How do Quadratic Regularizers Prevent Catastrophic Forgetting: The Role of Interpolation

Ekdeep Singh Lubana, Puja Trivedi, Danai Koutra, & Robert P. Dick
EECS Department
University of Michigan, Ann Arbor
USA
{eslubana, pujat, danaik, dickrp}@umich.edu

Abstract

Catastrophic forgetting undermines the effectiveness of deep neural networks (DNNs) in scenarios such as continual learning and lifelong learning. While several methods have been proposed to tackle this problem, there is limited work explaining why these methods work well. This paper has the goal of better explaining a popularly used technique for avoiding catastrophic forgetting: quadratic regularization. We show that quadratic regularizers prevent forgetting of past tasks by interpolating current and previous values of model parameters at every training iteration. Over multiple training iterations, this interpolation operation reduces the learning rates of more important model parameters, thereby minimizing their movement. Our analysis also reveals two drawbacks of quadratic regularization: (a) dependence of parameter interpolation on training hyperparameters, which often leads to training instability and (b) assignment of lower importance to deeper layers, which are generally the place forgetting occurs in DNNs. Via a simple modification to the order of operations, we show these drawbacks can be easily avoided, resulting in 6.2% higher average accuracy at 4.5% lower average forgetting. We confirm the robustness of our results by training over 2000 models in different settings. [1]

1 Introduction

Learning algorithms are often designed under the assumption of independent and identical data distributions. However, this assumption is violated in several practical scenarios, such as continual learning and lifelong learning, where data distributions evolve constantly. In such settings, deep neural networks (DNNs) witness catastrophic forgetting and have difficulty adapting to new tasks without losing performance on previously learned ones. Several past works have tried to address this problem. For example, Kirkpatrick et al. (2017); Aljundi et al. (2018); Zenke et al. (2017); Chaudhry et al. (2018) propose quadratic regularizers that penalize changes in parameters which are important for preserving performance on past tasks; Chaudhry et al. (2019b); Riemer et al. (2019); Aljundi et al. (2019); Shin et al. (2017) describe the use of memory buffers or generative models to fine-tune on samples from past tasks while learning new ones; and Rusu et al. (2016); Li et al. (2019); Yoon et al. (2018) propose dynamic modification of network architecture to increase model capacity for accommodating new tasks. While these works have shown promising results, a detailed understanding of their proposed methods is still to be developed. Understanding the reasons due to which existing methods for preventing catastrophic forgetting work or fail can open the possibility of developing better methods.

With this motivation, in this work, we analyze *quadratic regularization*, a popular technique for preventing catastrophic forgetting in DNNs. Specifically, quadratic regularization based methods penalize changes in model parameters that are important for maintaining performance on previously learned tasks. For example, if θ_n denotes model parameters corresponding to the n^{th} task, T_n, then the total training loss under quadratic regularization follows

$$L = L_{T_n} + \frac{\lambda}{2} \sum_k \alpha_{n-1}^{(k)} \left(\theta_n^{(k)} - \theta_{n-1}^{*(k)} \right)^2, \tag{1}$$

where L_{T_n} is the n^{th} task's loss, λ is a regularization constant, θ_{n-1}^* is model parameterization at the end of $(n-1)^{\text{th}}$ task, α_{n-1} is the importance of parameters according to tasks 0 to $n-1$, and $v^{(k)}$ indexes a vector v at location k. Different methods define importance differently. For ex., Aljundi et al. (2018) define importance as the sensitivity of

[1] Code available at https://github.com/EkdeepSLubana/QRforgetting

model output to changes in its parameters, while Kirkpatrick et al. (2017) define importance as the diagonal of Fisher information matrix.

Beyond continual/lifelong learning and related applications, quadratic regularizers are often used for improving performance in transfer learning scenarios (Li et al. (2018)). However, despite their clear importance, there is limited work explaining why quadratic regularizers reduce catastrophic forgetting (see Ramasesh et al. (2021); Yin et al. (2020) for notable exceptions). To bridge this gap, we analyze parameter updates under quadratic regularization and show it prevents forgetting by *interpolating current values of model parameters and their values at the end of the previous task's training* (see Equation 3). By unrolling this interpolation operation over multiple iterations, we further show that, *in proportion to their importance, quadratic regularizers change the "effective" learning rate of a parameter* (Equation 4): learning rate of parameters that are more important to previously learned tasks is reduced, disallowing their change; meanwhile parameters that are less important are allowed to change relatively freely, allowing the model to accommodate new tasks.

Our analysis exposes two important drawbacks to using quadratic regularization for preventing catastrophic forgetting. (i) We find improper hyperparameter configurations can result in *extrapolation*, instead of interpolation, of parameters (see Section 3). This often leads to unstable training (see Section 3.1). (ii) To satisfy conservation properties associated with the hierarchical nature of DNNs (Du et al. (2018); Kunin et al. (2021)), generally used importance definitions in quadratic regularization assign lower importance values to parameters in deeper layers (see Section 3.1). As recently shown by Ramasesh et al. (2021), forgetting in DNNs is primarily caused by changes to deeper layers' parameters. Thus, assignment of lower importance to deeper layers reduces the effectiveness of quadratic regularizers. Interestingly, both these limitations can be eliminated by breaking the update into two stages, thus making interpolation of parameters an explicit operation (see Section 4).

Main Contributions: In this work, we analyze quadratic regularization based methods for mitigating catastrophic forgetting. We train more than 2000 models to verify our analysis. Our main contributions follow.

- **Role of Interpolation in Quadratic Regularizers (Section 3).** We analyze parameter updates under quadratic regularization and show that it interpolates current and past values of model parameters to prevent catastrophic forgetting at a given training iteration. Unrolled over multiple iterations, this interpolation operation reduces the effective learning rate of parameters that are more important for preserving performance on previously learned tasks.
- **Limitations in Quadratic Regularization (Section 3.1).** We identify two primary drawbacks in the formulation of quadratic regularization: (a) due to dependence of the interpolation operation on training hyperparameters, improper hyperparameter configurations often result in extrapolation of parameters (instead of interpolation), hence resulting in training instability and (b) assignment of lower importance to parameters in deeper layers, a consequence of the hierarchical nature of DNNs, makes it difficult for quadratic regularizers to prevent catastrophic forgetting.
- **Avoiding Limitations via Explicit Interpolation (Section 4).** By making the interpolation operation explicit, we show training instability issues caused by sensitivity to training hyperparameters can be completely avoided. Further, we show that redefining importance scores relative to past tasks prevents catastrophic forgetting caused by inappropriate changes to deeper layer parameters. Overall, using these modifications enable quadratic regularizers to achieve 6.2% higher average accuracy at 4.5% lower average forgetting.

2 Related Work

Several prior works have proposed solutions to the problem of catastrophic forgetting in DNNs. Due to their higher relevance to our work, we focus on quadratic regularization based methods in this section. Detailed discussion of other techniques is provided in the appendix.

Quadratic Regularization Based Methods: Kirkpatrick et al. (2017) developed Elastic Weight Consolidation (EWC), a quadratic regularization strategy that defines the importance of model parameters using the diagonal of the Fisher information matrix. Since EWC, several quadratic regularizers have been proposed. For ex., Zenke et al. (2017) describe Synaptic Intelligence (SI), which defines a parameter's importance as the contribution of that parameter to reduction in loss for previous tasks; Ritter et al. (2018) replace EWC's diagonal Fisher approximation with a block-diagonal Hessian approximation; Aljundi et al. (2018) describe Memory Aware Synapses (MAS), for which a parameter's importance is defined as the sensitivity of model output to change in that parameter; and Chaudhry et al. (2018) describe Riemannian Walk (RWalk), which defines a parameter's importance as the sum of the diagonal of the Fisher information matrix and the contribution of that parameter to reduction in loss for previous tasks.

Understanding Methods for Mitigating Catastrophic Forgetting: A few papers have sought to understand the reasons behind catastrophic forgetting and the effectiveness of existing methods for alleviating it (Krishnan & Balaprakash

(2021); Mirzadeh et al. (2020; 2021); Doan et al. (2021); Knoblauch et al. (2020); Bennani et al. (2020)). We specifically highlight an important relevant work by Ramasesh et al. (2021), who use Centered Kernel Alignment (CKA) (Kornblith et al. (2019)) to measure representational similarity between models trained on different numbers of tasks. The authors demonstrate that catastrophic forgetting primarily arises due to adaptation of deeper layers to new tasks. Solutions that prevent catastrophic forgetting minimize this adaptation, thus ensuring the model can perform well on both novel and previously learned tasks. Closely related to our work is also the recent paper by Yin et al. (2020), who analyze quadratic regularizers by focusing on their asymptotic properties (i.e., convergence and generalization behavior). Similarly, Benzing (2020) discusses how different importance defintions in popular quadratic regularizers are related to each other via different approximations of the Fisher information matrix. In contrast to these papers, we focus on the model updates themselves: our work aims to understand limitations in quadratic regularization methods arising from its gradient descent dynamics. As we show, an implicit interpolation operation helps QR methods prevent catastrophic forgetting, and its interplay with training hyperparameters yields clear regimes of training instability in QR methods. We also note that some prior works have proposed Bayesian perspectives for understanding and improving quadratic regularization techniques by introducing better importance definitions (Huszar (2018); Kirkpatrick et al. (2017); Ritter et al. (2018)), however these works often make strong assumptions that are unlikely to hold in practice, such as assuming the model gradient remains zero for previous tasks (to allow a Laplace or second-order Taylor's approximation). In contrast, our work makes no assumptions about the model's capabilities for solving a prior task, using the exact dynamics equations.

3 The Role of Interpolation in Quadratic Regularization

We first establish notations used throughout the paper. θ_n denotes model parameters and L_{T_n} denotes a task-specific loss for the n^{th} task; θ^*_{n-1} denotes model parameters at the end of task $n-1$; η denotes the learning rate; and λ denotes the regularization constant. We use $v^{(k)}$ to index a vector v at location k. To match the empirical setup in which quadratic regularizers generally function, we assume importance scores for regularizing the n^{th} task (denoted α_{n-1}) are calculated during training of tasks 0 to $n-1$. These scores stay constant as the n^{th} task proceeds. The total loss for learning the n^{th} task follows Equation 1.

