these extensions is beyond the scope of the current work and will be left for future research.

VI. CONCLUSION

In this paper, we present Continual Learning with Sampled Quasi-Newton (CSQN), a novel extension of Elastic Weight Consolidation (EWC) that addresses its limitations by leveraging Hessian approximations derived from Sampled Quasi-Newton (SQN) methods. By overcoming EWC's diagonal Hessian assumption, CSQN provides a more accurate regularization mechanism that accounts for parameter interactions across layers, resulting in significantly reduced forgetting and improved performance. CSQN extends EWC by improving its Hessian estimation, enhancing its performance on complex continual learning tasks. It is straightforward to implement, requiring only sampled points and gradient computations, and is applicable to diverse architectures without architecture-specific modifications.

TABLE 5. Regularization weights (λ) tested and selected for Rotated MNIST (rm), Split CIFAR-10/100 (sc), Split TinyImageNet (st) and Vision Dataset (vd) experiments.

Method	λ
EWC	$1e3$ (rm), $1e4$, $1e5$ (sc), $1e6$ (st), $1e7$ (vd)
MAS	$1e0$ (rm, sc), $1e1$ (vd), $1e2$, $1e3$ (st), $1e4$
KF	$1e4$, $1e5$ (rm), $1e6$ (st), $1e7$ (sc, vd), $1e8$
LWF	$1e$-2 (vd), $1e$-1 (st), $1e0$, $1e1$, $1e2$
DER++ (α, β)	[0.5, 0.5] (vd), [0.1, 1.0], [1.0, 0.1] (st), [0.2, 0.5], [0.5, 0.2]
CSQN-S (1)	$1e2$, $1e3$, $1e4$ (sc, vd), $1e5$, $1e6$
CSQN-S (2)	$1e2$, $1e3$, $1e4$ (sc), $1e5$ (vd), $1e6$
CSQN-S (5)	$1e2$, $1e3$ (sc), $1e4$ (vd), $1e5$, $1e6$
CSQN-S (7)	$1e2$, $1e3$ (sc), $1e4$ (vd), $1e5$, $1e6$
CSQN-S (10)	$1e2$, $1e3$ (sc), $1e4$ (rm, st vd), $1e5$, $1e6$
CSQN-S (20)	$1e2$, $1e3$ (sc), $1e4$ (rm, st, vd), $1e5$, $1e6$
CSQN-B (1)	$1e2$, $1e3$, $1e4$ (sc), $1e5$ (vd), $1e6$
CSQN-B (2)	$1e2$, $1e3$, $1e4$ (sc), $1e5$ (vd), $1e6$
CSQN-B (5)	$1e2$, $1e3$, $1e4$ (sc), $1e5$ (vd), $1e6$
CSQN-B (7)	$1e2$, $1e3$ (sc), $1e4$, $1e5$ (vd), $1e6$
CSQN-B (10)	$1e2$, $1e3$, $1e4$ (rm, sc, st), $1e5$ (vd), $1e6$
CSQN-B (20)	$1e2$, $1e3$ (sc), $1e4$ (rm, st, vd), $1e5$, $1e6$
CSQN-S (20) CT	$1e2$, $1e3$ (sc), $1e4$ (rm, st), $1e5$ (vd), $1e6$
CSQN-S (20) BTREE	$1e2$, $1e3$ (sc), $1e4$ (rm, st), $1e5$ (vd), $1e6$
CSQN-S (20) MRT	$1e2$, $1e3$, $1e4$ (rm, sc, st), $1e5$ (vd), $1e6$

CSQN demonstrates strong performance across four benchmarks, outperforming all tested baselines on three, reducing EWC's forgetting by an average of 50%, and improving its performance by an average of 8%. These results underscore CSQN's effectiveness as a continual learning method.

While CSQN's strengths include ease of implementation and broad applicability, its memory requirements are substantial, particularly for large models. To address this, we propose three reduction strategies—CT, BTREE, and MRT—that alleviate this burden with minimal performance trade-offs. Future work will explore reducing CSQN's memory requirements through efficient sampling strategies or hybrid approaches. By refining CSQN, we aim to make it an even more scalable and robust solution for continual learning tasks.

