Optimizer Mismatch Drives Catastrophic Forgetting

Celine Tan
UC Berkeley
celinetan@berkeley.edu

Katie Wang
UC Berkeley
katiewang@berkeley.edu

Swetha Rajkumar
UC Berkeley
swetha.rajkumar@berkeley.edu

E Harrison
UC Berkeley
ehharrison@berkeley.edu

Abstract

A major challenge in the continual learning setting is catastrophic forgetting, which occurs when training a model on a new task causes a rapid deterioration in its performance on previously learned tasks. We analyze how the combination of different optimizers during pre-training and fine-tuning affects catastrophic forgetting, and how to best mitigate it. We argue that due to the distinct forms of implicit regularization that different optimizers impose on weight updates, combining different optimizers can lead to an increase in forgetfulness on previous tasks. To support this claim, we train AlexNet [10] and ViT-B/16 [4] and measure model accuracy on the multi-class datasets ImageNet, MNIST, CIFAR-10, and Omniglot. We first utilize common continual learning techniques to achieve a baseline performance for these tasks, and observe that matching hyperparamters during pre-training and fine-tuning consistently outperforms fine-tuning with mismatching optimizers. We further justify these findings by measuring the change in weight norms over training, and give a geometric explanation for the drop in performance. All code can be found here.

1 Introduction

Continual learning (CL) is a class of machine-learning problems where a model learns from a sequence of tasks over time without restarting training from scratch. The goal is to accumulate knowledge from each new task and adapt to new information, while remaining robust as the environment or task distribution changes. CL can also be considered as a generalized form of fine-tuning, wherein a model trained on an initial set of tasks is further trained to adapt to a new set of tasks. The initial training done on the first set of tasks is referred to as pre-training, while training on the new task is known as post-training/fine-tuning.

A major challenge in this setting is catastrophic forgetting [20], a phenomenon where learning new tasks unexpectedly overwrites or degrades performance on previously learned tasks. In general, this occurs because the gradient updates from new data can interfere with the model's learned representations for older tasks. Many continual learning methods and their effects have been investigated in prior works, such as constrained optimization algorithms ([23], [18], [5]), adding regularization to existing optimization methods ([24] [19], [17], [11]), and reinforcement learning [22].

Current research already exists that investigates the effects of optimizers such as Adam and SGD on catastrophic forgetting ([1], [6], [8]), and recent preliminary findings [17] suggest that optimizer mismatch (namely, AdamW→Muon and Muon→AdamW) during pre-training and post-training exerts a negative effect on model performance. This raises the question of whether choices of optimizer combinations affect the magnitude of a model's catastrophic forgetting from pre-training to fine-tuning. The answer to this question can be vital for a model's success—if the difference in optimizers does matter, then understanding an existing model's pre-training protocol is necessary to achieve optimal performance in fine-tuned tasks while maintaining robustness and pre-trained generalization abilities. In this report, we investigate this scenario by performing experiments over three task domains, each varying in difficulty, and examining how changing the optimizer from pre-training to fine-tuning affects the accuracy of our model in the pre-trained and fine-tuned data distributions.

Hypothesis: We come to the conclusion that, due to the distinct forms of implicit regularization found in popular optimization methods (e.g. Adam, SGD, and Muon), optimizer mismatch between pre-training and fine-tuning exacerbates catastrophic forgetting.

2 Background and Relevant Works

2.1 Problem Statement

Continual learning can be modeled given a parameterized network f_θ and a sequence of tasks $\mathcal{T}_1,\ \mathcal{T}_2,\ \ldots,\ \mathcal{T}_K$ where each task $\mathcal{T}_t$ has a data distribution of $p_t(x, y)$. For each task $\mathcal{T}_t$, we train the model to minimize the task-specific loss:

$$\theta_t = \arg\min_\theta \mathbb{E}_{(x,y)\sim p_t} \left[\mathcal{L}_t(f_\theta(x), y)\right] \tag{1}$$

where after learning task $\mathcal{T}_t$, we no longer can sample from p_t. Training starts with task $\mathcal{T}_1$ and continues in sequence to task $\mathcal{T}_K$, where we then measure the performance of the final network f_{θ_K} on the test data from all tasks. In some cases, we disregard the limitation that we can no longer sample from prior p_t's, which can be helpful in reducing forgetting [19].

Finetuning can also be represented with this paradigm. We start with an initial set of parameters θ_{pre} which was pre-trained on task $\mathcal{T}_{\text{pre}}$ with distribution p_{pre}. Then, we wish to update our parameters for a new task $\mathcal{T}_{\text{post}}$ with the distribution p_{post}.

In both general CL and fine-tuning, catastrophic forgetting can be an issue. Given a previous task i and current task t, the catastrophic forgetting of the model is defined as:

$$F_{i,t} = A_{i,i} - A_{i,t} \tag{2}$$

or the difference in accuracy on task i during training on i versus during training on task t. In the case for fine-tuning, we can rewrite this as:

$$F_{\text{pre,post}} = A_{\text{pre,pre}} - A_{\text{pre,post}} \tag{3}$$

Additionally, we can define the relative forgetting of the model to be:

$$R_{\text{pre,post}} = \frac{F_{\text{pre,post}}}{A_{\text{pre,pre}}} \tag{4}$$

We wish to investigate how using different optimizers during the pre-training and post-training phases affects $F_{\text{pre,post}}$ and $R_{\text{pre,post}}$. To do so effectively, we first utilize common continual learning strategies to achieve adequate robustness on our trained models.

2.2 Regularization Methods

While seen as a model "forgetting" its ability to perform well on previously-trained tasks, catastrophic forgetting can equivalently be seen as a form of overfitting on the new task, leaving the model unable to generalize to previous tasks. As such, usual regularization strategies which hope to reduce overfitting can be helpful in solving catastrophic forgetting. A common approach is to use weight decay in during training, which adds a regularization term to the model's loss function which encourages parameters to stay small:

$$\mathcal{L}^{\text{decay}}(\theta) = \mathcal{L}(\theta) + \lambda\|\theta\|_2^2 \tag{5}$$

In continual learning, a similar approach is weight anchoring (also known as L2-SP) [15], where parameters are encouraged to stay close to the previous task's weights rather than 0:

$$\mathcal{L}_t^{\text{anchor}}(\theta_t) = \mathcal{L}_t(\theta_t) + \lambda\|\theta_t - \theta_{t-1}\|_2^2 \tag{6}$$

Selective Projection Decay (SPD) [24] further adapts weight decay for the CL domain by selectively applying stronger decay to layers whose updates are misaligned with consistent loss reduction. SPD imposes regularization only when the update risks moving away from low loss regions, and further reduces the ability of underfitting due to decay. While SPD has shown to greatly improve model robustness, for the purposes of achieving just adequate performance on our tasks and ease of implementation, we opt to stick to weight decay and anchoring.

Another approach to regularization is pre-training data injection, also known as experience replay [19]. This method keeps a buffer of labels from previously learned tasks which is then used to modify the dataset that the model trains on during fine-tuning. There are many ways to combine this buffer into the target dataset, but the simplest approach is random sampling, where the modified dataloader samples from this buffer with some probability p. Figure 1 illustrates this process.

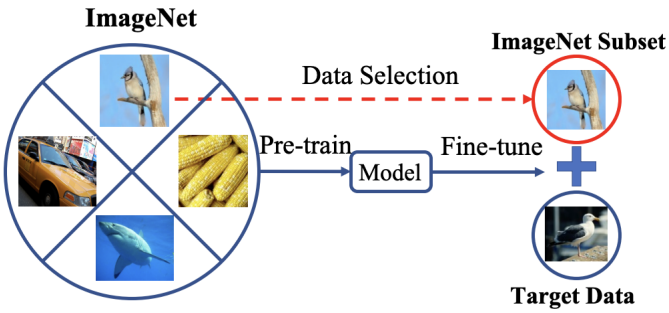


Figure 1: **Pre-training data injection on ImageNet.** A subset of the original pre-training data is used during the fine-tuning of the model alongside the target data. Multiple methods are possible to determine this pre-training data subset, but we opt to use random sampling in our experiments. Illustration from [19].

In our experiments, we find that only a small percent of the fine-tuning dataset needs to be from pre-training data—sampling from the buffer with a 1-5% probability was enough to avoid complete collapse of the pre-trained task, while still leaving room for improvement with other regularization methods.

Like weight anchoring, pre-training data injection can be used to keep our model's fine-tuned weights close to their pre-trained values. This is done through knowledge distillation [7], where

samples from the pre-trained dataset are passed through both the fine-tuned model and a frozen copy of the initial model, and a knowledge distillation loss weighted by α is added to the original loss.

For a formal explanation of knowledge distillation, we let $f_{\text{old}}(\cdot)$ denote the frozen *teacher model* trained on the pre-training task, $f(\cdot)$ denote the *student model* being trained on the fine-tuning task, and $\mathcal{C}_{\text{old}}$ be the set of initial task classes.

For an input image x from the pre-training data, the teacher and student produce logits:

$$\mathbf{z}^{\text{old}} = f_{\text{old}}(x), \quad \mathbf{z}^{\text{new}} = f(x)_{\mathcal{C}_{\text{old}}}, \tag{7}$$

where $f(x)_{\mathcal{C}_{\text{old}}}$ denotes the subset of student logits corresponding to the initial task classes.