Our analysis into the mechanics of quadratic regularization is based on a decomposition of the parameter updates. Specifically, computing the gradient of Equation 1, we see the parameter vector, θ_n, updates as follows:

$$\theta_n \rightarrow \theta_n - \eta \left(\nabla_{\theta_n} L_{T_n} + \lambda \alpha_{n-1} \odot \left(\theta_n - \theta^*_{n-1} \right) \right). \tag{2}$$

Rearranging Equation 2, we see the k^{th} model parameter updates as follows:

$$\theta_n^{(k)} \rightarrow \underbrace{\left(1 - \eta\lambda\alpha_{n-1}^{(k)}\right)\theta_n^{(k)} + \left(\eta\lambda\alpha_{n-1}^{(k)}\right)\theta_{n-1}^{*(k)}}_{\text{Interpolation b/w current and previous values}} \underbrace{-\eta\frac{\partial L_{T_n}}{\partial \theta_n^{(k)}}}_{\text{Task Derivative}}. \tag{3}$$

The above equation illustrates that under quadratic regularization, *a new task is learned by moving model parameters along task-specific derivatives*, while forgetting of previously learned tasks is prevented by *simultaneously performing an interpolation operation between current and previous values of model parameters*. Essentially, the interpolation operation minimizes drift of important model parameters, thus ensuring one can retain performance on previously learned tasks while learning new ones. To understand how mere interpolation of parameters can help mitigate catastrophic forgetting over multiple iterations, we derive the total change in model parameters between the $(n\text{-}1)^{\text{th}}$ and the n^{th} tasks. Specifically, if the task-specific gradient for the j^{th} training iteration is denoted as g_j, then unrolling the parameter updates in Equation 3 via recursive substitution for i iterations yields:

$$\theta_n^{(k)} = \theta_{n-1}^{*(k)} - \sum_{j=0}^{i-1} \underbrace{\left[\left(1 - \eta\lambda\alpha_{n-1}^{(k)}\right)^{i-j-1}\eta\right]}_{\text{Effective Learning Rate}} g_j^{(k)}. \tag{4}$$

Equation 4 shows that, *based upon its importance, quadratic regularizers change the learning rate of a parameter*. In particular, the effective learning rate at which the k^{th} parameter updates is calculated by multiplying η with exponents of $1 - \eta\lambda\alpha_{n-1}^{(k)}$. If the parameter's importance is high, the value of $1 - \eta\lambda\alpha_{n-1}^{(k)}$ becomes smaller, consequently reducing the parameter's learning rate and restricting its change. If the parameter's importance is low, the value of $1 - \eta\lambda\alpha_{n-1}^{(k)}$ remains essentially unchanged; consequently, the parameter's effective learning rate is approximately equal to η, allowing it to freely change according to task derivatives and enable learning of new tasks.

Overall, Equation 3 and Equation 4 establish the exact mechanisms by which quadratic regularizers help mitigate catastrophic forgetting. Specifically, Equation 3 shows that in the short-term, quadratic regularizers use a weighted

interpolation of model parameters with their previous values to prevent changes to important parameters. Meanwhile, Equation 4 shows that, in the long-term, quadratic regularizers reduce the learning rate of important parameters, slowing their change as new tasks are learned. These mechanisms alter a model's optimization path, allowing unimportant parameters to adapt and accommodate new tasks, while restricting movement of important parameters to retain performance on previously learned tasks and prevent catastrophic forgetting. We also note that since the weights used in parameter interpolation (Equation 3) depend on the product of the learning rate (η), the regularization constant (λ), and the parameter's importance ($\alpha^{(k)}$), essentially these three variables determine the effectiveness of a regularizer.

3.1 Limitations in Quadratic Regularization

Having elucidated the role of parameter interpolation in the effectiveness of quadratic regularizers, we now show how the dependence of this operation on the product $\eta\lambda\alpha^{(k)}$ introduces several drawbacks. We intentionally interleave our analysis with experimental evidence to show our claims hold well in practice.

Setup: We study four widely used quadratic regularizers: *EWC* (Kirkpatrick et al. (2017)), *MAS* (Aljundi et al. (2018)), *SI* (Zenke et al. (2017)), and *RWalk* (Chaudhry et al. (2018)). Training without any strategy to mitigate catastrophic forgetting is called *plain fine-tuning*. We use a 6-layer CNN trained using SGD at a fixed learning rate (η) of 0.001 on CIFAR-100 tasks (10 tasks; 10 classes per task). This is similar to architectures used in prior work: Zenke et al. (2017) use a 6-layer model (4 convolutional+2 dense layers); Chaudhry et al. (2018) use a 5-layer model (4 convolutional+1 dense layer); Aljundi et al. (2018) use AlexNet (5 convolutional+3 dense layers). We define training as *unstable* if loss proceeds towards infinity, yielding out of precision (*NaN*) gradients and parameters. Our experiments in Section 3.1.1 follow the 1 epoch, batch-size 10 setting advocated by Lopez-Paz & Ranzato (2017), who argue that for lifelong learning problems, the model should be allowed to see a sample only once. We highlight that training instabilities elucidated in this section emerge in the first few training iterations themselves (<10, generally); hence, these results are not sensitive to the number of epochs. Our analysis in Section 3.1.2 trains model in the stable training regime, where number of epochs can matter. We thus provide results for both 1 epoch, batch-size 10 and 30 epochs, batch-size 256 setting. We find our results follow the same qualitative patterns for both settings. Results are reported on the following metrics (Chaudhry et al. (2018)): (a) *average accuracy*, defined as average test accuracy achieved by the final model over all tasks, and (b) *average forgetting*, defined as the average amount of forgetting over all tasks, where an individual task's forgetting is described as the difference between the maximum and final accuracy achieved on it. All results are averaged over five seeds; standard deviations are reported as error bars in figures. For results on other datasets, see appendix.

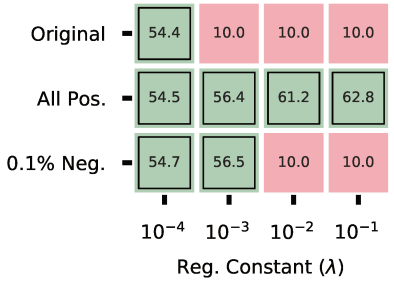


Figure 1: **Case 1: $\eta\lambda\alpha^{(k)}_{n-1} < 0$. Negative importance scores lead to unstable training.** Green, outlined (red) cells imply stable (unstable) training. We analyze a 6-layer CNN trained using SI on 10 CIFAR-100 tasks. *Original* implies both positive and negative scores are allowed, *All Pos.* implies all scores are positive, and *0.1% Neg.* implies only 0.1% of parameters are allowed negative scores. Average accuracy is noted inside figure cells (random accuracy=10%, since any given task has 10 classes; plain fine-tuning accuracy=55.2%). Training is always stable when all scores are positive. However, allowing even a few negative scores produces unstable training. A small λ mitigates this behavior, but reduces the regularizer's effectiveness, resulting in similar performance to plain fine-tuning. Results on other datasets are provided in the appendix.

3.1.1 Extrapolation and Training Instability

For the k^{th} parameter, the interpolation operation at every training iteration follows: $(1-\eta\lambda\alpha_{n-1}) \odot \theta^{(k)}_n + (\eta\lambda\alpha_{n-1}) \odot \theta^{*(k)}_{n-1}$. Since there are generally no constraints on the training hyperparameters and since the scale of importance values can be arbitrarily large or small (e.g., see Figure 3), it is possible for the value of $\eta\lambda\alpha^{(k)}$ to become either negative or greater than 1. In both cases, the interpolation operation converts into an *extrapolation* operation. In this extrapolation regime, we find quadratic regularizers witness unstable training and severe performance degradation.

Case 1. $\eta\lambda\alpha^{(k)}_{n-1} < 0$. For parameters with negative importance, the value of $1-\eta\lambda\alpha^{(k)}_{n-1} > 1$. This pushes the regularizer to the extrapolation regime for such parameters. To understand the implications of this setting, recall the learning rate of a parameter is multiplied by exponents of $1-\eta\lambda\alpha^{(k)}_{n-1}$ (see Equation 4). This implies, for finite gradient, *parameters with negative importance will experience exponentially large updates*, resulting in unstable training.

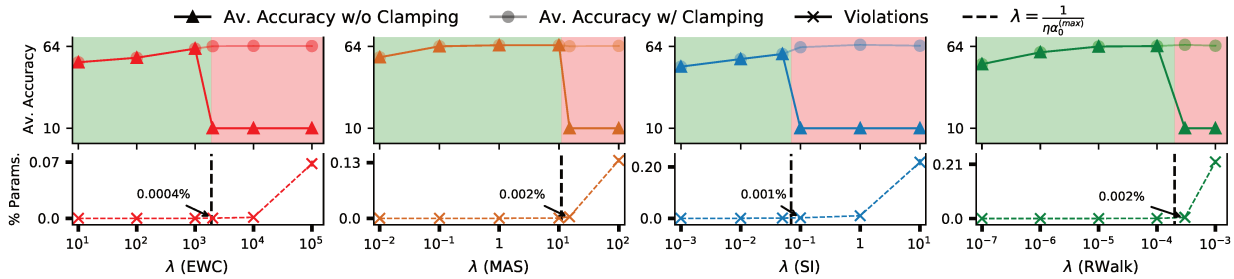


Figure 2: **Case 2: $\eta\lambda\alpha_{n-1}^{(k)} > 1$. Percentage of parameters that violate $\eta\lambda\alpha_0^{(k)} < 1$, leading to unstable training.** We use different values of λ to train a 6-layer CNN on CIFAR-100 tasks. We include results with and without clamping of importance scores, which enforces the constraint that $\eta\lambda\alpha_0^{(k)} \leq 1$ for all parameters. A dotted line marks λ at which $\eta\lambda\alpha_{n-1}^{(k)} = 1$ for just one parameter–we first expect to see unstable training right after this value. Green (Red) shading implies we expect training to be stable (unstable) for the corresponding λ. As shown, across 1.2 million parameters, with even a few violations, training becomes unstable: 5 (0.0004% parameters) for EWC, 26 (0.002% parameters) for MAS, 12 (0.001% parameters) for SI, 31 (0.002% parameters) for RWalk. The use of clamping helps avoid this instability and yields good performance, indicating the violating parameters were the cause for training instability. Results on other datasets are provided in the appendix.

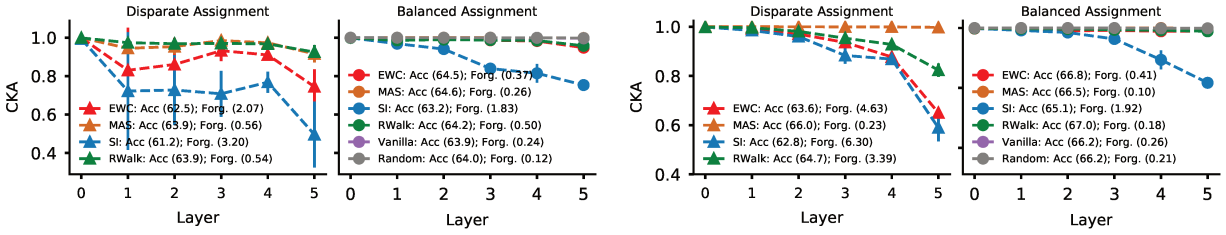
To experimentally test this claim, we analyze SI, which allows negative importance scores to be assigned to a parameter. We use SI to train models for different values of λ with a fixed learning rate of 0.001. We compare three cases: (a) the original method, which allows both positive and negative scores; (b) using absolute values of the assigned importance scores, thus ensuring all parameters have positive scores; and (c) using absolute values of the importance scores for all but 0.1% randomly picked, negative importance parameters, hence allowing a very small number of negative scores. The results are shown in Figure 1 and lead to the following observations. (i) When only positive scores are allowed, training remains stable and the final model achieves high average accuracy. (ii) The original method, which allow both positive and negative scores, results in unstable training. In fact, allowing even a few negative scores leads to unstable training. For a very small value of λ, we note that this behavior is partially mitigated. This can be attributed to the fact the exponent's base, $1 - \eta\lambda\alpha_{n-1}^{(k)}$, remains approximately 1 for small λ. However, for this setting, the amount of interpolation is very low and hence the effectiveness of quadratic regularization is reduced, resulting in similar performance to plain fine-tuning.

The above experiment corroborates our claim that for negative importance scores, training becomes unstable. This result also shows that *importance scores should be positive for all parameters.* This raises the question, why does SI work well? Interestingly, we find that even though in their paper the authors propose SI as a signed method (i.e., negative scores are allowed) and derive theoretical results for such a signed importance setting, their implementation actually uses a rectifier function (ReLU) to remove negative scores during training (Zenke et al. (2018)). This detail is not discussed in the paper and is likely why the method works well in practice.