APPENDIX A
HYPER-PARAMETER SEARCH

Most of the baselines, as well as our method, require choosing a hyper-parameter λ to determine the weight of the regularization. Table 5 gives a summary of the values of λ tested for each of the methods. Note that DER++ instead of λ requires two hyper-parameters α and β as the regularization weight of its two losses. For each experiment, we selected the value of λ with the best performance on the validation sets of all tasks and then repeated the experiment five times with this value of λ. We indicate between brackets which value was used for the Rotated MNIST (rm), Split

CIFAR-10/100 (sc), Split TinyImageNet (st) and Vision Datasets (vd) benchmarks.

Apart from a regularization weight, LWF also requires a temperature T_{LWF}. Similar to [21], we use $T_{\text{LWF}} = 2$. OGD requires a hyper-parameter K, the number of gradients per task to store. We set $K = 20$, for OGD to have similar memory requirements as CSQN. $K = 20$ was also used in [27]. For GPM, we use the same hyper-parameters as in the original paper [28]. Finally, for ER, DER++ and A-GEM, we use a memory of N_{memory} samples (randomly sampled from the training set) per task. Consequently, after learning T tasks, the memory contains $T * N_{memory}$ samples. For both the Split TinyImageNet and Vision Datasets experiments, N_{memory} was set to 1000, or 50 samples per class per task for the former and 100 samples per class per task for the latter experiments.

APPENDIX B
ADDITIONAL RESULTS

All the reported results in Section IV were the averages of five runs. In Table 4, we report these averages along with the standard deviation of these five runs for all experiments and all methods from Section IV.