The knowledge distillation loss is defined as:

$$\mathcal{L}_{\mathcal{KD}} = T^2 \,\mathrm{KL}\left(\mathrm{softmax}\left(\frac{\mathbf{z}^{\text{old}}}{T}\right) \,\middle\|\, \mathrm{softmax}\left(\frac{\mathbf{z}^{\text{new}}}{T}\right)\right), \tag{8}$$

where $\mathrm{KL}(\cdot\|\cdot)$ denotes the Kullback–Leibler divergence, and T is the distillation temperature. The T^2 scaling follows [7] and compensates for gradient attenuation at higher temperatures.

Finally, the total loss used in fine-tuning combines the cross-entropy loss on new task and replayed old task samples, $\mathcal{L}_{CE}$, with the distillation loss as in [16]:

$$\mathcal{L}_{\text{total}} = \mathcal{L}_{CE} + \alpha\, \mathcal{L}_{KD}, \tag{9}$$

where α is the weighting factor controlling the contribution of the distillation term.

Linear probing is another example of regularization, where instead of training the entire model for the new task, just a final linear layer is trained. While this removes the ability of forgetting previous tasks, it severely limits the expressivity of our model if its feature representations aren't generalizable to new tasks. Linear probing then fine-tuning (LP-FT) [11] is a popular modification of this approach that removes this limitation. LP-FT first trains a linear probe with the rest of the model frozen, then later unfreezes the rest of the model and completes a fine-tune. Figure 2 describes the difference between a full fine-tune, a linear probe, and LP-FT.

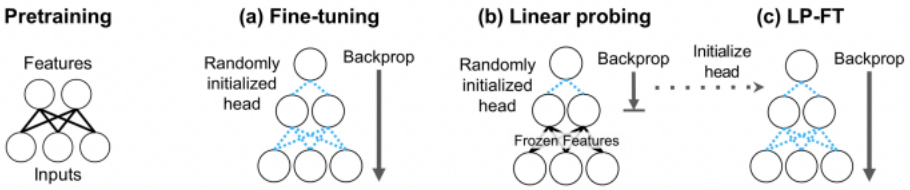


Figure 2: **Finetuning versus linear probing versus LP-FT.** LP-FT starts by performing a regular linear probe, then afterwards does a full fine-tune on the pre-trained model. By first initializing a good linear probe for fine-tuning, gradients that flow into the pre-trained model during fine-tuning are already well adjusted for the fine-tuned task. Illustration from [11].

2.3 Optimizers for Continual Learning

Many optimizers exist that aim to update a set of parameters θ such that they minimize an expected loss $\mathcal{L}$ over some data distribution. For deep learning, the most common types of optimizers involve gradient-based methods, which take the gradient or approximate gradient of the loss function with respect to the learnable network's parameters, and update these parameters based on its value. We investigate three main gradient-based optimizers in this report: SGD, Adam, and Muon [17].

For continual learning, proposed modifications to these optimizers exist which constrain network updates to reduce catastrophic forgetting. Sharpness-Aware Minimization (SAM) [5], for example,

modifies the gradient of the original loss function, approximating the maximum training loss in a small neighborhood of parameter space, and enabling the model to seek flatter regions of the loss landscape rather than sharp regions with many local minima. Since SAM is used before applying gradients for optimization, it is compatible with standard gradient-based optimizers like SGD, Adam, and Muon.

Other works like [12] and [18] try to directly improve individual optimizers. [12] investigates improving vision model robustness while fine-tuning with SGD, and highlights that freezing the embedding layer of these models can allow SGD to outperform AdamW for pre-training retention after fine-tuning. REG [18] instead tries to improve Muon, specifically by resolving Muon's training instabilities when fine-tuning from AdamW-pre-trained models. It does so by replacing Muon's aggressive matrix sign operator with a Row-and-Column-Scaling operator, outperforming Muon in SFT of Qwen2.5-Math-1.5B, but slightly underperforming AdamW on ResNet-50.

Novel optimizers also exist that are designed to overcome catastrophic forgetting. Direction-Constrained Optimization (DCO) [23] identifies that for each task, there exists a cone in the network's parameter space where model performance remains close to the recovered optimum. DCO identifies the top principles directions of this cone, and restrict future updates of subsequent tasks to remain along these directions. Doing this for all tasks reduces the likelihood of forgetting each one.

For the purposes of this report, we choose to stick to SGD, Adam, and Muon for a few reasons. DCO requires the model to be trained with the same optimizer over all tasks, making it infeasible to transfer optimizers between tasks. We also want to emulate a "plug-and-play" approach to the pre-train-fine-tune pipeline, which means utilizing prebuilt optimizers that are already implemented in popular deep learning libraries like PyTorch. SAM and REG are not yet readily available in this way, making it difficult to quickly investigate the effect of changing optimizers when using these methods. Lastly, SGD, Adam, and Muon already have deep theoretical backgrounds that make them much easier to analyze when considering how and why they affect catastrophic forgetting.

2.4 Optimization Theory

A key insight developed by [2] is that gradient-based optimizers can be modeled as trying to solve the same constrained optimization question, just over different norms. This works as a form of implicit regularization by the optimizer and makes updates to optimal regions in parameter space dependent on this norm.

To illustrate this, consider the approximate change of our loss function after some parameter update with its first-order Taylor expansion:

$$\mathcal{L}(\theta + \Delta\theta) \approx \mathcal{L}(\theta) + \langle \nabla_\theta \mathcal{L}(\theta), \Delta\theta \rangle \tag{10}$$

When deciding on an optimizer to choose, a good optimizer would be one that minimizes our loss as much as possible with a restricted step size in our parameters (to maintain the Taylor approximation):

$$\Delta\theta^* = \arg \min_{\|\Delta\theta\| \leq \eta} \mathcal{L}(\theta + \Delta\theta) \tag{11}$$

$$\Delta\theta^* = \arg \min_{\|\Delta\theta\| \leq \eta} \mathcal{L}(\theta) + \langle \nabla_\theta \mathcal{L}(\theta), \Delta\theta \rangle \tag{12}$$

$$\Delta\theta^* = \arg \min_{\|\Delta\theta\| \leq \eta} \langle \nabla_\theta \mathcal{L}(\theta), \Delta\theta \rangle \tag{13}$$

Where η is an adjustable hyperparameter. An important observation made by [2] is that common optimizers such as SGD and Adam can be derived by changing the norm that this constraint cares about. The standard ℓ_2-norm reduces to a scaled version of gradient descent, which can be easily adapted to SGD if we approximate our gradient using batches. Using the ℓ_∞-norm instead reduces to sign-GD, which closely resembles Adam if momentum is added when calculating the sign of our gradient. Muon was derived with the understanding of this theory, opting to use the RMS$\to$RMS induced norm for matrices.

We argue that this fundamental difference in the construction of SGD, Adam, and Muon is a core factor that increases catastrophic forgetting when switching optimizers between pre-training and fine-tuning. When parameter updates are constrained by a particular norm, the resulting parameters live on distinct surface that minimizes the norm of the loss gradient for the given task, akin to

the cone that parameters reside on in DCO. By later fine-tuning with a different optimizer and thus different norm constraint, the parameters no longer reside on this surface, leading to a sudden collapse of the performance in the pre-trained task.

3 Research Questions

We investigate a number of research questions with this paper, including the following:

RQ1: How effective are various combinations of optimizers for mitigating forgetting when fine-tuning? What combination of optimizers leads to the best performance?

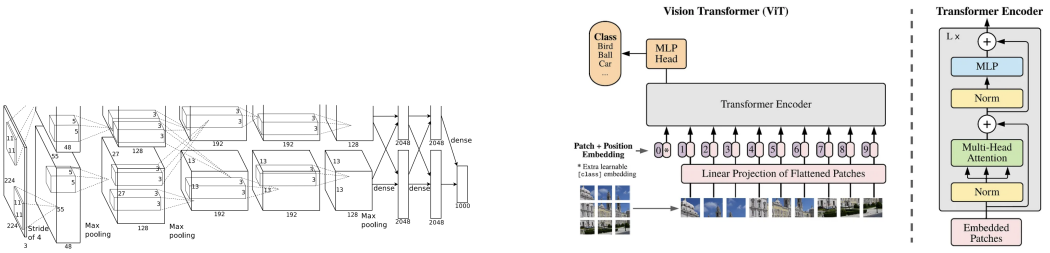
RQ2: Do these performance observations persist across task difficulties and dataset sizes?

RQ3: Are there methods of fine-tuning and regularization that prove most effective when retaining pre-trained knowledge?

4 Hypothesis

Different optimizers impose distinct forms of implicit regularization and bias the model toward specific regions of parameter space with different geometrical properties. For instance, Muon converges to solutions that help minimize an induced RMS→RMS norm of the loss gradients, while Adam is designed to converge to solutions minimizing ℓ_∞ norm, meaning that model weights trained by these two optimizers will tend to different optimal regions in the parameter space. When fine-tuning begins, the optimizer starts in a region of parameter space that might not align with the norm it aims to minimize. If the fine-tuning optimizer's implicit regularization differs substantially from that of the pre-training optimizer, their update directions conflict: the fine-tuning dynamics "regularize away" information encoded by the pre-training phase. This regularization mismatch can manifest as unstable loss trajectories, degraded transfer accuracy, or catastrophic forgetting. Conversely, using optimizers with compatible implicit regularization (e.g., pre-trained on Muon then fine-tuned on Muon or pre-trained on SGD then fine-tuned on SGD) helps maintain alignment with the pre-trained manifold, stabilizing adaptation.