Case 2. $\eta\lambda\alpha_{n-1}^{(k)} > 1$. Note that if $\eta\lambda\alpha_{n-1}^{(k)} > 1$ for a parameter, the value of $1 - \eta\lambda\alpha_{n-1}^{(k)} < 0$ for it. That is, similar to Case 1, the regularizer is again pushed into the extrapolation regime. Since $1 - \eta\lambda\alpha_{n-1}^{(k)}$ is multiplied to the current parameter value at every iteration (see Equation 3), for a reasonably high value of the parameter (i.e., greater than the gradient), we see the parameter's *sign will be flipped (changed) every update*, resulting in unstable training.

To demonstrate this claim, we train models using different quadratic regularizers and calculate the number of parameters that violate the inequality $\eta\lambda\alpha_0^{(k)} < 1$ (i.e., after first task) for various regularization constants. As shown in Figure 2, training is *always* stable when λ is small enough to ensure the inequality is not violated. However, with even a *few* violations (*<4–30 out of 1.2 million parameters*), training becomes unstable. To confirm that it is indeed these few violating parameters that cause instability, we also include results with a "clamping" operation, which enforces the constraint $\eta\lambda\alpha_0^{(k)} \leq 1$ for all parameters by reassigning violating parameters an importance score of $1/\eta\lambda$. As can be seen in the figure, the clamped models do not witness any instability and are able to achieve high performance.

Combined, these results corroborate our claim that if $\eta\lambda\alpha_{n-1}^{(k)} > 1$, training becomes unstable. We stress that even though the clamping operation in the experiments above helps address unstable training, its imposed constraint of $\eta\lambda\alpha_{n-1}^{(k)} = 0$ will yield an effective learning rate of 0 for violating parameters (see Equation 10). This implies such



(a) 1 epoch. (b) 30 epochs.

Figure 4: **Balanced importance scores prevent forgetting more effectively.** We use CKA to measure representational similarity between layers of a model trained only on the first task of CIFAR-100 versus the model trained on all tasks. Average accuracy (Acc.) and average forgetting (Forg.) are reported in the legend. Balanced variants ensure importance scores are not biased against deeper layers; they achieve high representational similarity across all layers with minimal forgetting. In contrast, the other measures have significantly lower CKA similarity in deeper layers and, hence, witness more forgetting. Results on other datasets are provided in the appendix.

parameters will be frozen at their current value and not allowed to change. Consequently, if an overly large λ is used, a significant portion of the model may remain frozen, disallowing learning of new skills.

Overall, the above two cases complete our analysis of training instability in the extrapolation regime. Experimentally, we find that as long as the regularizer is in the interpolation regime, training remains stable. In fact, during hyperparameter tuning, we recommend the constraint $0 < \eta\lambda\alpha_{n-1}^{(k)} < 1$ for all parameters should be used to reject values of η and λ that are likely to produce unstable training.

3.1.2 Disparate Importance Assignment

We find that importance definitions used in popular regularizers can assign (much) lower importance to deeper layers than to earlier layers (e.g., see results for EWC in Figure 3). This phenomenon can be explained by borrowing a result from the field of network pruning, where similar importance definitions are used for removing unnecessary parameters in DNNs (Blalock et al. (2020)). Specifically, due to rescale symmetry in DNNs with ReLU non-linearities, layerwise norms of model parameters are conserved across a model (Du et al. (2018); Kunin et al. (2021)). Tanaka et al. (2020) recently show that to satisfy this property, generally used importance definitions assign much lower importance to parameters in deeper layers than to parameters in earlier layers.

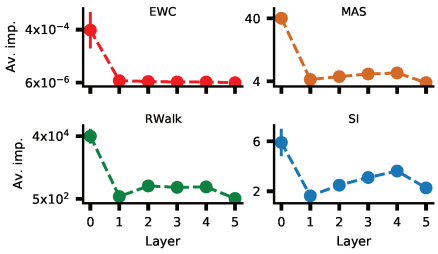


Figure 3: **Average importance is lower for parameters in deeper layers.** We train a 6-layer CNN on the first CIFAR-100 task and analyze importance definitions used by popular quadratic regularizers. As shown, except for SI, importance definitions used in quadratic regularizers result in much lower average importance for parameters in deeper layers. Results on other datasets are provided in the appendix.

To understand the implications of this disparate importance assignment, recall that for stable training, the constraint $\eta\lambda\alpha_{n-1}^{(k)} < 1$ must be satisfied for all parameters. Thus, assuming stable training and a given value of η, the regularization constant (λ) will be bounded as follows:

$$\lambda < \lambda_{\text{upper}} = \min_k \left[\frac{1}{\eta\, \alpha_{n-1}^{(k)}} \right] = \frac{1}{\eta\, \max_k \alpha_{n-1}^{(k)}}. \tag{5}$$

Equation 5 shows that under stable training, *the valid range for λ is restricted by the largest importance score assigned to a model parameter*. However, given the disparate importance assignment in popular quadratic regularizers, will this range be suitable for all parameters in a model? To understand this, we note for a general parameter θ_n^l, the value of $\eta\lambda\alpha_{n-1}^{(l)}$ is bounded as follows:

$$\eta\lambda\alpha_{n-1}^{(l)} < \eta\lambda_{\text{upper}}\alpha_{n-1}^{(l)} = \frac{\alpha_{n-1}^{(l)}}{\max_k \alpha_{n-1}^{(k)}}. \tag{6}$$

If $\alpha_{n-1}^{(l)} \ll \max_k \alpha_{n-1}^{(k)}$, Equation 6 shows the product $\eta\lambda\alpha_{n-1}^{(l)} \to 0$. Thus, *regardless of how large the value of λ is*, we find parameters with relatively lower importance will be permitted to change rapidly during training, enabling the model

to learn new tasks. Indeed, this is the desired behavior. *However*, in the context of DNNs, we have shown standard importance definitions in quadratic regularizers can assign much lower importance to deeper layers (see Figure 3). This indicates *currently used quadratic regularizers will allow a model to change its deeper layers more readily than its earlier layers*. As shown by Ramasesh et al. (2021), forgetting in DNNs is primarily caused by adaptation of deeper layers to recent tasks. Thus, by using lower importance scores for parameters in deeper layers, quadratic regularizers are unable to address the primary source of catastrophic forgetting in DNNs: changes to deeper layers.

Our analysis above also indicates treating deeper layers on the similar scale of importance as earlier layers should allow a quadratic regularizer to prevent forgetting more easily. To experimentally test this, we design quadratic regularizers with balanced importance assignments across layers. This is achieved via rescaling layerwise importance scores, such that the average importance of scores in any layer is the same. We also consider two naive baselines that satisfy this constraint: (a) *Vanilla*–assign unit importance to all parameters, and (b) *Random*–assign uniformly picked, random importance between $[0, 1]$ to all parameters. Following Ramasesh et al. (2021), we analyze each layer's contribution to catastrophic forgetting by using Centered Kernel Alignment (CKA) (Kornblith et al. (2019)) and evaluate similarity of feature representations between a model trained only on the first task and a model trained on all tasks. If forgetting is low, features are similar and CKA is high across all layers of the two models. We use the first 3 tasks of our CIFAR-100 setup to perform a hyperparameter search for λ. The remaining 7 tasks are used for training and evaluation. We train models for both 1 epoch with a batch-size of 10 and 30 epochs with a batch-size of 256. Results are shown in Figure 4. As can be seen, models trained using balanced quadratic regularizers forget the least and have comparable average accuracy. CKA similarity is high across all layers. In contrast, methods with disparate importance assignment forget more and show low CKA similarity in deeper layers. The above experiment corroborates our claim that *catastrophic forgetting is better prevented by assigning similar importance to all layers, than by using existing, more complex importance definitions.* Surprisingly, prior work on quadratic regularization generally do not evaluate their methods against such simple baselines like Vanilla and Random. [2]

4 Addressing Limitations in Quadratic Regularization

Having established the mechanisms by which quadratic regularizers prevent catastrophic forgetting and the corresponding drawbacks of those mechanisms, we seek a simple way to avoid these problems and still prevent catastrophic forgetting. To this end, we note our analysis in Section 3.1 shows that two primary conditions should be satisfied to ensure a quadratic regularizer performs well: (a) the regularizer should function in the interpolation regime and (b) importance scores be relatively constant across layers. We show these conditions can be easily satisfied by breaking the quadratic regularizer update (Equation 3) into two operations:

$$\begin{aligned} &\text{(i)}\ \theta_n \to \theta_n - \eta \nabla_{\theta_n} L_{T_n} \qquad \text{(Task-specific change) and} \\ &\text{(ii)}\ \theta_n \to (1 - R_j) \odot \theta_n + R_j \odot \theta^*_{n-1} \quad \text{(Interpolation).} \end{aligned} \tag{7}$$

Here, the vector R_j is defined to control the amount of interpolation at iteration j. Contrasting this update with the quadratic regularization update in Equation 3, one can see the main difference lies in the order of operations: while quadratic regularizers perform the task-specific update and the interpolation operation *in a single step*, our proposed modification breaks that update into *two separate operations*. This makes the interpolation operation *explicit*, instead of the implicit operation in quadratic regularization. .

To understand the benefit of making interpolation an explicit operation, notice that in Equation 7, *the amount of interpolation is independent of the training hyperparameters*; only the vector R_j controls the amount of interpolation. This allows us to circumvent the need of a regularization coefficient and helps avoid training instabilities arising from gradient descent dynamics (see Section 3.1.1). Further, we can exploit this independence to define R_j in a manner such that the desired conditions for effective quadratic regularization are always satisfied. To this end, we define R_j as a measure of the importance of a parameter for previous tasks relative to its importance for all tasks. Specifically, assume learning of the n^{th} task is currently underway. Again using α_{n-1} to denote importance of parameters according to tasks 0 to $n-1$ and α_{T_n} to denote the importance of parameters according to the n^{th} task, we define the relative importance of a parameter $\theta_n^{(k)}$ as

$$R_j^{(k)} = \frac{\sqrt{\alpha_{n-1}^{(k)}}}{\sqrt{\alpha_{T_n}^{(k)}} + \sqrt{\alpha_{n-1}^{(k)}}}. \tag{8}$$

[2] We note that Kirkpatrick et al. (2017) do have a comparison to distance based regularization, which is similar to our Vanilla technique, but their results are for a setting where one computes and stores importance scores computed after every task. This yields a memory complexity that is proportional to the product of model size and number of tasks, ignoring one of the core desiderata of continual learning that memory requirements should grow at most sub-linearly with the number of tasks (Aljundi (2019)).