REFERENCES

[1] M. McCloskey and N. J. Cohen, "Catastrophic interference in connectionist networks: The sequential learning problem," in *Psychology of Learning and Motivation*, vol. 24. New York, NY, USA: Academic, 1989, pp. 109–165.
[2] J. Kirkpatrick, R. Pascanu, N. C. Rabinowitz, J. Veness, G. Desjardins, A. A. Rusu, K. Milan, J. Quan, T. Ramalho, A. Grabska-Barwińska, D. Hassabis, C. Clopath, D. Kumaran, and R. Hadsell, "Overcoming catastrophic forgetting in neural networks," *Proc. Nat. Acad. Sci. USA*, vol. 114, no. 13, pp. 3521–3526, Mar. 2017.
[3] D. J. C. MacKay, "A practical Bayesian framework for backpropagation networks," *Neural Comput.*, vol. 4, no. 3, pp. 448–472, May 1992.
[4] J. Nocedal and S. J. Wright, *Numerical Optimization*, 2nd ed., New York, NY, USA: Springer, 2006.
[5] A. S. Berahas, M. Jahani, P. Richtárik, and M. Takáč, "Quasi-Newton methods for machine learning: Forget the past, just sample," *Optim. Methods Softw.*, vol. 37, no. 5, pp. 1668–1704, Sep. 2022.
[6] M. De Lange, R. Aljundi, M. Masana, S. Parisot, X. Jia, A. Leonardis, G. Slabaugh, and T. Tuytelaars, "A continual learning survey: Defying forgetting in classification tasks," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 44, no. 7, pp. 3366–3385, Jul. 2022.
[7] L. Wang, X. Zhang, H. Su, and J. Zhu, "A comprehensive survey of continual learning: Theory, method and application," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 46, no. 8, pp. 5362–5383, Aug. 2024.
[8] B. Wickramasinghe, G. Saha, and K. Roy, "Continual learning: A review of techniques, challenges, and future directions," *IEEE Trans. Artif. Intell.*, vol. 5, no. 6, pp. 2526–2546, Jun. 2024.
[9] F. Zenke, B. Poole, and S. Ganguli, "Continual learning through synaptic intelligence," in *Proc. 34th Int. Conf. Mach. Learn.*, vol. 70, Jan. 2017, pp. 3987–3995.
[10] R. Aljundi, F. Babiloni, M. Elhoseiny, M. Rohrbach, and T. Tuytelaars, "Memory aware synapses: Learning what (not) to forget," in *Proc. Eur. Conf. Comput. Vis. (ECCV)*, 2018, pp. 144–161.
[11] S.-W. Lee, J.-H. Kim, J.-H. Jun, J.-W. Ha, and B. Zhang, "Overcoming catastrophic forgetting by incremental moment matching," in *Proc. Adv. Neural Inf. Process. Syst. 30, Annu. Conf. Neural Inf. Process. Syst.*, Jan. 2017, pp. 4652–4662.
[12] C. V. Nguyen, Y. Li, T. D. Bui, and R. E. Turner, "Variational continual learning," in *Proc. Int. Conf. Learn. Represent.*, 2018. [Online]. Available: https://openreview.net/forum?id=BkQqq0gRb
[13] A. Kutalev, "Natural way to overcome the catastrophic forgetting in neural networks," 2020, *arXiv:2005.07107*.
[14] J. Schwarz, W. M. Czarnecki, J. Luketina, A. Grabska-Barwińska, Y. W. Teh, R. Pascanu, and R. Hadsell, "Progress & compress: A scalable framework for continual learning," in *Proc. 35th Int. Conf. Mach. Learn. (ICML)*, Stockholm, Sweden, Jul. 2018, pp. 4528–4537.
[15] A. Chaudhry, P. K. Dokania, T. Ajanthan, and P. H. S. Torr, "Riemannian walk for incremental learning: Understanding forgetting and intransigence," in *Proc. Eur. Conf. Comput. Vis. (ECCV)*, Jan. 2018, pp. 556–572.
[16] X. Liu, M. Masana, L. Herranz, J. Van de Weijer, A. M. López, and A. D. Bagdanov, "Rotate your networks: Better weight consolidation and less catastrophic forgetting," in *Proc. 24th Int. Conf. Pattern Recognit. (ICPR)*, Aug. 2018, pp. 2262–2268.
[17] H. Ritter, A. Botev, and D. Barber, "Online structured Laplace approximations for overcoming catastrophic forgetting," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 31, May 2018, pp. 3738–3748.
[18] J. Martens and R. Grosse, "Optimizing neural networks with Kronecker-factored approximate curvature," in *Proc. 32nd Int. Conf. Int. Conf. Mach. Learn.*, vol. 37, Jan. 2015, pp. 2408–2417.