5 Methods



(a) AlexNet Architecture

(b) ViT Architecture

Figure 3: **Model Architectures.** For each task, we modify the last linear layer of AlexNet and the MLP head of the ViT to perform classification over different number of classes. AlexNet illustration from [10] and ViT illustration from [4]

To specifically investigate catastrophic forgetting, we perform experiments with a variety of vision-based tasks. We use AlexNet and a simple Vision Transformer, as these architectures have been investigated extensively in prior works ([16] [21]). We investigate how our models' accuracy rates on both our pre-training and fine-tuning datasets evolve over time, and compare the magnitudes of catastrophic forgetting for different combinations of optimizers. We primarily focus our study on Muon, AdamW, and SGD with momentum.

Additionally, to investigate the effects of the varying implicit regularization and update geometries of our chosen optimizers, we examine the ℓ_∞, ℓ_2, and RMS norms of the final classifier heads of our networks.

To investigate whether this effect is pervasive across fine-tuning methods, we additionally perform experiments using a variety of knowledge retention methods.

5.1 Transfer Learning from ImageNet to MNIST with AlexNet

Building upon previous work [16] investigating various methods of learning without forgetting, we first pre-train AlexNet [10] on ImageNet [3] and fine-tune on MNIST [14] by first detaching our pre-trained ImageNet head, then adding a second, randomly initialized output layer for MNIST.

To evaluate the efficacy of various optimizer matchups, we pre-train three variants of our network with three different optimizers: SGD with momentum, AdamW, and Muon. Following the original training recipe for AlexNet[10], our SGD with momentum pre-trained variant uses a step rate decay. To facilitate a fairer comparison between optimizer choices, we perform a grid search for optimal learning rate schedules for AdamW and Muon; further details are described in A. We then fine-tune each variant with each optimizer, and periodically evaluate the accuracy of the fine-tuned network on validation sets from both ImageNet and MNIST. Similar to [16], we lower the learning rate from pre-training to fine-tuning ($0.1\times$ the pre-training rate) to preserve original task performance.

5.2 Transfer Learning from ImageNet to MNIST with ViT

To complement our experiments with AlexNet, we additionally examine catastrophic forgetting using a different model architecture: a Vision Transformer (ViT-B/16) [4] pre-trained on ImageNet using the AdamW optimizer. We employ the official torchvision implementation with ImageNet-1K pre-trained weights. The goal of this experiment is to assess whether catastrophic forgetting persists under different model architectures.

The original 1000-class classification head of the pre-trained ViT is first stored for later re-evaluation on ImageNet. For MNIST fine-tuning, we replace this head with a newly initialized linear classifier mapping the 768-dimensional final hidden representation to ten output classes. All pre-trained transformer layers remain unchanged in structure and are fine-tuned jointly with the new classifier.

Before any fine-tuning, we evaluate the pre-trained model on the ImageNet validation set to obtain baseline accuracy. After swapping in the MNIST classifier, we fine-tune the model on MNIST and then restore the original ImageNet head to measure post-training performance. The difference between these two metrics quantifies the degree of catastrophic forgetting.

Because ViT-B/16 is significantly larger and more computationally demanding than AlexNet, we perform training using a fixed number of optimization steps rather than full epochs. This step-based scheme makes the runs feasible on available hardware. All hyperparameters are provided in Appendix A. To account for variability, we repeat the full procedure across multiple random seeds.

5.3 Continual Learning with CIFAR-10

We investigate catastrophic forgetting using the CIFAR-10 dataset [9] divided into sequential tasks. Our primary objectives are to quantify the extent of catastrophic forgetting in a baseline continual learning scenario, evaluate experience replay as a mitigation strategy, compare different optimization approaches across training phases, and assess the impact of linear probing and distillation on knowledge retention. By systematically studying these factors, we aim to identify effective strategies for building continual learning systems that can balance stability (retaining old knowledge) with plasticity (acquiring new knowledge).

We use the CIFAR-10 dataset, which consists of 60,000 32×32 color images distributed across 10 classes. We divided these classes into two sequential tasks to simulate a continual learning scenario. Task 1, representing the initial learning phase, consists of classes 0 through 4. Task 2, representing the continual learning phase, introduces classes 5 through 9. This division creates a realistic scenario where the model must learn to recognize new categories without forgetting previously learned ones. The choice to split classes evenly between tasks ensures that neither task is trivially easy or over-

whelmingly difficult, and the semantic diversity within each task prevents the model from exploiting simple shortcuts.

For our model architecture, we employ AlexNet, a well-understood convolutional network. We first pre-train the network on a subset of CIFAR-10 classes as described previously. In practice, AlexNet converges rapidly, requiring only 10 epochs to reach a regime with minimal further optimization. During fine-tuning, we expand the classifier head to accommodate new classes while preserving the weights corresponding to the initial classes by copying them into the appropriate positions of the expanded layer. This weight-preservation strategy minimizes disruption to representations learned during pre-training and provides a stable foundation upon which experience replay and other mitigation mechanisms can operate.

Without any mitigation strategies, naive fine-tuning on the new classes would typically result in severe catastrophic forgetting, with the model's accuracy on pre-training classes dropping nearly to zero. To address this, we employ multiple regularization strategies, including linear probing, knowlegde distillation, and pre-training data injection, and compare optimizer configurations before and after implementing these strategies.

5.4 Continual Learning with Omniglot

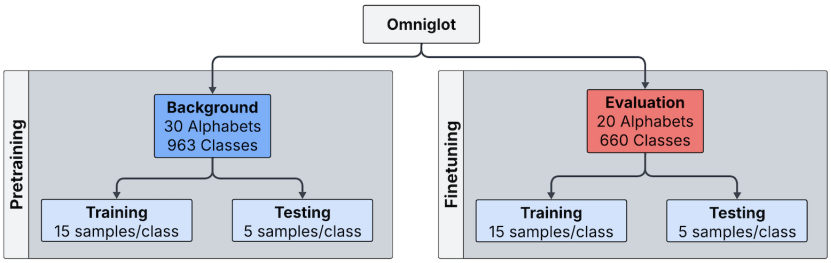


Figure 4: **Omniglot Data Organization.** For clarity, we refer to the background and evaluation datasets as pre-training and fine-tuning datasets respectively.

To observe the level of catastrophic forgetting in a few-shot environment, we train AlexNet on the Omniglot dataset [13].

Omniglot consists of 1623 different characters from 50 different writing systems, where each character has 20 samples. Omniglot is split into two datasets, background and evaluation, which we us as the pre-training and fine-tuning datasets respectively. Similar to MNIST, Omniglot's characters are grayscale images, however Omniglot can be a difficult task due to the lack of samples per class (only 15 examples), and the large number of classes. An illustration of Omniglot organization is shown in Figure 4.

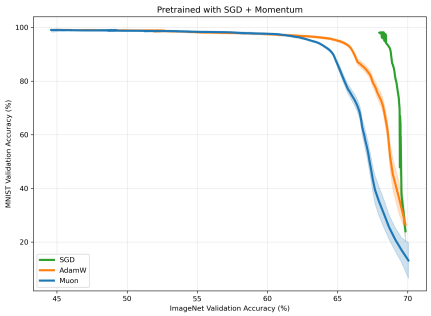
We perform pre-training for 90 epochs on AlexNet with two optimizers, AdamW and Muon. For each optimizer, we perform a grid search for the optimal learning rate and weight decay. Further details on optimal values can be found in Appendix D. We then fine-tune each of our pre-trained AlexNets for another 90 epochs with Muon and Adam, and perform a grid search over learning rates. We repeat this fine-tuning over different amounts of pre-training data injection, sampling from the pre-training data randomly with 1% and 5% probability.

To determine forgetting, we measure the pre-training dataset accuracy prior to fine-tuning and after fine-tuning, then take the difference of these accuracies. For further interpretability, we plot the ℓ_∞ and RMS norm of the weights and weight updates over training epochs.

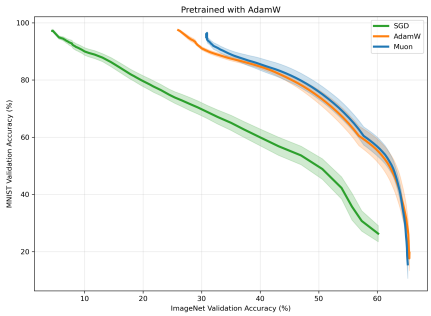
6 Results

6.1 Transfer Learning from ImageNet to MNIST with AlexNet

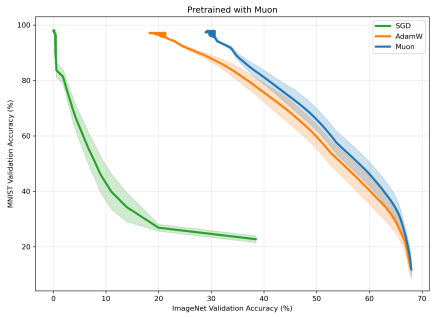
Pictured in 5 are the validation accuracies of the fine-tuned networks on both ImageNet (the pre-training dataset) and MNIST (the fine-tuning dataset). We evaluate our models on a subset of both



(a) SGD Pre-training



(b) AdamW Pre-training



(c) Muon Pre-training

Figure 5: **ImageNet with AlexNet.** MNIST Validation Accuracy vs. ImageNet Validation Accuracy. Shaded error gradients depict σ of runs over 3 different seeds.