Algorithm 1 Quadratic Regularization with Explicit Interpolation Steps

Input: parameterization θ^*_{n-1} and importance α_{n-1} after learning $n-1$ tasks; number of training iterations #iters and Loss L_{T_n} for learning the n^{th} task.
Initialize: $\theta_n = \theta^*_{n-1}$; $\alpha_{T_n} = \mathbf{0}$.
for $j = 0$ **to** #iters-1 **do**
  $\theta_n \rightarrow \theta_n - \eta \nabla_{\theta_n} L_{T_n}$ (Task-specific change)
  Update α_{T_n}
  Compute $R_j = \frac{\sqrt{\alpha_{n-1}}}{\sqrt{\alpha_{T_n}} + \sqrt{\alpha_{n-1}}}$ (Relative importance)
  $\theta_n \rightarrow (1 - R_j) \odot \theta_n + (R_j) \odot \theta^*_{n-1}$ (Interpolation)
end for
$\alpha_n \rightarrow (\alpha_{T_n} + \alpha_{n-1})/2$ (Update importance)
Return: θ_n, α_n

Here, the definition of importance of a parameter can be borrowed from any popular quadratic regularizer. Note that under this formulation, $R_j^{(k)}$ necessarily ranges from 0 to 1 for all parameters, hence satisfying condition (a) and ensuring stable training. If $R_j^{(k)}$ is large (closer to 1), then the k^{th} parameter is relatively more important for preserving performance on previous tasks than for the current task. In this case, the parameter's *previous* value is weighted more heavily to prevent its change. If $R_j^{(k)}$ is small (closer to 0), then the k^{th} parameter is relatively less important for previous tasks. In this case, the parameter's *current* value is weighted more heavily, allowing the parameter to adapt to a new task. Finally, we note that even if a parameter's absolute importance is low, its relative importance can be high if it is unimportant to previous tasks. *Thus, using this definition for R_j, we can also avoid problems related to assignment of lower importance scores to deeper layers.*

We also add that the use of square root in Equation 8 yields us small-but-significant improvements over use of simple magnitude ($\sim$ 1–2% less forgetting, in general). To understand why, recall importance scores in our paper are computed in a running average manner. In a small-batch scenario (one of our primary setups), gradient noise is very high, especially in the early iterations (magnitude of order 100). This noise manifests as gradient spikes that produce large fluctuations in the current task's importance scores. Since the previously computed importance scores remain fixed and since we regularize the model using relative importance to previous tasks' importance scores, these gradient spikes bias learning towards the current task. To avoid this behavior, we use a sublinear function (square root) to minimize sensitivity to fluctuating scores. In fact, we found similar improvements with cube root too (another sublinear function). Over a few iterations, gradient noise is removed by the running average, thus improving signal to noise ratio and yielding reliable estimates. Thus, in the later iterations, both square root and magnitude show similar behavior. However, square root has better ability to avoid forgetting in the early iterations and hence its overall forgetting ends up being smaller.

The overall algorithm for quadratic regularization with explicit interpolation steps is shown in Algorithm 1. Before proceeding, we highlight that instead of using our explicit update formulation, the dependence on hyperparameters could also have been removed by scaling down the regularization coefficient in standard quadratic regularizers with the model learning rate. However, this would still retain the presence of hyperparameters in the overall setup, in contrast with our explicit update formulation.

4.1 Experimental Evaluation

We now evaluate the effectiveness of the proposed quadratic regularizers with explicit interpolation steps. We also provide comparisons with two experience replay-based methods: A-GEM (Chaudhry et al. (2019a)) and ER-Reservoir (Chaudhry et al. (2019b)). We stress that our objective in reporting this comparison is *not* to beat the state-of-the-art, but to evaluate if our modified quadratic regularizers can reduce the gap between regularization and replay based methods, which can be inappropriate for applications where long-term storage of training data undermines privacy or greatly increases cost (e.g., in embedded systems).

We use three datasets of different complexities: (i) CIFAR-100, divided into 10 tasks with 10 classes per task and 500 samples per class; (ii) Oxford-Flowers, divided into 17 tasks with 6 classes per task and 72 samples per class (on average); and (iii) Caltech-256, divided into 32 tasks with 8 classes per task and 95 samples per class (on average). Note that Oxford-Flowers and Caltech-256 have unbalanced classes and few samples per class, which are characteristics of several real-world applications. We use a 6-layer CNN for all datasets and train the model using SGD with momentum for all tasks; results for ResNet-18 are in the appendix. The first 3 tasks for all datasets are reserved to run a grid search for finding the optimal hyperparameter configuration (learning rate/regularization constant). Since the models are

Table 1: Comparison of Average Accuracy (Acc; ↑ indicates higher is better) and Average Forgetting (Forg; ↓ indicates lower is better) for plain and explicit interpolation variants of EWC, MAS, SI, and RWalk, averaged over five seeds (std. dev. are reported in appendix). Results are also provided with two replay based methods (A-GEM and ER-Reservoir) and plain fine-tuning (Plain). We consider three datasets: CIFAR-100 (10 tasks); Oxford-Flowers (17 tasks); and Caltech-256 (32 tasks). For a specific regularizer, the better performing variant is underlined; the best results are in bold. As shown, variants with explicit interpolation steps consistently outperform their Quadratic Regularization counterparts. This behavior is most prominent in datasets with unbalanced classes and longer task sequences (Oxford-Flowers and Caltech-256), where hyperparameter tuning is difficult.

1 Epoch		Plain Quadratic Regularizers						Explicit Interpolation Variants				Replay	
CIFAR	Plain	EWC	MAS	SI	RWalk	Van.	Rand.	EWC	MAS	SI	RWalk	A-GEM	ER
Acc (↑)	55.3	62.5	63.9	61.2	63.9	63.9	64.0	63.8	64.0	63.9	63.8	67.2	**68.8**
Forg (↓)	9.13	2.07	0.56	3.20	0.54	0.24	0.12	0.08	**0.06**	0.07	0.09	1.29	0.69
Flowers	Plain	EWC	MAS	SI	RWalk	Van.	Rand.	EWC	MAS	SI	RWalk	A-GEM	ER
Acc (↑)	37.9	41.3	57.5	36.2	44.9	59.5	58.5	60.0	59.6	59.9	59.4	68.61	**68.9**
Forg (↓)	13.4	10.6	3.61	15.1	9.68	1.19	3.61	1.03	**0.44**	0.99	1.31	4.08	3.45
Cal-256	Plain	EWC	MAS	SI	RWalk	Van.	Rand.	EWC	MAS	SI	RWalk	A-GEM	ER
Acc (↑)	40.1	41.9	53.7	40.9	43.7	55.1	56.3	56.2	57.3	56.0	55.8	63.9	**64.3**
Forg (↓)	6.21	5.59	3.92	5.68	4.69	1.89	1.59	1.41	**0.36**	1.42	1.31	2.89	2.75
30 Epochs		Plain Quadratic Regularizers						Explicit Interpolation Variants				Replay	
CIFAR	Plain	EWC	MAS	SI	RWalk	Van.	Rand.	EWC	MAS	SI	RWalk	A-GEM	ER
Acc (↑)	60.7	63.6	66.3	62.8	64.7	66.2	66.2	66.3	66.0	66.2	66.1	66.4	**66.8**
Forg (↓)	8.01	4.63	0.23	6.30	3.39	0.26	0.21	**0.21**	0.23	0.25	0.29	3.10	1.64
Flowers	Plain	EWC	MAS	SI	RWalk	Van.	Rand.	EWC	MAS	SI	RWalk	A-GEM	ER
Acc (↑)	49.9	54.1	60.5	51.1	55.7	60.3	60.9	61.3	61.6	61.8	61.7	68.9	**69.4**
Forg (↓)	11.8	9.07	1.19	12.4	7.19	**0.36**	1.79	1.07	0.48	0.62	0.71	4.29	3.01
Cal-256	Plain	EWC	MAS	SI	RWalk	Van.	Rand.	EWC	MAS	SI	RWalk	A-GEM	ER
Acc (↑)	40.6	50.3	60.1	50.6	53.7	60.0	60.1	60.6	60.9	60.8	60.6	65.7	**66.1**
Forg (↓)	15.8	7.15	0.99	7.41	4.69	0.32	0.22	0.22	**0.18**	0.32	0.34	2.86	2.32

trained in the stable training regime (enforced via hyperparameter tuning), we train models for both 1 epoch, batch-size 10 setting advocated by Lopez-Paz & Ranzato (2017) and 30 epochs, batch-size 256 setting where we find all models achieve 95–100% training accuracy when training on a given task, ensuring convergence (an assumption made by prior work to derive their regularizer strategies). Further, following prior works (Aljundi et al. (2018); Kirkpatrick et al. (2017); Zenke et al. (2017)), we use a multi-head setting.

Results are shown in Table 1. We note explicit interpolation variants *significantly* outperform plain quadratic regularizers. For ex., in the 1 epoch setting, *accuracy improves by 6.2% on average and up to 23%.* We further make the following observations: (i) On unbalanced datasets and long task sequences, explicit interpolation results in substantial improvements. However, on CIFAR-100, whose tasks correspond to hundreds of parameter updates, plain quadratic regularizers perform competitively. This indicates when longer task sequences are used for hyperparameter search, the resulting configuration is able to perform well for subsequent tasks as well. In contrast, on Oxford-Flowers and Caltech-256, especially in the 1 epoch setting, where the respective tasks have only 43 and 76 parameter updates (on average), searching for adequate hyperparameters is difficult and plain quadratic regularizers result in up to 10% forgetting. (ii) With our proposed modifications, quadratic regularizers achieve witness lower forgetting than replay based methods, but continue to underperform on average accuracy. However, we note that when long-term storage of training data is prevented by privacy or cost concerns, our modifications allow quadratic regularizers to serve as a useful and reliable alternative to replay based methods.

5 Conclusion

In this work, we determine the mechanisms using which quadratic regularization based methods prevent catastrophic forgetting in DNNs. Our primary findings show that in the short-term, these methods interpolate current and past

values of model parameters, disallowing change. In the long-term, they change the learning rate of a parameter in proportion to its importance, reducing the overall movement of more important parameters. Our analysis also shows two primary pitfalls that limit the effectiveness of quadratic regularization: (i) extrapolation, instead of interpolation, of parameters due to inappropriate hyperparameters and (ii) lower importance assignment to deeper layers. We propose an explicit interpolation variant of quadratic regularizers, which is able to circumvent these pitfalls, boasting much better performance, consistently. In future work, we aim to perform similar analyses of the mechanisms which allow replay-based techniques to prevent catastrophic forgetting.

ACKNOWLEDGEMENTS

The authors thank Hidenori Tanaka and anonymous reviewers for valuable feedback on the paper. This work was supported in part by NSF under award CNS-2008151.

REFERENCES

R. Aljundi. Continual Learning in Neural Networks. *arXiv preprint*, arXiv:1910.02718v2 [cs.LG], 2019.

R. Aljundi, F. Babiloni, M. Elhoseiny, M. Rohrbach, and T. Tuytelaars. Memory Aware Synapses: Learning what (not) to forget . In *Proc. European Conf. on Computer Vision (ECCV)*, September 2018.

R. Aljundi, M. Lin, B. Goujaud, and Y. Bengio. Gradient Based Sample Selection for Online Continual Learning. In *Proc. Adv. in Neural Information Processing Systems (NeurIPS)*, volume 32, pp. 11816–11825, 2019.

M. Bennani, T. Doan, and M. Sugiyama. Generalisation Guarantees for Continual Learning with Orthogonal Gradient Descent. *arXiv preprint*, arXiv:2006.11942 [cs.LG], 2020.

F. Benzing. Unifying Regularisation Methods for Continual Learning. *arXiv preprint*, arXiv:2006.06357v2 [cs.LG], 2020.

D. Blalock, J. Ortiz, J. Frankle, and J. Guttag. What is the State of Neural Network Pruning? In *Proc. Conf. on Machine Learning and Systems*, 2020.

A. Chaudhry, P. Dokania, T. Ajanthan, and P. Torr. Riemannian Walk for Incremental Learning: Understanding Forgetting and Intransigence. In *Proc. European Conf. on Computer Vision (ECCV)*, September 2018.

A. Chaudhry, M. Ranzato, M. Rohrbach, and M. Elhoseiny. Efficient Lifelong Learning with A-GEM. In *Proc. Int. Conf. on Learning Representations (ICLR)*, 2019a.