[19] J. Lee, H. G. Hong, D. Joo, and J. Kim, "Continual learning with extended Kronecker-factored approximate curvature," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2020, pp. 8998–9007.
[20] G. E. Hinton, O. Vinyals, and J. B. Dean, "Distilling the knowledge in a neural network," in *Proc. NeurIPS*, 2014.
[21] Z. Li and D. Hoiem, "Learning without forgetting," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 40, no. 12, pp. 2935–2947, Dec. 2018.
[22] H. Jung, J. Ju, M. Jung, and J. Kim, "Less-forgetting learning in deep neural networks," 2016, *arXiv:1607.00122*.
[23] A. Rannen, R. Aljundi, M. B. Blaschko, and T. Tuytelaars, "Encoder based lifelong learning," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, Oct. 2017, pp. 1329–1337.
[24] S.-A. Rebuffi, A. Kolesnikov, G. Sperl, and C. H. Lampert, "ICaRL: Incremental classifier and representation learning," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2017, pp. 5533–5542.
[25] P. Buzzega, M. Boschini, A. Porrello, D. Abati, and S. Calderara, "Dark experience for general continual learning: A strong, simple baseline," in *Proc. Adv. Neural Inf. Process. Syst.*, Jan. 2020, pp. 15920–15930.
[26] A. Douillard, M. Cord, C. Ollion, T. Robert, and E. Valle, "PODNet: Pooled outputs distillation for small-tasks incremental learning," in *Proc. Eur. Conf. Comput. Vis. (ECCV)*, Jan. 2020, pp. 86–102.
[27] M. Farajtabar, N. Azizan, A. Mott, and A. Li, "Orthogonal gradient descent for continual learning," in *Proc. Int. Conf. Artif. Intell. Statist.*, 2020, pp. 3762–3773.
[28] G. Saha, I. Garg, and K. Roy, "Gradient projection memory for continual learning," in *Proc. Int. Conf. Learn. Represent.*, 2021. [Online]. Available: https://openreview.net/forum?id=3AOj0RCNC2
[29] S. Wang, X. Li, J. Sun, and Z. Xu, "Training networks in null space of feature covariance for continual learning," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2021, pp. 184–193.
[30] Y. Teng, A. Choromanska, and M. Campbell, "Continual learning with direction-constrained optimization," in *Proc. MetaLearn (NeurIPS)*, 2020.
[31] Y. Guo, W. Hu, D. Zhao, and B. Liu, "Adaptive orthogonal projection for batch and online continual learning," in *Proc. AAAI Conf. Artif. Intell.*, Jun. 2022, vol. 36, no. 6, pp. 6783–6791.
[32] A. Chaudhry, N. Khan, P. K. Dokania, and P. H. S. Torr, "Continual learning in low-rank orthogonal subspaces," in *Proc. Adv. Neural Inf. Process. Syst.*, Jan. 2020, pp. 9900–9911.
[33] D. Rolnick, A. Ahuja, J. Schwarz, T. Lillicrap, and G. Wayne, "Experience replay for continual learning," in *Proc. Adv. Neural Inf. Process. Syst.*, 2019. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2019/file/fa7cdfad1a5aaf8370ebeda47a1ff1c3-Paper.pdf
[34] D. López-Paz and M. Ranzato, "Gradient episodic memory for continual learning," in *Proc. Adv. Neural Inf. Process. Syst.*, 2017. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2017/file/f87522788a2be2d171666752f97ddebb-Paper.pdf
[35] D. Maltoni and V. Lomonaco, "Continuous learning in single-incremental-task scenarios," *Neural Netw.*, vol. 116, pp. 56–73, Aug. 2019.
[36] A. Chaudhry, M. Ranzato, M. Rohrbach, and M. Elhoseiny, "Efficient lifelong learning with A-GEM," in *Proc. ICLR*, 2019. [Online]. Available: https://openreview.net/forum?id=Hkf2_sC5FX
[37] G. Hu, W. Zhang, H. Ding, and W. Zhu, "Gradient episodic memory with a soft constraint for continual learning," 2020, *arXiv:2011.07801*.
[38] M. Riemer, I. Cases, R. Ajemian, M. Liu, I. Rish, Y. Tu, and G. Tesauro, "Learning to learn without forgetting by maximizing transfer and minimizing interference," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2019. [Online]. Available: https://openreview.net/forum?id=B1gTShAct7