	$A_{\text{pre,pre}}$	$A_{\text{post,post}}$	$A_{\text{pre,post}}$	$F_{\text{pre,post}}$
AdamW→AdamW	50.8%	$\mathbf{97.5\% \pm 0.1\%}$	$14.8\% \pm 4.3\%$	$36.0\% \pm 4.3\%$
AdamW→Muon	50.8%	$97.4\% \pm 0.1\%$	$\mathbf{16.5\% \pm 3.8\%}$	$\mathbf{34.3\% \pm 3.8\%}$
AdamW→SGD	50.8%	$97.0\% \pm 0.3\%$	$3.62\% \pm 0.61\%$	$47.18\% \pm 0.61\%$
Muon→AdamW	50.4%	$96.7\% \pm 0.0\%$	$11.1\% \pm 1.2\%$	$39.3\% \pm 1.2\%$
Muon→Muon	50.4%	$\mathbf{97.4\% \pm 0.2\%}$	$\mathbf{15.3\% \pm 1.5\%}$	$\mathbf{35.1\% \pm 1.5\%}$
Muon→SGD	50.4%	$97.1\% \pm 0.4\%$	$0.27\% \pm 0.04\%$	$50.13\% \pm 0.04\%$
SGD→AdamW	$\mathbf{53.1\%}$	$97.8\% \pm 0.1\%$	$38.2\% \pm 1.4\%$	$14.9\% \pm 1.4\%$
SGD→Muon	53.1%	$\mathbf{97.8\% \pm 0.2\%}$	$40.3\% \pm 0.9\%$	$12.8\% \pm 0.9\%$
SGD→SGD	53.1%	$96.9\% \pm 0.1\%$	$\mathbf{50.1\% \pm 0.2\%}$	$\mathbf{3.0\% \pm 0.2\%}$

Table 1: **AlexNet ImageNet→MNIST:** Pre-training to Finetuning Accuracies after 1 epoch on MNIST. $\text{Opt}_1 \rightarrow \text{Opt}_2$ refers to pre-training with Opt 1 and fine-tuning with Opt 2. Data is in the form of $\mu \pm 1\sigma$ over 3 different seeds. Bold entries indicate the best performance for each column, and for each pretraining optimizer. (For brevity, SGD + momentum is referred to as simply "SGD")

validation sets in intervals of 5 optimizer steps, due to the model's fast convergence on MNIST. As noted, green represents fine-tuning with SGD with momentum, blue represents fine-tuning with Muon, and orange represents fine-tuning with AdamW. We additionally evaluate our model on the full validation sets for both datasets after 1 full epoch on MNIST and record results in Table 1 and Figure 6. We observe a clear advantage in mitigating catastrophic forgetting when using SGD with momentum to fine-tune a network pre-trained with SGD with momentum, as well as a slight advantage when using Muon to fine-tune a network pre-trained with Muon. We also observe a slight advantage when using Muon when fine-tuning a network pre-trained with AdamW (within our error threshold). We posit that this may in part be due to how we use AdamW to train our non-matrix

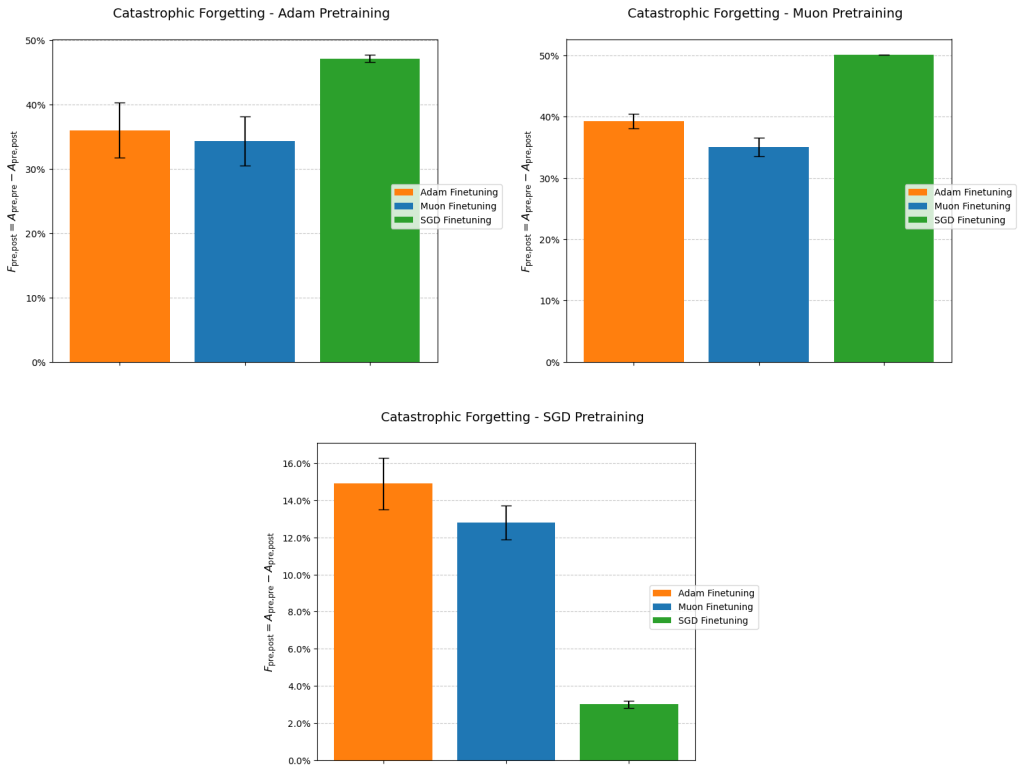


Figure 6: **ImageNet → MNIST with AlexNet:** Catastrophic Forgetting. For each pre-training, we measure the amount of catastrophic forgetting with AdamW, Muon, and SGD + momentum fine-tuning. Bars indicate $\mu \pm 1\sigma$ over 3 different seeds.

layers during Muon pre-training, thus allowing us to preserve a portion of the implicit regularization from AdamW that benefits the subsequent Muon fine-tuning phase. This partial optimizer overlap may reduce the severity of the geometric mismatch discussed in our hypothesis, resulting in more stable retention of pre-trained features compared to fully mismatched optimizer combinations.

6.2 Transfer Learning from ImageNet to MNIST with ViT

As shown in Figure 7, the plots compare the pre-trained validation accuracy on ImageNet with the fine-tuned validation accuracy on MNIST for the ViT model using three different optimizers for fine-tuning: AdamW, SGD with momentum, and Muon. These plots were generated using a run with the seed set to 42. As noted earlier, the pre-trained ImageNet weights used for fine-tuning were trained with the AdamW optimizer, making AdamW the baseline for these comparisons. Across both plots, we observe that the MNIST validation accuracy is roughly similar for all three optimizers after 90 steps. However, there is a notable drop in pre-training validation accuracy for both Muon and SGD compared to AdamW, as the curve representing AdamW consistently remains above those of SGD with momentum and Muon. Muon demonstrates a particularly drastic loss, losing almost all pre-trained accuracy over 90 steps while still achieving MNIST validation accuracies comparable to AdamW. These results highlight that optimizer choice plays a critical role in mitigating forgetting.

Figure 8 demonstrates the amount of forgetting measured by the accuracy on the ImageNet validation set before and after fine-tuning, across all three optimizers. We plot both the mean score across four different seeds and the corresponding standard deviation as error bars. We observe that forgetting is lowest when using AdamW for fine-tuning. Both SGD with momentum and Muon show greater forgetting, with Muon exhibiting nearly complete loss of pre-trained knowledge after fine-tuning.

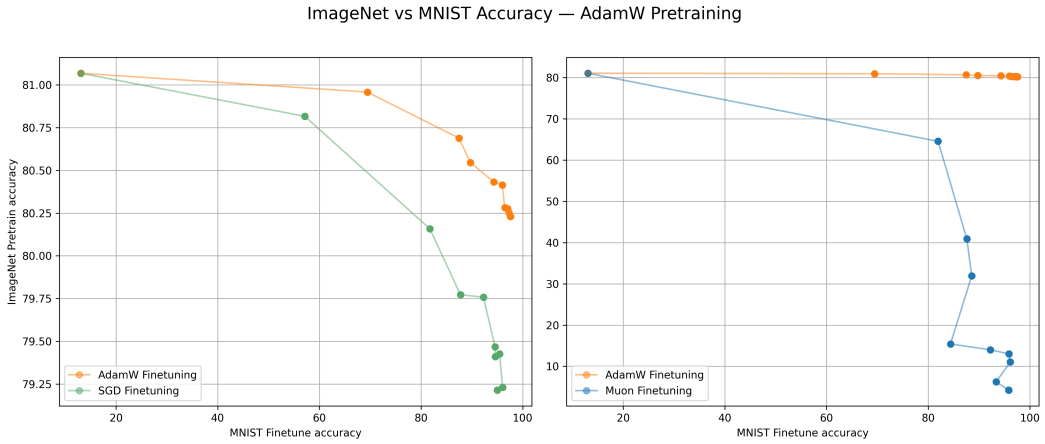


Figure 7: **ImageNet with ViT.** Pre-trained vs fine-tuned validation accuracy for ViT pre-trained on AdamW using different fine-tuning optimizers.