A. Chaudhry, M. Rohrbach, M. Elhoseiny, T. Ajanthan, P. Dokania, P. Torr, and M. Ranzato. On Tiny Episodic Memories in Continual Learning. *arXiv preprint*, arXiv:1902.10486v4 [cs.LG], 2019b.

T. Doan, M. Bennani, B. Mazoure, G. Rabusseau, and P. Alquier. A Theoretical Analysis of Catastrophic Forgetting through the NTK Overlap Matrix. In *Proc. Conf. on Artificial Intelligence and Statistics (AISTATS)*, volume 24, 2021.

S. Du, W. Hu, and J. Lee. Algorithmic Regularization in Learning Deep Homogeneous Models: Layers are Automatically Balanced. In *Proc. Adv. in Neural Information Processing Systems (NeurIPS)*, volume 31, pp. 384–395, 2018.

M. Farajtabar, N. Azizan, A. Mott, and A. Li. Orthogonal Gradient Descent for Continual Learning. In *Proc. Int. Conf. on Artificial Intelligence and Statistics (AISTATS)*, volume 108, pp. 3762–3773, 2020.

S. Farquhar and Y. Gal. Towards Robust Evaluations of Continual Learning. *arXiv preprint*, arXiv:1805.09733v3 [cs.LG], 2019.

F. Huszar. Note on the Quadratic Penalties in Elastic Weight Consolidation. *Proc. National Acad. of Sciences (PNAS)*, 2018.

J. Kirkpatrick, R. Pascanu, N. Rabinowitz, J. Veness, G. Desjardins, A. Rusu, K. Milan, J. Quan, T. Ramalho, A. Grabska-Barwinska, D. Hassabis, C. Clopath, D. Kumaran, and R. Hadsell. Overcoming Catastrophic Forgetting in Neural Networks. *Proc. of the National Academy of Sciences (PNAS)*, 114:3521–3526, 2017.

J. Knoblauch, H. Husain, and T. Diethe. Optimal Continual Learning has Perfect Memory and is NP-hard. In *Proc. Int. Conf. on Machine Learning (ICML)*, volume 98, 2020.

S. Kornblith, M. Norouzi, H. Lee, and G. Hinton. Similarity of Neural Network Representations Revisited. In *Proc. Int. Conf. on Machine Learning (ICML)*, volume 97, pp. 3519–3529, 2019.

R Krishnan and Prasanna Balaprakash. Formalizing the Generalization-Forgetting Trade-Off in Continual Learning. In *Proc. Adv. in Neural Information Processing Systems (NeurIPS)*, volume 35, 2021.

D. Kunin, J Sagastuy-Brena, S. Ganguli, D. Yamins, and H. Tanaka. Neural Mechanics: Symmetry and Broken Conservation Laws in Deep Learning Dynamics. In *Proc. Int. Conf. on Learning Representations (ICLR)*, 2021.

X. Li, Y. Grandvalet, and F. Davoine. Explicit Inductive Bias for Transfer Learning with Convolutional Networks. In *Proc. Int. Conf. on Machine Learning (ICML)*, volume 70, pp. 3987–3995, 2018.

X. Li, Y. Zhou, T. Wu, R. Socher, and C. Xiong. Learn to Grow: A Continual Structure Learning Framework for Overcoming Catastrophic Forgetting. In *Proc. Int. Conf. on Machine Learning (ICML)*, volume 97, pp. 3925–3934, 2019.

Z. Li and D. Hoiem. Learning without Forgetting. *IEEE Tran. on Pattern Analysis and Machine Intelligence (TPAMI)*, 40:2935–2947, 2018.

D. Lopez-Paz and M. Ranzato. Gradient Episodic Memory for Continual Learning. In *Proc. Adv. in Neural Information Processing Systems (NeurIPS)*, volume 30, pp. 6467–6476, 2017.

S. Mirzadeh, M. Farajtabar, R. Pascanu, and H. Ghasemzadeh. Understanding the Role of Training Regimes in Continual Learning. In *Proc. Adv. in Neural Information Processing Systems (NeurIPS)*, 2020.

S. Mirzadeh, M. Farajtabar, D. Gorur, R. Pascanu, and H. Ghasemzadeh. Linear Mode Connectivity in Multitask and Continual Learning. In *Proc. Int. Conf. on Learning Representations (ICLR)*, 2021. URL https://openreview.net/forum?id=Fmg_fQYUejf.

V. Ramasesh, E. Dyer, and M. Raghu. Anatomy of Catastrophic Forgetting: Hidden Representations and Task Semantics. In *Proc. Int. Conf. on Learning Representations (ICLR)*, 2021.

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. on Learning Representations (ICLR)*, 2019.

H. Ritter, A. Botev, and D. Barber. Online Structured Laplace Approximations For Overcoming Catastrophic Forgetting. In *Proc. Adv. in Neural Information Processing Systems (NeurIPS)*, volume 31, pp. 3738–3748, 2018.

A. Rusu, N. Rabinowitz, G. Desjardins, H. Soyer, J. Kirkpatrick, K. Kavukcuoglu, R. Pascanu, and R. Hadsell. Progressive Neural Networks. *arXiv preprint*, arXiv:1606.04671v3 [cs.LG], 2016.

G. Saha and K. Roy. Gradient Project Memory for Continual Learning. In *Proc. Int. Conf. on Learning Representations (ICLR)*, 2021.

H. Shin, J. Lee, J. Kim, and J. Kim. Continual Learning with Deep Generative Replay. In *Proc. Adv. in Neural Information Processing Systems (NeurIPS)*, volume 30, pp. 2990–2999, 2017.

H. Tanaka, D. Kunin, D. Yamins, and S. Ganguli. Pruning Neural Networks Without Any Data by Iteratively Conserving Synaptic Flow. In *Proc. Adv. in Neural Information Processing Systems (NeurIPS)*, volume 33, 2020.

M. Titsias, J. Schwarz, A. Matthews, R. Pascanu, and Y. Teh. Functional Regularisation for Continual Learning with Gaussian Processes. In *Proc. Int. Conf. on Learning Representations (ICLR)*, 2020.

D. Yin, M. Farajtabar, A. Li, N. Levine, and A. Mott. Optimization and Generalization of Regularization-Based Continual Learning: a Loss Approximation Viewpoint. *arXiv preprint*, arXiv:2006.10974v3 [cs.LG], 2020.

J. Yoon, E. Yang, J. Lee, and S. Hwang. Lifelong Learning with Dynamically Expandable Networks. In *Proc. Int. Conf. on Learning Representations (ICLR)*, 2018.

F. Zenke, B. Poole, and S. Ganguli. Continual Learning Through Synaptic Intelligence. In *Proc. Int. Conf. on Machine Learning (ICML)*, volume 70, pp. 3987–3995, 2017.

F. Zenke, B. Poole, and S. Ganguli. Implementation code for Synaptic Intelligence (Zenke et al., 2017). https://github.com/ganguli-lab/pathint/blob/master/pathint/protocols.py#L37, 2018. Github link.

A Appendix

The appendix is organized as follows:

- Appendix B contains a more detailed related work on strategies to prevent catastrophic forgetting.
- Appendix C provides derivation for the relationship between parameters of n^{th} and $(n-1)^{\text{th}}$ task (Equation 5 from main paper).
- Appendix D explains our experimental setup, including details on the datasets used in our experiments, the model architecture, dataset preprocessing, training/evaluation protocol, and the hyperparameter tuning protocol.
- Appendix E provides more results for experiments conducted in Section 3.1 of main paper.
  - Section E.1: Extrapolation Regime Case 1: $\eta\lambda\alpha_{n-1}^{(k)} < 0$.
  - Section E.2: Extrapolation Regime Case 2: $\eta\lambda\alpha_{n-1}^{(k)} > 1$.
  - Section E.3: Disparate Importance Assignment.
- Appendix F provides more results for experiments conducted in Section 4.1 of main paper.
  - Section F.1: Compares plain and explicit interpolation quadratic regularization variants on ResNet-18.
  - Section F.2: Contains a detailed version of Table 1 with standard deviations of the results.

B Related Work

Regularization Based Methods: We note that beyond quadratic regularization, prior works have also described *functional regularization* strategies based on knowledge distillation (Li & Hoiem (2018)) and Bayesian modeling (Titsias et al. (2020)). In this work, we specifically focus on quadratic regularization techniques.

Replay Based Methods: Another successful approach to mitigate catastrophic forgetting is based on the idea of *experience replay* in biological systems. Such methods maintain a memory buffer of samples from previous tasks and fine-tune the model on this buffer while learning new tasks (Chaudhry et al. (2019b); Riemer et al. (2019); Aljundi et al. (2019)). Shin et al. (2017) train a *generative model* to learn previous tasks' data distributions and generate synthetic samples for experience replay while learning new tasks. Recent works have also used a *gradient episodic memory* (Lopez-Paz & Ranzato (2017); Chaudhry et al. (2019a)) and *orthogonal gradient updates* (Farajtabar et al. (2020); Saha & Roy (2021)) to avoid catastrophic forgetting.

Expansion Based Methods: To mitigate catastrophic forgetting, prior works have also proposed to increase model capacity by allocating more neurons. For example, Rusu et al. (2016) add a small network to the original model every time a new task is learned. The original model remains fixed and the added network is fine-tuned. Similarly, Li et al. (2019) optimize the model architecture via neural architecture search. This "optimized" architecture is then fine-tuned for the next task. Yoon et al. (2018) design dynamically expandable networks, which split neurons on the arrival of new tasks to increase model capacity. While such methods do not suffer from catastrophic forgetting, they can significantly increase memory consumption, making them infeasible for scenarios with long sequences of tasks.

C Derivation of relationship between parameters of n^{th} and $(n-1)^{\text{th}}$ task

Preliminaries: We restate notations used throughout the paper for ease of understanding. θ_n denotes model parameters and L_{T_n} denotes a task-specific loss for the n^{th} task; θ^*_{n-1} denotes model parameters at the end of task $n-1$; $\theta_n[i]$ denotes model parameterization at the i^{th} iteration; η denotes the overall model learning rate; and λ denotes the regularization constant. We use $v^{(k)}$ to index a vector v at location k. To match the empirical setup in which quadratic regularizers generally function, we assume importance scores for regularizing the n^{th} task (denoted α_{n-1}) are calculated during training of tasks 0 to $n-1$. These scores stay constant as the n^{th} task proceeds.