[39] E. Verwimp, M. De Lange, and T. Tuytelaars, "Rehearsal revealed: The limits and merits of revisiting samples in continual learning," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV)*, Oct. 2021, pp. 9365–9374.
[40] H. Shin, J. K. Lee, J. Kim, and J. Kim, "Continual learning with deep generative replay," in *Proc. 31st Int. Conf. Neural Inf. Process. Syst.*, Red Hook, NY, USA, Jan. 2017, pp. 2994–3003.
[41] S. Farquhar and Y. Gal, "A unifying Bayesian view of continual learning," in *Proc. Bayesian Deep Learn. Workshop (NeurIPS)*, 2018.
[42] G. Sokar, D. C. Mocanu, and M. Pechenizkiy, "Learning invariant representation for continual learning," in *Proc. AAAI-21st Int. Workshop Meta-Learn. Comput. Vis.*, 2020.
[43] A. A. Rusu, N. C. Rabinowitz, G. Desjardins, H. Soyer, J. Kirkpatrick, K. Kavukcuoglu, R. Pascanu, and R. Hadsell, "Progressive neural networks," 2016, *arXiv:1606.04671*.
[44] C. Fernando, D. Banarse, C. Blundell, Y. Zwóls, D. Ha, A. A. Rusu, A. Pritzel, and D. Wierstra, "PathNet: Evolution channels gradient descent in super neural networks," 2017, *arXiv:1701.08734*.
[45] J. Yoon, E. Yang, J. Lee, and S. J. Hwang, "Lifelong learning with dynamically expandable networks," in *Proc. ICLR*, 2018. [Online]. Available: https://openreview.net/forum?id=Sk7KsfW0-
[46] J. Serrà, D. Surís, M. Miron, and A. Karatzoglou, "Overcoming catastrophic forgetting with hard attention to the task," in *Proc. 35th Int. Conf. Mach. Learn.*, Jan. 2018, pp. 4548–4557.
[47] V. Lomonaco and D. Maltoni, "CORe50: A new dataset and benchmark for continuous object recognition," in *Proc. 1st Annu. Conf. Robot Learn.*, Oct. 2017, pp. 17–26.
[48] F. Huszár, "Note on the quadratic penalties in elastic weight consolidation," *Proc. Nat. Acad. Sci. USA*, vol. 115, no. 11, pp. E2496–E2497, Mar. 2018.
[49] Ya Le and Xuan S. Yang, "Tiny ImageNet visual recognition challenge," Stanford CS231N Course, Tech. Rep., 2015.
[50] Y. LeCun and C. Cortes, "MNIST handwritten digit database," Google Labs, Microsoft Res., New York Univ., New York, NY, USA, Tech. Rep., 2010.
[51] A. Krizhevsky, "Learning multiple layers of features from tiny images," Univ. Toronto, Toronto, ON, Canada, Tech. Rep. 0, 2009.
[52] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng, "Reading digits in natural images with unsupervised feature learning," in *Proc. NIPS*, 2011.
[53] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-MNIST: A novel image dataset for benchmarking machine learning algorithms," 2017, *arXiv:1708.07747*.
[54] Y. Bulatov, *Notmnist Dataset*, Google (Books/OCR), 2011.
[55] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *J. Mach. Learn. Res.*, vol. 15, no. 1, pp. 1929–1958, Jan. 2014.
[56] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *Proc. 3rd Int. Conf. Learn. Represent. (ICLR)*, San Diego, CA, USA, 2015.
[57] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Los Alamitos, CA, USA, Jun. 2016, pp. 770–778.
[58] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998.
[59] A. Paszke, "PyTorch: An imperative style, high-performance deep learning library," in *Proc. Adv. Neural Inf. Process. Syst. 32*, Jan. 2019, pp. 8024–8035.
[60] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 30, Jun. 2017, pp. 5998–6008.
[61] S. V. Eeckt and H. Van Hamme, "Using adapters to overcome catastrophic forgetting in end-to-end automatic speech recognition," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP)*, Jun. 2023, pp. 1–5.



STEVEN VANDER EECKT received the Master of Science degree in mathematical engineering (burgerlijk ingenieur) from KU Leuven, in 2019, and the Master of Science degree in data science and big data from Université Libre de Bruxelles, in 2020. He is currently pursuing the Ph.D. degree with the Department of Electrical Engineering ESAT-PSI, KU Leuven. His Ph.D. topic is continual learning in speech recognition.



HUGO VAN HAMME (Senior Member, IEEE) received the master's degree in engineering (burgerlijk ingenieur) from Vrije Universiteit Brussel (VUB), Brussels, Belgium, in 1987, the M.Sc. degree from Imperial College London, London, U.K, in 1988, and the Ph.D. degree in electrical engineering from VUB, in 1992. From 1993 to 2002, he was with LH Speech Products and ScanSoft, initially as a Senior Researcher and later as a Research Manager. Since 2002, he has been a Professor with the Department of Electrical Engineering, KU Leuven, Leuven, Belgium. His main research interests include automatic speech assessment, assistive speech technology, source separation, noise robust speech recognition, models of language acquisition, and computer-assisted learning.