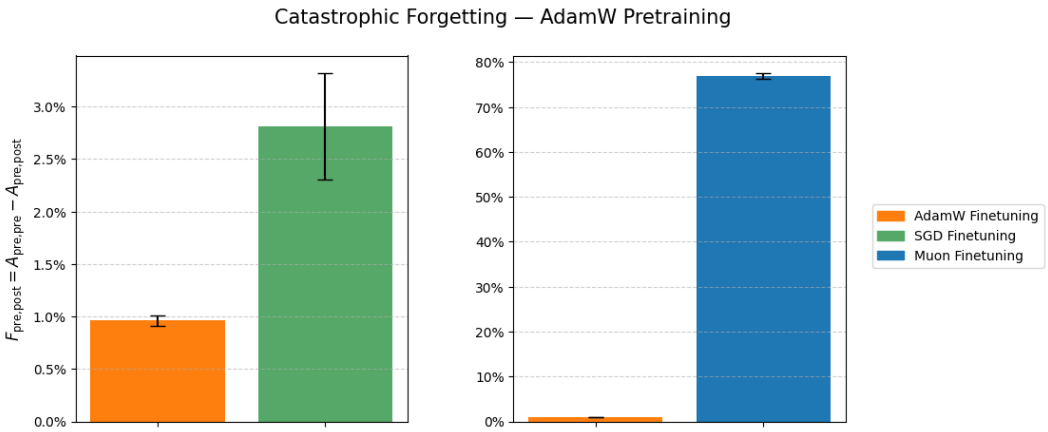


Figure 8: **ImageNet with ViT.** Catastrophic forgetting when using matching or mismatching optimizers. The plot on the left compares AdamW fine-tuning to SGD fine-tuning, while the plot on the right compares AdamW to Muon. All fine-tuning runs start from the same AdamW pre-trained checkpoint. Data is in the form of $\mu \pm 1\sigma$ over 4 different seeds.

6.3 Continual Learning with CIFAR-10

	$A_{\text{pre,post}}$	$A_{\text{post,post}}$	$R_{\text{pre,post}}$
SGD→SGD	$\mathbf{41.4\% \pm 0.62\%}$	$75.4\% \pm 1.3\%$	$\mathbf{51.2\% \pm 0.88\%}$
SGD→Adam	$33.6\% \pm 0.57\%$	$\mathbf{81.6\% \pm 0.53\%}$	$60.4\% \pm 0.51\%$
Adam→Adam	$\mathbf{31.6\% \pm 1.8\%}$	$77.1\% \pm 0.65\%$	$\mathbf{62.7\% \pm 0.45\%}$
Adam→SGD	$21.2\% \pm 0.76\%$	$\mathbf{78.0\% \pm 0.70\%}$	$74.9\% \pm 1.1\%$

Table 2: **CIFAR-10:** Pre-training set accuracy during finetuning, Finetuning set accuracy during finetuning, and Relative forgetting across optimizer combinations in the form of $\mu \pm 1\sigma$ over 3 different seeds. Bold entries indicate the best performance for each column, and for each pretraining optimizer.

Table 2 and figure 9 illustrate how the choice of optimizers during pre-training and fine-tuning interacts with catastrophic forgetting. When the model was pre-trained with SGD with momentum and fine-tuned with the same optimizer, pre-training task knowledge was largely retained. In contrast,

	Baseline	Knowledge Distillation	Linear Probe
SGD→SGD	**51.41%**	**41.93%**	**32.50%**
SGD→Adam	61.05%	53.66%	47.78%
Adam→Adam	63.34%	49.04%	44.00%
Adam→SGD	75.27%	54.59%	47.11%

Table 3: **CIFAR-10:** Relative forgetting of mitigation strategies under optimizer match and mismatch. Bold entries indicate the best performance for each column, and for each pretraining optimizer.

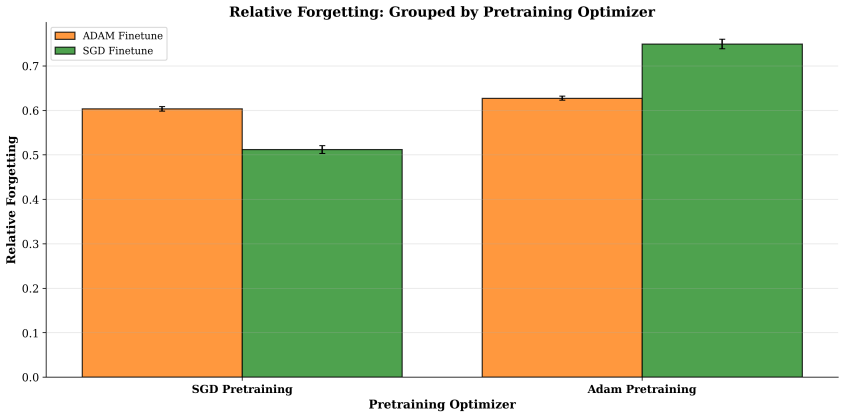


Figure 9: **CIFAR-10:** Error bars depicting relative forgetting across optimizer combinations of SGD with momentum and Adam.

switching to Adam for fine-tuning while keeping SGD with momentum for pre-training resulted in substantially higher forgetting, indicating that optimizer mismatch can disrupt the learned representations. The same trend holds for the other mismatched combination Adam→SGD with momentum when compared to Adam→Adam.

These results and figure 10 also highlight a tradeoff between pre-training retention and fine-tuning performance: mismatched optimizers may improve adaptation to the new task but at the cost of greater forgetting on the initial task.

We further evaluate mitigation strategies for catastrophic forgetting. Following [16], during fine-tuning, we employ the knowledge distillation loss to mitigate catastrophic forgetting on initial task classes. We also employ linear probing as a complementary mitigation strategy, which allows us to regularize the trained model and maintain features learned during pre-training. We compare the performance of different combinations of optimizers using these techniques with the baseline performance with a small injection of pre-training data in 3.

6.4 Continual Learning with Omniglot

The results of pre-training to fine-tuning can be found in Table 4. All results are from training with a pre-training data injection ratio of 5%, i.e, pre-training data was randomly selected in fine-tuning batches wtih 5% probability. Bold entries indicate the best performance of each metric.

Figure 11 shows catastrophic forgetting for varying percents of pre-training data injection and fine-tuning optimizer selection. For the sake of highlighting the relative difference of forgetting between Muon and Adam fine-tuning, we adjust the y-axis scales for each plot.

We also plotted weight norm and change in weight norm over training in Figure 12. Since Adam and Muon theoretically impose regularization for the ℓ_∞-norm and RMS-norm respectively, we plot these norms. For this experiment, we show results from AdamW and Muon pre-training with a pre-training data ratio of 5%, in figures 12 and 13 respectively.

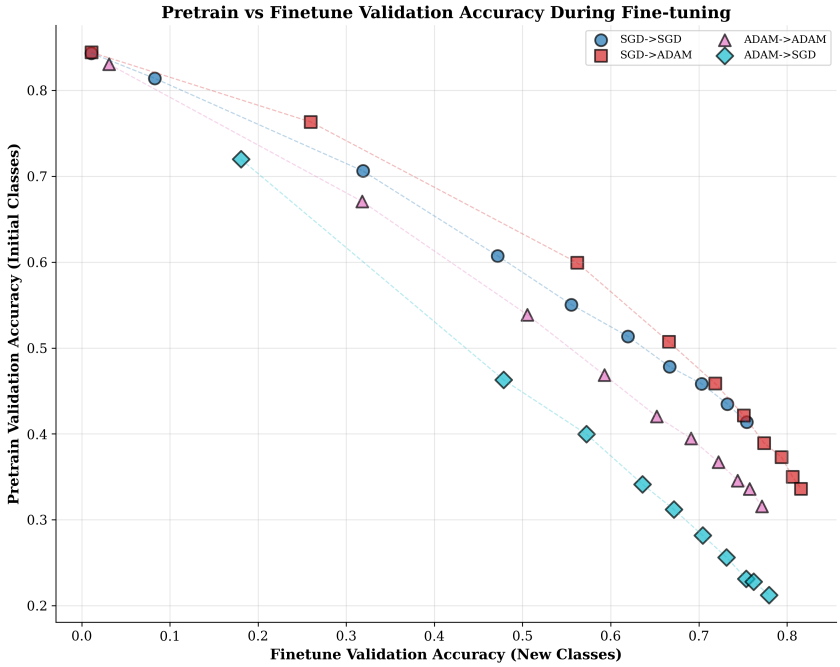


Figure 10: **CIFAR-10:** Pre-training task validation accuracy versus fine-tuning task validation accuracy for optimizer combinations.