Derivation: Equation 4 of the main paper describes the relationship between model parameters before and after training for the n^{th} task. Specifically, the following relationship is noted:

$$\theta_n^{(k)} = \theta_{n-1}^{*(k)} - \sum_{j=0}^{i-1} \underbrace{\left[\left(1-\eta\lambda\alpha_{n-1}^{(k)}\right)^{i-j-1}\eta\right]}_{\text{Effective Learning Rate}} g_j^{(k)}. \tag{9}$$

To derive the above result, we can unroll the model updates under quadratic regularization. Recall, the model updates as follows (Equation 2 from main paper):

$$\begin{aligned} \theta_n[i+1] &= \theta_n[i] - \eta \left(\nabla_{\theta_n[i]} L_{T_n} + \lambda \alpha_{n-1} \odot \left(\theta_n[i] - \theta^*_{n-1} \right) \right). \\ \text{After rearranging:} \quad \theta_n[i+1] &= (\mathbf{1} - \eta\lambda\alpha_{n-1}) \odot \theta_n[i] + (\eta\lambda\alpha_{n-1}) \odot \theta^*_{n-1} - \eta \nabla_{\theta_n[i]} L_{T_n}. \end{aligned} \tag{10}$$

Recall, the gradient of task-specific loss at iteration j is denoted as $g_j = \nabla_{\theta_n[j]} L_{T_n}$. Then, Equation 10 can be used to show the following:

$$\begin{aligned} \theta_n[1] &= (\mathbf{1} - \eta\lambda\alpha_{n-1}) \odot \theta_n[0] + (\eta\lambda\alpha_{n-1}) \odot \theta^*_{n-1} - \eta g_0 \\ &= (\mathbf{1} - \cancel{\eta\lambda\alpha_{n-1}}) \odot \theta^*_{n-1} + (\cancel{\eta\lambda\alpha_{n-1}}) \odot \theta^*_{n-1} - \eta g_0 \\ &= \theta^*_{n-1} - \eta g_0 \\ \implies \theta_n[1] &= \theta^*_{n-1} - \sum_{j=0}^{0} \left[(1 - \eta\lambda\alpha_{n-1})^{1-j-1} \eta \right] \odot g_j. \end{aligned} \tag{11}$$

where the second equality follows from the fact $\theta_n[0] = \theta^*_{n-1}$ (the model for n^{th} task is initialized according to $(n-1)^{\text{th}}$ task's final parameterization).

Unrolling for further iterations, we have:

$$\begin{aligned} \theta_n[2] &= (\mathbf{1} - \eta\lambda\alpha_{n-1}) \odot \theta_n[1] + (\eta\lambda\alpha_{n-1}) \odot \theta^*_{n-1} - \eta g_1 \\ &= (\mathbf{1} - \eta\lambda\alpha_{n-1}) \odot \left(\theta^*_{n-1} - \eta g_0 \right) + (\eta\lambda\alpha_{n-1}) \odot \theta^*_{n-1} - \eta g_1 \\ &= (\mathbf{1} - \cancel{\eta\lambda\alpha_{n-1}}) \odot \theta^*_{n-1} + (\cancel{\eta\lambda\alpha_{n-1}}) \odot \theta^*_{n-1} - \eta \left(\mathbf{1} - \eta\lambda\alpha_{n-1} \right) \odot g_0 - \eta g_1 \\ &= \theta^*_{n-1} - \eta \left(\mathbf{1} - \eta\lambda\alpha_{n-1} \right) \odot g_0 - \eta g_1 \\ \implies \theta_n[2] &= \theta^*_{n-1} - \sum_{j=0}^{1} \left[(1 - \eta\lambda\alpha_{n-1})^{2-j-1} \eta \right] \odot g_j. \end{aligned} \tag{12}$$

Assume, for the i^{th} iteration, following holds true:

$$\theta_n[i] = \theta^*_{n-1} - \sum_{j=0}^{i-1} \left[(1 - \eta\lambda\alpha_{n-1})^{i-j-1} \eta \right] \odot g_j. \tag{13}$$

Then, for the $(i+1)^{\text{th}}$ iteration, we have:

$$\begin{aligned} \theta_n[i+1] &= (\mathbf{1} - \eta\lambda\alpha_{n-1}) \odot \theta_n[i] + (\eta\lambda\alpha_{n-1}) \odot \theta^*_{n-1} - \eta g_i \\ &= (\mathbf{1} - \eta\lambda\alpha_{n-1}) \odot \left(\theta^*_{n-1} - \sum_{j=0}^{i-1} \left[(1 - \eta\lambda\alpha_{n-1})^{i-j-1} \eta \right] \odot g_j \right) + (\eta\lambda\alpha_{n-1}) \odot \theta^*_{n-1} - \eta g_i \\ &= (\mathbf{1} - \cancel{\eta\lambda\alpha_{n-1}}) \odot \theta^*_{n-1} + (\cancel{\eta\lambda\alpha_{n-1}}) \odot \theta^*_{n-1} - \sum_{j=0}^{i-1} \left[(1 - \eta\lambda\alpha_{n-1})^{i+1-j-1} \eta \right] \odot g_j - \eta g_i \\ &= \theta^*_{n-1} - \sum_{j=0}^{i-1} \left[(1 - \eta\lambda\alpha_{n-1})^{i+1-j-1} \eta \right] \odot g_j - \eta g_i \\ \implies \theta_n[i+1] &= \theta^*_{n-1} - \sum_{j=0}^{i} \left[(1 - \eta\lambda\alpha_{n-1})^{i+1-j-1} \eta \right] \odot g_j. \end{aligned} \tag{14}$$

Hence, assuming Equation 13 holds true for the i^{th} iteration, we see it holds true for the $(i+1)^{\text{th}}$ iteration as well. Since the relationship is true for $i = 1$ (see Equation 11) and $i = 2$ (see Equation 12), thus, by principle of induction, the relationship holds true for all i. This completes the derivation.

D Experimental Setup

We provide further details on our experimental setup in this section.

Table 2: Hyperparamaters found using grid search for different methods/datasets for the 1 epoch, 10 batch-size setting.

Dataset	Plain Quadratic Regularizers						Explicit Interpolation Variants				Replay-Based	
CIFAR	EWC	MAS	SI	RWalk	Van.	Rand	EWC	MAS	SI	RWalk	A-GEM	ER
LR (η)	0.001	0.001	0.001	0.003	0.001	0.001	0.001	0.001	0.001	0.001	0.001	0.001
Reg (λ)	10^3	10^{-2}	10^{-2}	10^{-4}	10^2	10^2	-	-	-	-	-	-
Flowers	EWC	MAS	SI	RWalk	Van.	Rand	EWC	MAS	SI	RWalk	A-GEM	ER
LR (η)	0.003	0.003	0.003	0.003	0.003	0.003	0.003	0.003	0.003	0.003	0.003	0.003
Reg (λ)	10^2	10^{-1}	10^{-3}	10^{-5}	10^3	10^1	-	-	-	-	-	-
Cal-256	EWC	MAS	SI	RWalk	Van.	Rand	EWC	MAS	SI	RWalk	A-GEM	ER
LR (η)	0.001	0.001	0.001	0.001	0.003	0.003	0.001	0.001	0.001	0.001	0.003	0.003
Reg (λ)	10^3	10^{-1}	10^{-4}	10^{-5}	10^2	10^2	-	-	-	-	-	-

Table 3: Hyperparamaters found using grid search for different methods/datasets for 30 epochs, 256 batch-size setting.

Dataset	Plain Quadratic Regularizers						Explicit Interpolation Variants				Replay-Based	
CIFAR	EWC	MAS	SI	RWalk	Van.	Rand	EWC	MAS	SI	RWalk	A-GEM	ER
LR (η)	0.001	0.001	0.001	0.003	0.001	0.001	0.001	0.001	0.001	0.001	0.001	0.001
Reg (λ)	10^3	10^{-2}	10^{-1}	10^{-3}	10^2	10^2	-	-	-	-	-	-
Flowers	EWC	MAS	SI	RWalk	Van.	Rand	EWC	MAS	SI	RWalk	A-GEM	ER
LR (η)	0.003	0.003	0.003	0.003	0.003	0.003	0.003	0.003	0.003	0.003	0.003	0.003
Reg (λ)	10^2	10^{-2}	10^{-1}	10^{-3}	10^3	10^1	-	-	-	-	-	-
Cal-256	EWC	MAS	SI	RWalk	Van.	Rand	EWC	MAS	SI	RWalk	A-GEM	ER
LR (η)	0.001	0.001	0.001	0.001	0.003	0.003	0.001	0.001	0.001	0.001	0.003	0.003
Reg (λ)	10^3	10^{-2}	10^{-2}	10^{-4}	10^2	10^2	-	-	-	-	-	-

D.1 DATASETS

We use CIFAR-100, Oxford-Flowers, and Caltech-256 datasets in our work.

CIFAR-100: CIFAR-100 is a standardly used datasets for evaluating continual/lifelong learning algorithms and has 100 classes overall. We divide the dataset into 10 tasks for all experiments in the paper. Each task corresponds to 10 sequential classes. Any given class has 500 train samples and 100 test samples. The dataset is available at https://www.cs.toronto.edu/~kriz/cifar.html.

Oxford-Flowers: Oxford-Flowers is a collection of 102 classes of flowers and is divided into a train/val/test split by default. We divide the dataset into 17 tasks and 6 classes per task. Similar to CIFAR-100, we use sequential classes for Oxford-Flowers as well. Designed for few-shot learning, the dataset has very few training samples per class. We thus use both the train and validation splits to perform training, while the test split is used for testing. The dataset is unbalanced, with different number of samples for different classes. On average, each class has 72 training and 20 test samples. The dataset is available at https://www.robots.ox.ac.uk/~vgg/data/flowers/.

Caltech-256: Caltech-256 is a standard classification benchmark with 256 classes. We divide the dataset into 32 tasks with 8 classes per task. Similar to CIFAR-100, we use sequential classes for Caltech-256 as well. The dataset is unbalanced, with different number of samples for different classes. On average, each class has 95 training and 27 test samples. The dataset is available at http://www.vision.caltech.edu/Image_Datasets/Caltech256/.

D.2 TRAINING SETUP

In this section, we report our training setup in more detail.

Model: We use a 6-layer CNN with a VGG-16 like architecture. Specifically, the model is configured as [64 conv, 64 conv, Maxpool, 128 conv, 128 conv, Maxpool, 256 conv, 256 conv, Maxpool], where "N conv" indicates a convolutional layer with N number of 3×3 filters and "Maxpool" indicates a Maxpool layer with kernel size 2×2 and stride 2. Each convolutional layer is followed by a ReLU activation layer.

Since quadratic regularizers focus on finding a model parameterization useful for next task while staying close to the previous parameterization, it is important the initial parameterization have high discriminative abilities. Thus, we follow prior works on quadratic regularization (Aljundi et al. (2018); Zenke et al. (2017)) and pretrain our model on CIFAR-10.

Classifier Heads: Following the setup proposed by original works on quadratic regularization and also the other relevant baselines evaluated in our experiments, we use a multi-head setting for all experiments (Kirkpatrick et al. (2017); Aljundi et al. (2018); Zenke et al. (2017); Chaudhry et al. (2018; 2019a); Lopez-Paz & Ranzato (2017)).

We do highlight that in a multi-head setting, with simple datasets (such as, MNIST) and only a few classes per task so that even random performance is quite high (e.g., only 2 classes per task or 50% random performance), most continual/lifelong learning algorithms can perform well (as pointed out by Farquhar & Gal (2019)). However, by using datasets of varying complexity, with both balanced/unbalanced classes, large/few number of samples per class, and enough classes per task so that random performance is sufficiently low (10–16%), our experimental setup circumvents these issues.

Preprocessing: Following general evaluation protocols (Ramasesh et al. (2021)), we do not use data augmentation in our experiments. All samples are preprocessed to have a mean of [0.5, 0.5, 0.5] and standard deviation of [0.5, 0.5, 0.5] using the dataloader.

Training/Evaluation Protocol: All datasets are trained using SGD with momentum of 0.9. Other relevant hyperparameters are determined using a grid search (see below). We train for only 1 epoch per task, with a batch-size of 10, as is expected in streaming data settings Lopez-Paz & Ranzato (2017). The first 3 tasks are reserved for hyperparameter search and the remaining tasks are used to train the model. Only the final model is evaluated on the test splits for all datasets.

Hyperparameters: We follow prior work (Chaudhry et al. (2019a)) and use the first 3 tasks for all datasets to search for training hyperparameters when needed (e.g., for all experiments in Section 6). We use grid-search for finding the hyperparameters, with the following grids:

- Learning rate (η): [0.1, 0.03, 0.01, 0.003, 0.001];
- Regularization constant (λ): [10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}, 1, 10^{1}, 10^{2}, 10^{3}, 10^{4}].