	$A_{\text{pre,pre}}$	$A_{\text{post,post}}$	$A_{\text{pre,post}}$	$F_{\text{pre,post}}$
AdamW→Adam	$57.4\% \pm 1.7\%$	$\mathbf{67.5\% \pm 2.6\%}$	$\mathbf{53.6\% \pm 1.2\%}$	$\mathbf{3.8\% \pm 2.1\%}$
AdamW→Muon	$57.4\% \pm 1.7\%$	$66.3\% \pm 0.61\%$	$43.9\% \pm 0.37\%$	$13.5\% \pm 1.7\%$
Muon→Muon	$\mathbf{73.5\% \pm 0.61\%}$	$\mathbf{73.5\% \pm 0.17\%}$	$\mathbf{70.9\% \pm 0.47\%}$	$\mathbf{2.6\% \pm 0.77\%}$
Muon→Adam	$73.5\% \pm 0.61\%$	$71.5\% \pm 0.22\%$	$66.8\% \pm 0.13\%$	$6.7\% \pm 0.62\%$

Table 4: **Omniglot:** Pre-training to Finetuning Accuracies (All Pre-train Ratios At 5%). $\text{Opt}_1 \rightarrow \text{Opt}_2$ refers to pre-training with Opt 1 and fine-tuning with Opt 2. Data is in the form of $\mu \pm 1\sigma$ over 4 different seeds. Bold entries indicate the best performance for each column, and for each pretraining optimizer.

7 Discussion

7.1 Mixing Optimizers Increases Catastrophic Forgetting

When fine-tuning with optimizers different from the pre-training optimizer, we see an increase in catastrophic forgetting for all the tasks and model architectures considered. As this effect persists across fine-tuning strategies, datasets, and architectures, we conclude that optimizer mismatch is a primary reason for the increase in catastrophic forgetting. Furthermore, the increase in forgetting due to optimizer mismatch is not negligible. As seen in Omniglot training 11, when pre-training with AdamW and injecting pre-training data with a ratio of 5%, there was a 3x increase in catastrophic forgetting. This result has vast implications for the future of fine-tuning models. When fine-tuning, it is very important to understand what optimizer was used during pre-training. If our results are consistent with other, non-classification domains (e.g. text generation, reinforcement learning), then these results may also apply to LLM fine-tuning.

Another novel observation from these experiments is that some combinations of optimizers are more compatible than others for certain architectures. As seen in 8 and 9, combining SGD with momentum and Adam shows a much smaller difference in the change in forgetting compared to combining Muon and Adam. This is consistent with the comments made by [17], where they highlight that

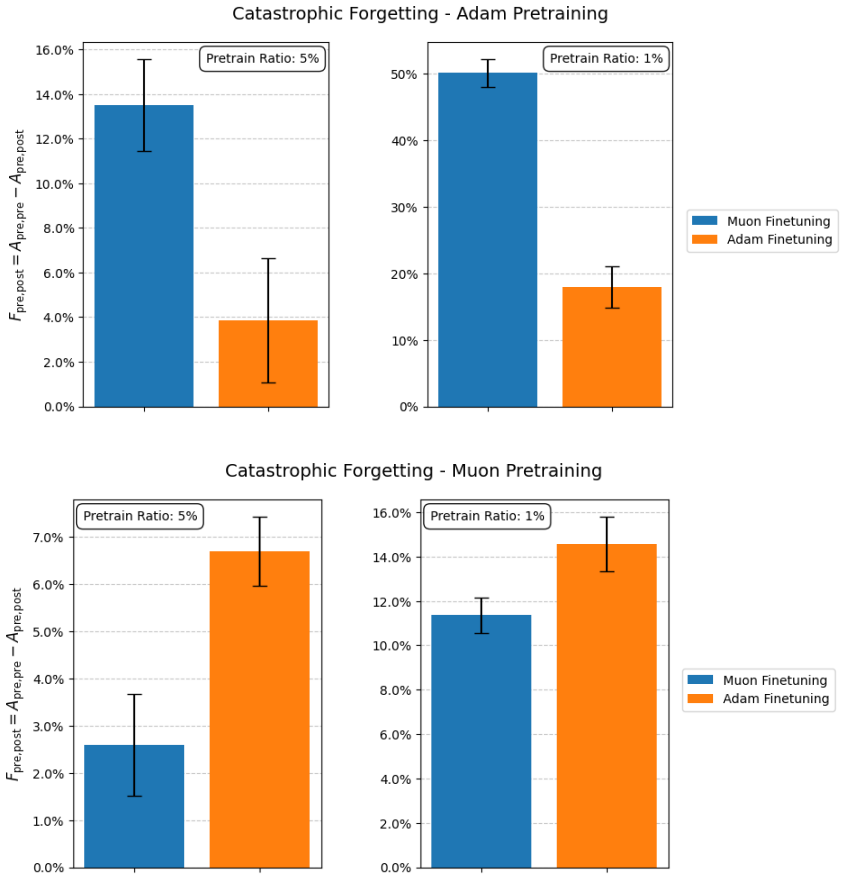


Figure 11: **Omniglot:** Catastrophic Forgetting With Adam and Muon Pre-training. For each pre-training, we measure the amount of catastrophic forgetting with Adam and Muon fine-tuning, and with different amounts of pre-training data injection. Bar indicate $\mu \pm 1\sigma$ over 4 different seeds.

fine-tuning with Muon after Muon pre-training is significantly better than fine-tuning with AdamW. However, there has been less discussion on the difference between using SGD and Adam during finetuning, which is understandable if the difference is small.

7.2 Optimizer Mismatch Forgetting Persists Despite Mitigation

Table 3 presents the effects of standard continual learning mitigation strategies on catastrophic forgetting across different combinations of pre-training and fine-tuning optimizers on CIFAR-10. Both knowledge distillation and linear probing consistently reduce forgetting relative to the baseline, with linear probing providing the largest absolute reductions across all optimizer configurations. However, despite these improvements, optimizer-mismatched settings (SGD with momentum→Adam and Adam→SGD with momentum) continue to exhibit substantially higher forgetting than matched settings, even when mitigation strategies are applied. The relative ordering of optimizer combinations remains largely unchanged, indicating that these mitigation techniques are insufficient to fully counteract the effects of optimizer mismatch. This persistence underscores the importance of optimizer choice in continual learning: while mitigation strategies can reduce forgetting, they do not eliminate the additional degradation introduced by changing optimizers between training phases.

7.3 Optimizers Follow Implicit Regularization

Different optimizers during training affect the norms of model weights differently, depending on the implicit regularization discussed in the Optimization Theory section of this report. As depicted in

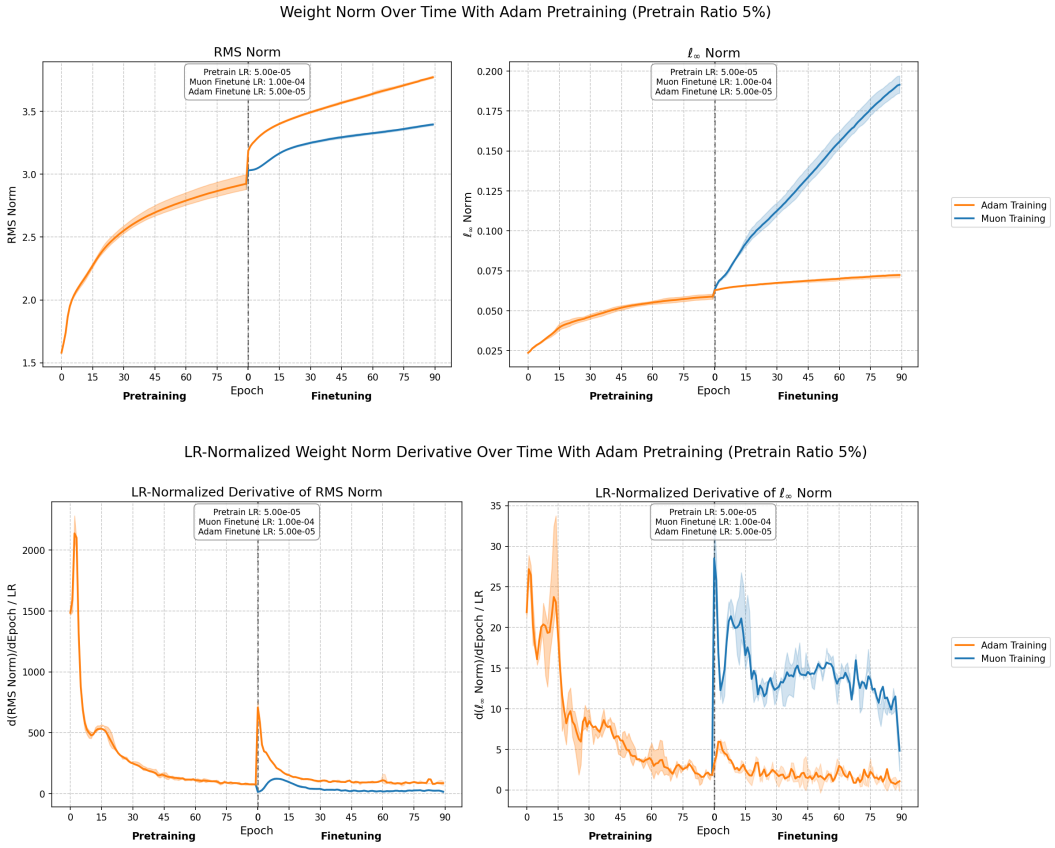


Figure 12: **Omniglot:** Weight Norm and Change in Weight Norm over Training, with AdamW Pre-training and Pre-training Data Injection of 5%. For the change in weight norms, we normalize by the current optimizer's learning rate as to measure the norm of the gradients at that point in training. Shaded regions indicate the maximum and minimum norms measured for each epoch over 4 different seeds.

figures 12 and 13, when investigating the change in the weight norms during Omniglot pre-training with Adam, we observe that fine-tuning on Adam keeps the ℓ_∞-norm of weight changes ($\Delta\theta$) the same as during pre-training. Conversely, when pre-training with Muon, it appears that the RMS norm of $\Delta\theta$ during Muon fine-tuning is able to return the initial pre-training values, but does not return for Adam fine-tuning.