Note our grid for λ is much wider than prior works (Lopez-Paz & Ranzato (2017); Chaudhry et al. (2019a)), which showed quadratic reguarlization approaches fail to achieve high performance. Our work demonstrates the high sensitivity of quadratic regularizers to hyperparameters. We thus note the use of a limited range of hyperparameters in prior works may also have been a factor in poor performance for quadratic regularizers.

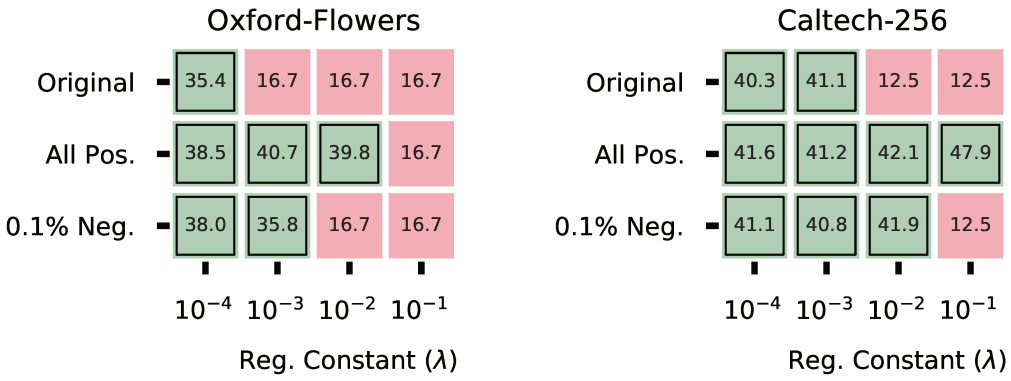
The hyperparameters for different methods and datasets are reported in Table 2 and Table 3. For A-GEM and ER-Reservoir, we use a memory buffer of 500 samples.

E More Results on Experiments from Section 3.1

Section 3.1 of the main paper uses models trained on CIFAR-100 to demonstrate training instability issues caused by hyperparamter sensitivity and the impact of biased importance definitions on the effectiveness of popular quadratic regularizers. In this section, we repeat experiments from Section 3.1 on Oxford-Flowers and Caltech-256. Note that all results in the paper and in this section have been averaged over five seeds.

E.1 Case 1: $\eta\lambda\alpha_{n-1}^{(k)} < 0$.

For parameters with negative importance, the quadratic regularizer enters the extrapolation regime and witnesses unstable training (see **Case 1** in Section 3.1.1 of main paper). In Figure 5, we show this behavior holds true for all datasets considered in this work.



(a) Oxford-Flowers (b) Caltech-256

Figure 5: Impact of negative importance scores. Green, outlined cells indicate stable training; red cells indicate stable training. *Original* implies both positive/negative scores are allowed, *All Pos.* implies all scores are positive, and *0.1% Neg.* implies only 0.1%, randomly picked parameters have negative scores. Average accuracy is mentioned in figure cells. Random accuracy is 10% for CIFAR-100, 16.7% for Oxford-Flowers, and 10% for Caltech-256. Plain fine-tuning accuracy is 55.3% for CIFAR-100, 37.9% for Oxford-Flowers, and 40.1% for Caltech-256. For all positive scores, training is always stable. Including even a few negative importance parameters often produces unstable training and training is rarely stable when both any number of negative importance parameters are allowed. Using a small λ mitigates this behavior, but reduces the regularizer's effectiveness, resulting in similar performance to plain fine-tuning.

E.2 Case 2: $\eta\lambda\alpha_{n-1}^{(k)} > 1$

When $\eta\lambda\alpha_{n-1}^{(k)} > 1$ for a parameter, the weights for interpolation becomes negative ($1 - \eta\lambda\alpha_{n-1}^{(k)} < 0$). This pushes the quadratic regularizer to the extrapolation regime, resulting in unstable training (see **Case 2** in Section 3.1.1 of main paper). In Figure 2 (CIFAR-100), Figure 6 (Oxford-Flowers), and Figure 7 (Caltech-256), we show this behavior holds true for all datasets considered in this work.

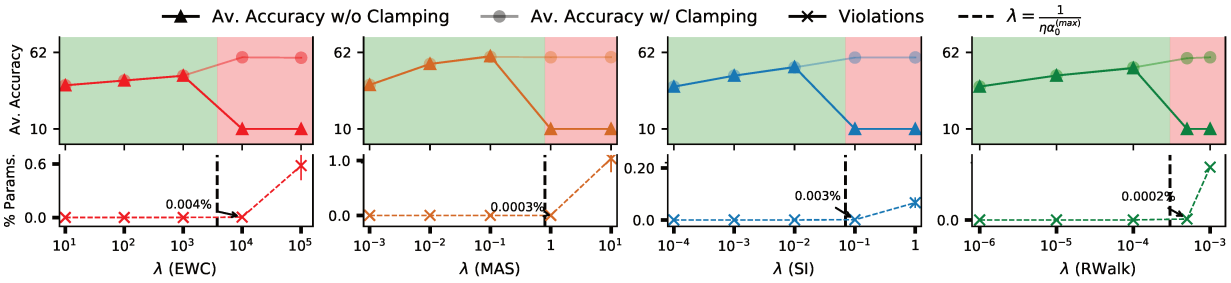


Figure 6: Oxford-Flowers: Number of violations of the inequality $\eta\lambda\alpha_{n-1}^{(k)} < 1$. The minimum number of violations at which we first see unstable training is noted as the lower limit on Y-axis of the plots (stable shaded green; unstable shaded red). As seen, across 1.2 million parameters, even a small number of violations result in unstable training: 35 (0.004% parameters) for EWC, 4 (0.0003% parameters) for MAS, 8 (0.0006% parameters) for SI, and 17 (0.001% parameters) for RWalk.

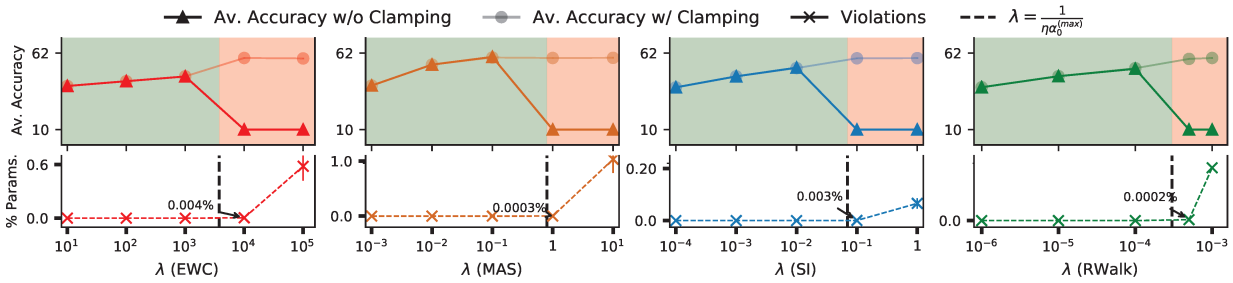
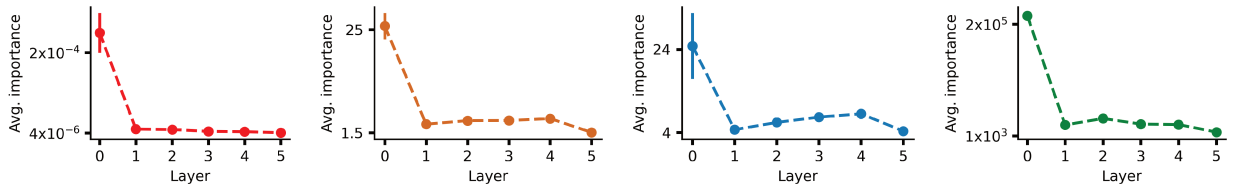


Figure 7: Caltech-256: Number of violations of the inequality $\eta\lambda\alpha_{n-1}^{(k)} < 1$. The minimum number of violations at which we first see unstable training is noted as the lower limit on Y-axis of the plots (stable shaded green; unstable shaded red). As seen, across 1.2 million parameters, even a small number of violations result in unstable training: 1 (0.0001% parameters) for EWC, 4 (0.0003% parameters) for MAS, 3 (0.0004% parameters) for SI, and 6 (0.0005% parameters) for RWalk.

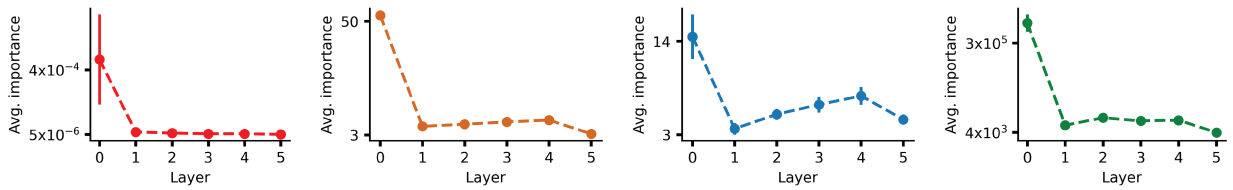
E.3 Disparate Importance Assignment

Due to use of biased importance definitions, parameters in deeper layers are often assigned much lower importance than parameters in earlier layers Section 3.1.2 of main paper). Thus, even if a valid training configuration (i.e., one that results in stable training) is used, the use of biased importance scores is unable to stop deeper layers from adapting to recent tasks. Since forgetting of previously learned tasks is majorly caused by changes in deeper layers, this disparate importance assignment renders quadratic regularizers ineffective. In Figure 3 (CIFAR-100), Figure 8 (Oxford-Flowers), and Figure 9 (Caltech-256), we show this disparate importance assignment is observed for all datasets considered in this work.



(a) EWC. (b) MAS. (c) SI. (d) RWalk.

Figure 8: Oxford-Flowers: Average importance of parameters in a layer.



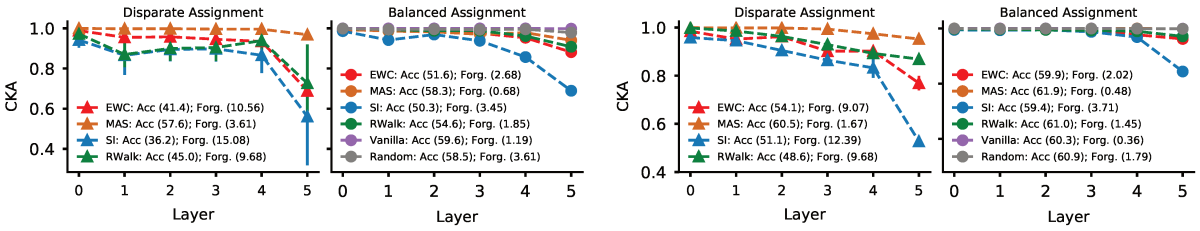
(a) EWC. (b) MAS. (c) SI. (d) RWalk.

Figure 9: Caltech-256: Average importance of parameters in a layer.

Balanced Importance Scores Prevent Forgetting More Effectively: In the main paper, we propose two regularizers that assign either unit importance score to all parameters (called Vanilla) or use uniformly picked, random importance score for all parameters (called Random). By treating all layers on a similar importance scale, we show these regularizers address problems pertaining to biased importance assignment in popular regularizes (see Figure 4 of main paper). In particular, following Ramasesh et al. (2021), we use CKA (Kornblith et al. (2019)) to measure representational similarity and estimate the contribution of a layer towards catastrophic forgetting. We find features across all layers of the model trained only on first task of a dataset and the model trained on all tasks of that dataset show high representational similarity for both Vanilla and Random. Here, we demonstrate this conclusion holds well for other datasets as well (see Figure 10).

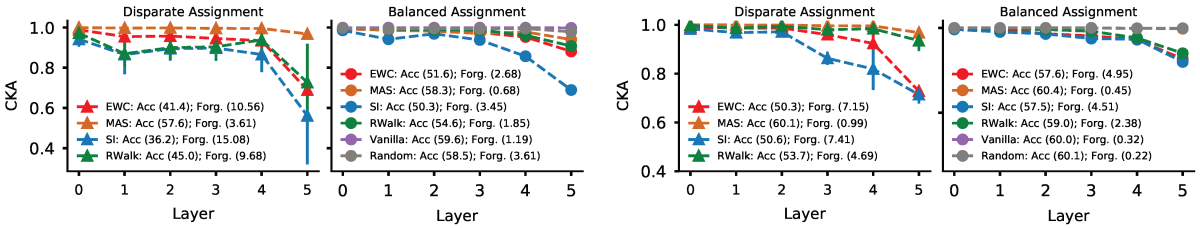
Table 4: Hyperparamaters found using grid search for different methods and datasets.

Dataset	Plain Quadratic Regularizers				Explicit Interpolation Variants			
CIFAR	EWC	MAS	SI	RWalk	EWC	MAS	SI	RWalk
LR (η)	0.003	0.003	0.003	0.003	0.01	0.01	0.01	0.01
Reg (λ)	10^4	10^0	10^{-4}	10^{-5}	-	-	-	-
Flowers	EWC	MAS	SI	RWalk	EWC	MAS	SI	RWalk
LR (η)	0.01	0.01	0.01	0.01	0.03	0.03	0.03	0.03
Reg (λ)	10^3	10^0	10^{-2}	10^{-5}	-	-	-	-
Cal-256	EWC	MAS	SI	RWalk	EWC	MAS	SI	RWalk
LR (η)	0.01	0.01	0.01	0.01	0.03	0.03	0.03	0.03
Reg (λ)	10^3	10^0	10^{-2}	10^{-5}	-	-	-	-



(a) Oxford-Flowers (1 epoch)

(b) Oxford-Flowers (30 epochs)



(c) Caltech-256 (1 epoch)

(d) Caltech-256 (30 epochs)

Figure 10: Representational similarity, as measured using CKA, between a model trained only on the first task of a dataset versus the model trained on all tasks of that dataset. Average accuracy (Acc.) and average forgetting (Forg.) are reported in the legend. Unlike other methods, Vanilla and Random are not biased against deeper layers, achieving high representational similarity across all layers with minimal forgetting. Meanwhile, the other measures suffer significant drift in similarity for deeper layers and, hence, more forgetting.

F More Results on Experiments from Section 4.1

F.1 Quadratic Regularization on ResNet-18

In the main paper, we only provide results comparing quadratic regularization variants on a 6-layer CNN. Here, we give comparisons for a ResNet-18 model. Following prior works, we use one-third the number of filters in each layer (Mirzadeh et al. (2021)). Hyperparameters are again determined via a grid-search on first 3 tasks and listed in Table 4. The hyperparameter search space and other training configurations are the same as before (discussed in Appendix D). The final results are provided in Table 6. As can be seen, the conclusions drawn on 6-layer CNN hold well with the ResNet-18 model too. We note the results for plain fine-tuning can sometimes be better than EWC. This happens because grid-search leads to a small learning rate for plain fine-tuning, but a relatively larger learning rate for EWC.

Table 5: **Detailed version of Table 1 from main paper**: Comparison of Average Accuracy (Acc; ↑ indicates higher is better) and Average Forgetting (Forg; ↓ indicates lower is better) for plain and explicit interpolation variants EWC, MAS, SI, and RWalk. We consider three datasets: CIFAR-100 (10 tasks); Oxford-Flowers (17 tasks); and Caltech-256 (32 tasks). For a specific regularizer, the better performing variant is in bold. As shown, variants with explicit interpolation steps consistently outperform their Quadratic Regularization counterparts. This behavior is most prominent in datasets with unbalanced classes and longer task sequences (Oxford-Flowers and Caltech-256), where hyperparameter tuning is difficult.

1 epoch		Plain Quadratic Regularizers				Explicit Interpolation Variants			
CIFAR	Plain	EWC	MAS	SI	RWalk	EWC	MAS	SI	RWalk
Acc (↑)	55.3 ± 1.6	62.5 ± 0.6	63.9 ± 0.8	61.2 ± 2.3	**63.9 ± 0.3**	**63.8 ± 0.4**	**64.0 ± 0.2**	**63.9 ± 0.3**	63.8 ± 0.9
Forg (↓)	9.13 ± 1.6	2.07 ± 1.0	0.56 ± 0.2	3.20 ± 0.3	**0.54 ± 0.1**	**0.08 ± 0.02**	**0.06 ± 0.05**	**0.07 ± 0.04**	**0.09 ± 0.03**
Flowers	Plain	EWC	MAS	SI	RWalk	EWC	MAS	SI	RWalk
Acc (↑)	37.9 ± 2.1	41.3 ± 0.5	57.5 ± 2.4	36.2 ± 2.3	44.9 ± 2.1	**60.0 ± 0.8**	**59.6 ± 1.1**	**59.9 ± 0.9**	**59.4 ± 1.1**
Forg (↓)	13.4 ± 2.5	10.6 ± 1.0	3.61 ± 0.7	15.1 ± 1.9	9.68 ± 2.3	**1.03 ± 0.5**	**0.44 ± 0.2**	**0.99 ± 0.2**	**1.31 ± 0.3**
Cal-256	Plain	EWC	MAS	SI	RWalk	EWC	MAS	SI	RWalk
Acc (↑)	40.1 ± 0.9	41.9 ± 0.5	53.7 ± 0.5	40.9 ± 0.5	43.7 ± 0.7	**56.2 ± 0.3**	**57.3 ± 0.6**	**56.0 ± 0.6**	**55.8 ± 0.5**
Forg (↓)	6.21 ± 0.5	5.59 ± 0.1	3.92 ± 0.4	5.68 ± 0.1	4.69 ± 0.1	**1.41 ± 0.2**	**0.36 ± 0.1**	**1.42 ± 0.2**	**1.31 ± 0.1**
30 epochs		Plain Quadratic Regularizers				Explicit Interpolation Variants			
CIFAR	Plain	EWC	MAS	SI	RWalk	EWC	MAS	SI	RWalk
Acc (↑)	60.7 ± 1.4	63.6 ± 0.8	66.3 ± 0.2	62.8 ± 1.6	64.7 ± 0.9	**66.3 ± 0.3**	**66.0 ± 0.2**	**66.2 ± 0.3**	**66.1 ± 0.3**
Forg (↓)	8.01 ± 1.2	4.63 ± 1.0	0.23 ± 0.07	6.30 ± 0.7	3.39 ± 1.1	**0.21 ± 0.06**	**0.23 ± 0.07**	**0.25 ± 0.04**	**0.29 ± 0.07**
Flowers	Plain	EWC	MAS	SI	RWalk	EWC	MAS	SI	RWalk
Acc (↑)	49.9 ± 1.7	54.1 ± 0.9	60.5 ± 0.6	51.1 ± 1.4	55.7 ± 1.5	**61.3 ± 0.7**	**61.6 ± 0.6**	**61.8 ± 0.7**	**61.7 ± 0.8**
Forg (↓)	11.8 ± 1.9	9.07 ± 0.8	1.19 ± 0.7	12.4 ± 2.0	7.19 ± 1.3	**0.21 ± 0.4**	**0.48 ± 0.5**	**0.62 ± 0.7**	**0.71 ± 0.5**
Cal-256	Plain	EWC	MAS	SI	RWalk	EWC	MAS	SI	RWalk
Acc (↑)	40.6 ± 1.2	50.3 ± 0.9	60.1 ± 0.4	50.6 ± 0.8	53.7 ± 0.8	**60.6 ± 1.0**	**60.9 ± 0.8**	**60.8 ± 0.8**	**60.6 ± 0.7**
Forg (↓)	15.8 ± 1.5	7.15 ± 0.8	0.99 ± 0.4	7.41 ± 1.1	4.69 ± 0.6	**0.22 ± 0.6**	**0.18 ± 0.4**	**0.32 ± 0.6**	**0.34 ± 0.6**

Table 6: **Results on ResNet-18**: Comparison of Average Accuracy (Acc; ↑ indicates higher is better) and Average Forgetting (Forg; ↓ indicates lower is better) for plain and explicit interpolation variants EWC, MAS, SI, and RWalk. We consider three datasets: CIFAR-100 (10 tasks); Oxford-Flowers (17 tasks); and Caltech-256 (32 tasks). For a specific regularizer, the better performing variant is in bold. As shown, variants with explicit interpolation steps consistently outperform their Quadratic Regularization counterparts. This behavior is most prominent in datasets with unbalanced classes and longer task sequences (Oxford-Flowers and Caltech-256), where hyperparameter tuning is difficult.

Dataset		Plain Quadratic Regularizers				Explicit Interpolation Variants			
CIFAR	Plain	EWC	MAS	SI	RWalk	EWC	MAS	SI	RWalk
Acc (↑)	40.1 ± 0.6	39.6 ± 1.4	**58.5 ± 0.5**	30.5 ± 1.8	**58.6 ± 0.3**	**57.6 ± 0.4**	57.4 ± 0.3	**57.2 ± 0.6**	57.3 ± 0.5
Forg (↓)	19.3 ± 0.2	33.2 ± 4.9	**1.7 ± 0.4**	39.4 ± 1.3	4.2 ± 0.2	**1.7 ± 0.3**	1.7 ± 0.6	**1.9 ± 0.3**	**1.6 ± 0.2**
Flowers	Plain	EWC	MAS	SI	RWalk	EWC	MAS	SI	RWalk
Acc (↑)	32.1 ± 1.7	27.3 ± 3.4	41.4 ± 1.5	24.6 ± 1.6	41.6 ± 0.6	**53.4 ± 1.4**	**52.6 ± 1.8**	**52.5 ± 1.8**	**53.1 ± 1.7**
Forg (↓)	9.6 ± 1.3	32.1 ± 3.4	16.9 ± 1.3	34.7 ± 1.5	15.4 ± 1.0	**7.1 ± 1.2**	**7.9 ± 1.2**	**5.1 ± 0.9**	**8.4 ± 1.8**
Cal-256	Plain	EWC	MAS	SI	RWalk	EWC	MAS	SI	RWalk
Acc (↑)	35.7 ± 1.1	24.6 ± 0.9	46.4 ± 1.4	22.9 ± 0.5	56.8 ± 2.6	**52.7 ± 1.5**	**52.9 ± 1.3**	**50.0 ± 1.4**	**52.8 ± 1.7**
Forg (↓)	13.7 ± 1.5	32.7 ± 1.2	12.9 ± 1.6	31.7 ± 1.0	12.1 ± 2.3	**3.47 ± 0.3**	**3.8 ± 0.3**	**3.2 ± 0.7**	**3.8 ± 0.7**

F.2 Quadratic Regularization on 6-layer CNN

In the main paper, we provided results comparing plain quadratic regularizers with their explicit interpolation variants. Due to space constraints, we did not provide standard deviations. This information, along with the average results, is shown in Table 5.