We suspect that this change in norms between optimizers is a leading cause of increased forgetting. During pre-training, the optimizer tries to minimize $\Delta\theta$ with respect to a certain norm. As a result, the optimizer will lead the parameters to regions in parameter space where the gradients in the same norm are small. When switching to a different optimizer in fine-tuning, this norm constraint changes, and instead pushes parameters to regions where the norm of the gradients is large. By doing this, the fine-tuning optimizer moves away from the optimal surface that was found by the pre-training optimizer, leading to a decrease in pre-training performance.

8 Future Work

It is very costly to pre-train modern transformer models such as LLMs and VLAs. As mentioned in [17], further theoretical analysis of the effects of optimizer mismatch between pre-training and fine-tuning is essential to optimize performance of fine-tuned models, especially large models where fine-tuning is common. The majority of modern open source large language models have been pre-

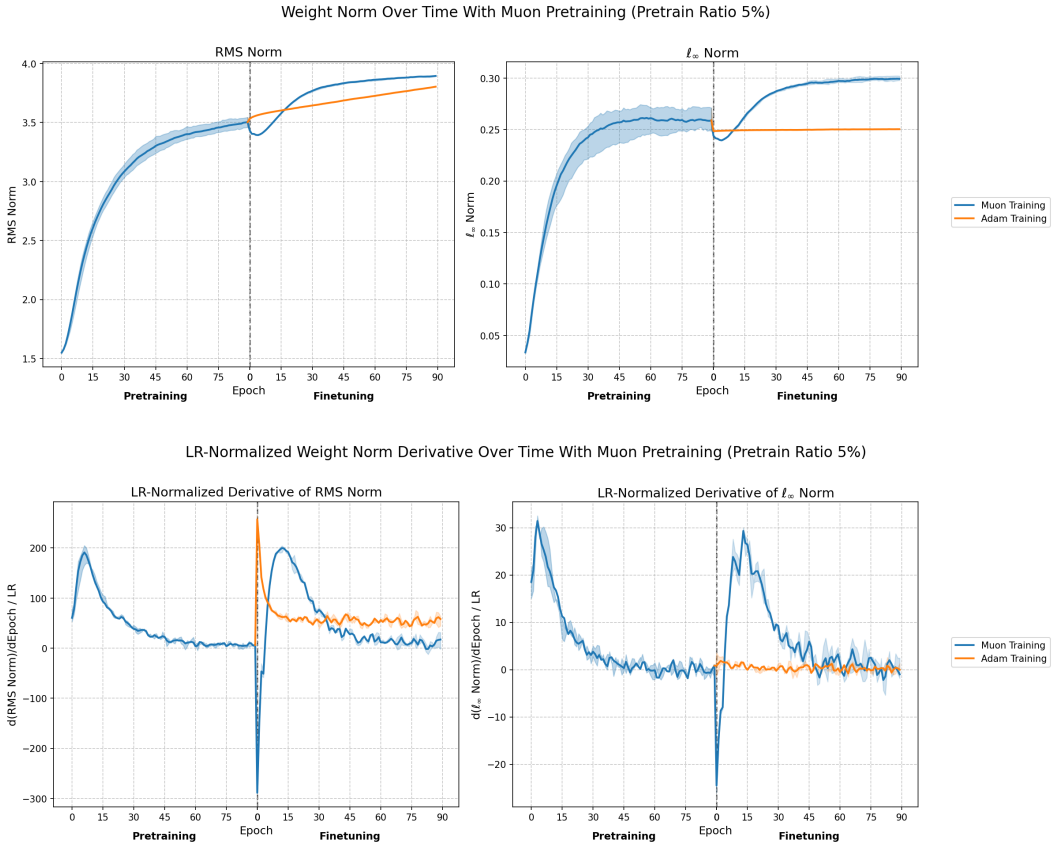


Figure 13: **Omniglot:** Weight Norm and Change in Weight Norm over Training, with Muon Pre-training and Pre-training Data Injection of 5%. For the change in weight norms, we normalize by the current optimizer's learning rate as to measure the norm of the gradients at that point in training.

trained with AdamW, and thus a solid theoretical understanding of the methods that can be used to mitigate the effects of optimizer mismatch while retaining the benefits of using newer optimizers (i.e. when transitioning from AdamW to Muon) can be used to better leverage the generalization capabilities of models pre-trained on internet-scale data. Thus, a potential avenue for future work could be in the development of optimization strategies or novel optimizers that can maintain the learned optimal regions within the parameter space.

References

[1] Dylan R. Ashley, Sina Ghiassian, and Richard S. Sutton. Does the adam optimizer exacerbate catastrophic forgetting? 2021.

[2] Jeremy Bernstein and Laker Newhouse. Modular duality in deep learning, 2024.

[3] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, 2009.

[4] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale, 2021.

[5] Pierre Foret, Ariel Kleiner, Hossein Mobahi, and Behnam Neyshabur. Sharpness-aware minimization for efficiently improving generalization, 2021.

[6] Ian J. Goodfellow, Mehdi Mirza, Xia Da, Aaron C. Courville, and Yoshua Bengio. An empirical investigation of catastrophic forgeting in gradient-based neural networks. *CoRR*, abs/1312.6211, 2013.

[7] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network, 2015.

[8] S. Kılıçarslan, H. A. Aydın, K. Adem, et al. Impact of optimizers functions on detection of melanoma using transfer learning architectures, 2025.

[9] Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, 2009.

[10] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C.J. Burges, L. Bottou, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012.

[11] Ananya Kumar, Aditi Raghunathan, Robbie Jones, Tengyu Ma, and Percy Liang. Fine-tuning can distort pretrained features and underperform out-of-distribution, 2022.

[12] Ananya Kumar, Ruoqi Shen, Sebastien Bubeck, and Suriya Gunasekar. How to fine-tune vision models with sgd, 2023.

[13] Brenden M. Lake, Ruslan Salakhutdinov, and Joshua B. Tenenbaum. Human-level concept learning through probabilistic program induction, 2015.

[14] Yann LeCun and Corinna Cortes. MNIST handwritten digit database. 2010.

[15] Xuhong Li, Yves Grandvalet, and Franck Davoine. Explicit inductive bias for transfer learning with convolutional networks, 2018.

[16] Zhizhong Li and Derek Hoiem. Learning without forgetting. *CoRR*, abs/1606.09282, 2016.

[17] Jingyuan Liu, Jianlin Su, Xingcheng Yao, Zhejun Jiang, Guokun Lai, Yulun Du, Yidao Qin, Weixin Xu, Enzhe Lu, Junjie Yan, Yanru Chen, Huabin Zheng, Yibo Liu, Shaowei Liu, Bohong Yin, Weiran He, Han Zhu, Yuzhi Wang, Jianzhou Wang, Mengnan Dong, Zheng Zhang, Yongsheng Kang, Hao Zhang, Xinran Xu, Yutao Zhang, Yuxin Wu, Xinyu Zhou, and Zhilin Yang. Muon is scalable for llm training, 2025.

[18] Zehua Liu, Han Wu, Xiaojin Fu, Shuqi Liu, Xiongwei Han, Tao Zhong, and Mingxuan Yuan. Reg: A regularization optimizer for robust training dynamics, 2025.

[19] Ziquan Liu, Yi Xu, Yuanhong Xu, Qi Qian, Hao Li, Xiangyang Ji, Antoni B. Chan, and Rong Jin. Improved fine-tuning by better leveraging pre-training data, 2022.

[20] Michael McCloskey and Neal J. Cohen. Catastrophic interference in connectionist networks: The sequential learning problem. volume 24 of *Psychology of Learning and Motivation*, pages 109–165. Academic Press, 1989.

[21] Chinmay Savadikar, Michelle Dai, and Tianfu Wu. Continual learning via learning a continual memory in vision transformer, 2024.

[22] Idan Shenfeld, Jyothish Pari, and Pulkit Agrawal. Rl's razor: Why online reinforcement learning forgets less, 2025.

[23] Yunfei Teng, Anna Choromanska, Murray Campbell, Songtao Lu, Parikshit Ram, and Lior Horesh. Overcoming catastrophic forgetting via direction-constrained optimization, 2022.

[24] Junjiao Tian, Chengyue Huang, and Zsolt Kira. Rethinking weight decay for robust fine-tuning of foundation models, 2024.

A ImageNet to MNIST Training (AlexNet)

Following the original training recipe for AlexNet [10], we adopt a batch size of 128 and use 90 epochs for all ImageNet pre-training runs. Our model pre-trained using SGD with momentum uses a step rate decay with maximum learning rate 0.01, again following the original AlexNet training recipe. As AdamW and Muon generally require lower learning rates compared to SGD, we performed a grid search on possible learning rates and schedulers and chose hyperparameters that reached the highest validation accuracy after 90 epochs. Similar to [16], we lower the learning rate from pre-training to fine-tuning ($0.1\times$ the pre-training rate) to preserve original task performance.

Table 5: Transfer Learning from ImageNet to MNIST with AlexNet

Hyperparameter	**Value**
Batch Size	128
Epochs	90
SGD + Momentum (pre-training)	
Learning Rate	1×10^{-2}
Momentum	0.9
Weight Decay	5×10^{-4}
LR Scheduler	Step Decay (every 30 epochs)
AdamW Settings (pre-training)	
Learning Rate	1×10^{-4}
Weight Decay	1×10^{-4}
LR Scheduler	Cosine Annealing
Muon Settings (pre-training)	
Muon Learning Rate	5×10^{-3}
Muon Momentum	0.95
Backbone Learning Rate	1×10^{-4}
Backbone Weight Decay	1×10^{-4}
Newton-Schulz Steps	5
LR Scheduler	Cosine Annealing
SGD + Momentum (fine-tuning)	
Learning Rate	1×10^{-3}
Momentum	0.9
Weight Decay	5×10^{-4}
AdamW Settings (fine-tuning)	
Learning Rate	1×10^{-5}
Weight Decay	1×10^{-4}
Muon Settings (fine-tuning)	
Muon Learning Rate	5×10^{-4}
Muon Momentum	0.95
Backbone Learning Rate	1×10^{-5}
Backbone Weight Decay	1×10^{-4}
Newton-Schulz Steps	5

B ImageNet to MNIST Training (ViT)

Similar to the pre-training settings, we use a batch size of 128. All MNIST fine-tuning runs are limited to 90 steps to reduce training time. For fine-tuning, we performed a small grid search over learning rates. Due to the long training times, an exhaustive search was not feasible. The objective was to identify learning rates for each optimizer that achieve the highest validation accuracy on ImageNet within 90 steps while also producing comparable accuracies across all three optimizers, ensuring a fair comparison when evaluating forgetting.

Table 6: Transfer Learning from ImageNet to MNIST with ViT-B/16

Hyperparameter	**Value**
Batch Size	128
Steps	90
AdamW Settings (fine-tuning)	
Learning Rate	1×10^{-5}
Weight Decay	1×10^{-4}
SGD + Momentum Settings (fine-tuning)	
Learning Rate	1×10^{-3}
Momentum	0.9
Weight Decay	1×10^{-4}
Muon Settings (fine-tuning)	
Muon Learning Rate	3×10^{-3}
Muon Momentum	0.9
Newton-Schulz Steps	3

C CIFAR-10 Training

Table 7: Continual Learning with CIFAR-10

Hyperparameter	**Value**
Batch Size	64
Pre-training Epochs	10
Finetuning Epochs	10
SGD + Momentum Settings (pre-training)	
Learning Rate	5×10^{-3}
Momentum	0.9
Weight Decay	5×10^{-4}
Adam Settings (pre-training)	
Learning Rate	5×10^{-5}
β_1	0.9
β_2	0.95
Weight Decay	5×10^{-4}
SGD + Momentum Settings (fine-tuning)	
Learning Rate	1×10^{-4}
Momentum	0.9
Weight Decay	5×10^{-4}
Adam Settings (fine-tuning)	
Learning Rate	1×10^{-6}
β_1	0.9
β_2	0.95
Weight Decay	5×10^{-4}
KL Divergence Settings	
α	1
Temperature	2

Table 7 summarizes the hyperparameters used for CIFAR-10 training. During pre-training, we performed a grid search over learning rates, weight decay values, and optimizer-specific hyperparameters such as momentum for SGD and the (β_1, β_2) coefficients for Adam. KL divergence hyperparameters were chosen following [16].

For fine-tuning, we fixed the optimizer-specific hyperparameters to the values selected during pre-training and performed a separate grid search over learning rates. In general, we selected smaller learning rates for fine-tuning than for pre-training, reflecting the fact that the pre-trained weights are already located in a region of parameter space that yields meaningful representations, and large updates risk overwriting previously learned information.

Additionally, Adam was tuned with a lower learning rate than SGD with momentum, consistent with prior empirical findings and common practice, as Adam's adaptive per-parameter updates effectively scale the step size and can lead to unstable or overly aggressive updates when combined with learning rates typically used for SGD. Using lower learning rates for Adam during fine-tuning improves stability and reduces catastrophic forgetting, particularly when transitioning between tasks.

D Omniglot Training

Table 8 shows the hyperparameters used for Omniglot training. For pre-training, we performed a grid search of learning rates and weight decays for both AdamW and Muon. For Muon, non-matrix parameters were trained using AdamW, using the same hyperparameter configuration as AdamW pre-training. For other hyperparameters related to the optimizers, we used the default values provided in the PyTorch implementations. SGD pre-training was not able to achieve non-zero success on Omniglot for any combination of learning rate and weight decay, so it was excluded for this task.

For fine-tuning, we selected the best hyperparameter configuration for each optimizer for pre-training accuracy. For each of these optimal configurations, we fine-tuned using Adam and Muon, doing grid search with another set or learning rates. Weight decay was set to 0 for all fine-tuning optimizers. For other hyperparameters related to the optimizers, we used the default values provided in the PyTorch implementations. Table 9 shows the optimal hyperparameter configurations found for each training run.

E Addressing Reviewer Feedback

On our initial draft, our reviewers had a couple key points:

1. **Hyperparameter and architecture choices need to be more thoroughly justified.** We have addressed this feedback in Appendix sections A through D. Notably, we justified our architectural choices through relevance to prior works, and more thoroughly described our process of grid search for optimal hyperparameters.
2. **We argue that optimizer mismatch leads to increased catastrophic forgetting due to the varying geometric landscapes that varying optimizers drive parameters towards. However, our prior results focused on accuracies, rather than providing direct evidence for our geometric argument.** We have addressed this feedback by including plots of the ℓ_∞ and RMS norms of the parameters in our final classifier heads. Current results show dramatic spikes in the various norms when switching optimizers from pre-training to fine-tuning, supporting our initial claim.
3. **AlexNet is a fairly old deep neural network and experiments should also be done on more modern architectures.** We included an experiment on transfer learning from ImageNet to MNIST using the vision transformer model ViT-B/16. Transformer architectures are more modern and more popular in current research, so we believe that the inclusion of these experiments help to validate this reviewer's concerns.
4. **The inclusion of CL techniques such as knowledge distillation, LP-FT, and linear probing feel tangential to the core hypothesis.** We added relevant literature exploring these techniques to our introduction. We further discuss how our usage of these techniques allows us to achieve adequate performance on our CL tasks for a proper comparison between optimizer choices, and how it allows us to emulate a range of realistic CL environments. Furthermore, we directly compare different optimizer choices in our results to narrow down confounding factors and support our core hypothesis.
5. **The language of we used was too hypothetical, and had inconsistent tense (i.e. "we hope...").** We have rewritten our sections to solely address work that we have already completed, in a consistent tense.

Table 8: Omniglot Training Hyperparameters

Hyperparameter	**Value**
AdamW Pre-training	
Batch Size	256
Epochs	90
Learning Rates	$\left[1 \times 10^{-4},\ 5 \times 10^{-5},\ 1 \times 10^{-5}\right]$
Weight Decays	$\left[5 \times 10^{-4},\ 1 \times 10^{-4},\ 1 \times 10^{-5}\right]$
β_1	0.9
β_2	0.999
Muon Pre-training	
Batch Size	256
Epochs	90
Learning Rates	$\left[5 \times 10^{-3},\ 1 \times 10^{-3},\ 5 \times 10^{-4}\right]$
Weight Decays	$\left[5 \times 10^{-4},\ 1 \times 10^{-4},\ 1 \times 10^{-5}\right]$
Momentum	0.95
Newton-Schulz Steps	5
Polynomial Coefficients	$a = 3.4445,\ b = -4.775,\ c = 2.0315$
Adam Finetuning	
Batch Size	256
Epochs	90
Learning Rates	$\left[1 \times 10^{-4},\ 5 \times 10^{-5},\ 1 \times 10^{-5}\right]$
β_1	0.9
β_2	0.999
Muon Finetuning	
Batch Size	256
Epochs	90
Learning Rates	$\left[1 \times 10^{-4},\ 5 \times 10^{-5},\ 1 \times 10^{-5}\right]$
Momentum	0.95
Newton-Schulz Steps	5
Polynomial Coefficients	$a = 3.4445,\ b = -4.775,\ c = 2.0315$

Table 9: Omniglot Optimal Hyperparameters

Optimizer	**Learning Rate**	**Weight Decay**
Pre-training		
AdamW	5×10^{-5}	1×10^{-4}
Muon	5×10^{-4}	1×10^{-5}
Adam Pre-training to Finetuning		
Adam	5×10^{-5}	0
Muon	1×10^{-4}	0
Muon Pre-training to Finetuning		
Adam	5×10^{-5}	0
Muon	1×10^{-4}	0