Continual Learning with Weight Interpolation

Supplementary Material

Table 3. Average accuracy and forgetting measure for CLeWI without rehearsal.

method	Acc(↑)	Acc_K (↑)	FM (↓)
CLeWI-SGD $\alpha = 0.2$	8.95	88.3	80.66
CLeWI-SGD $\alpha = 0.5$	9.87	85.7	71.04
CLeWI-SGD $\alpha = 0.6$	11.66	81.1	54.08
CLeWI-SGD $\alpha = 0.7$	16.41	62.2	22.0
CLeWI-SGD $\alpha = 0.8$	8.87	3.1	1.04

7. CLeWI without rehearsal

In preliminary experiments, we started by applying the weight interpolation without any other form of forgetting prevention. In such an approach, we kept the buffer in our algorithm, but it was used only to compute activations and batch normalization statistics required by REPAIR. Data from the buffer was not used for network training. The results are provided in Tab. 3. We tried several settings of α, but none provided significant gains in accuracy. With $\alpha = 0.2$ or $\alpha = 0.5$, the model forgets completely what it has learned previously, as illustrated by the high accuracy on the last task and low forgetting. By using $\alpha = 0.8$, we could almost entirely eliminate catastrophic forgetting, but only for the first task. The model lacks the plasticity to learn new tasks. The best results are obtained for $\alpha = 0.7$. However, they are far worse than any other method that stores previous data in a buffer.

We also provide the interpolation plots of CLeWI-SGD for the several values of α in Fig. 4. The local maxima of interpolation plots move significantly when different values of α are used. This was not observed for other continual learning algorithms that CLeWI was combined with. We can also see here that the model with $\alpha = 0.6$ or 0.7 should obtain higher accuracy, as this value aligns well with the local maxima of the plot. We obtain the best results as shown by Tab. 3 for these α values.

8. Interpolation plots for other algorithms

We present interpolation plots for various forms of rehearsal in Fig. 5. These plots show that each form of rehearsal requires a different interpolation α hyperparameter. These could be obtained with either a hyperparameter search, and interpolation plots can be helpful in this process. In experiments with standard benchmarks, we have tuned α only once and used it across different datasets.

We provide the exact values of the interpolation α in Tab. 4 for completeness.

Table 4. Values of interpolation α hyperparameter used in experiments

method	α
CLeWI ER	0.3
CLeWI aGEM	0.5
CLeWI ER ACE	0.3
CLeWI MIR	0.5
CLeWI BIC	0.5
CLeWI DER++	0.2

9. Interpolation and REPAIR algorithm details

In this section, we provide details about the interpolation process and REPAIR [20] algorithm used in our experiments. Both of these steps are represented in pseudocode as functions *calc_permutation* and *update_batchnorm*. In the first step, the optimal permutation is found. In the second step, batch normalization is performed in order to mitigate variance collapse.

Step 1 - alignment In the first step, we search for the permutation π of θ_P that maximizes the correlation between feature maps from θ_P and θ networks. The feature maps are obtained from samples in buffer $\mathcal{M}$. Note that $\mathcal{M}$ contains information about all tasks. Specifically, for a given layer, having the feature maps of the dimension $N \times C \times W \times H$ from two networks, we first calculate the $C \times C$ correlation matrix between them. Next, we choose the optimal permutation by solving the linear sum assignment problem with an optimizer from *scipy*. In our experiments to obtain the activations, we use only a single epoch with a batch size equal to 32.

Step 2 - normalization Aligned networks suffer from the phenomenon called variance collapse [20]. The variance of the feature maps is decreasing with network depth leading to poor performance. We prevent variance collapse by renormalizing feature maps, ensuring the variances of feature maps in the interpolated network θ_α satisfy conditions: $EX_\alpha = (1-\alpha)EX + \alpha EX_P$ and $\text{Var}X_\alpha = (1-\alpha)\text{Var}X + \alpha\text{Var}X_P$, where X_α, X_P, X are random variables corresponding to feature maps of θ_α (interpolated), θ_P (trained on previous tasks and permuted), and θ (trained on current task) networks respectively. To this aim, after all interpolated layers, Batch Normalization layers are added to apply an affine transformation to feature maps of the network θ_α. Parameters of the affine transformation are set to the interpolations of means and variances of feature maps in networks θ_P and θ, producing feature maps with means and standard deviations satisfying condi-

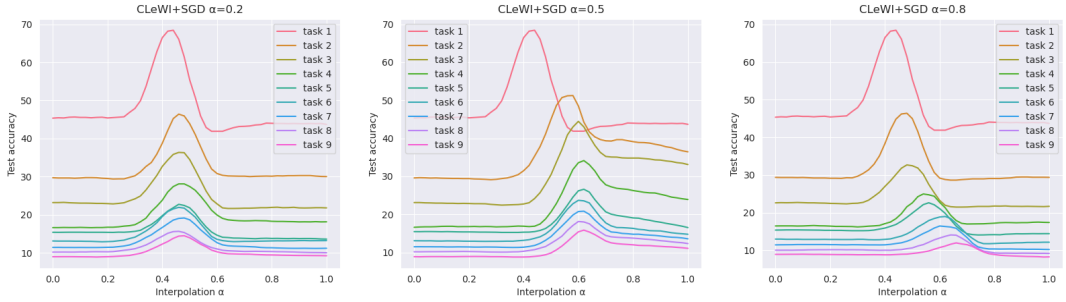

Figure 4. Test accuracy in the function of interpolation alpha for CLeWI with no replay

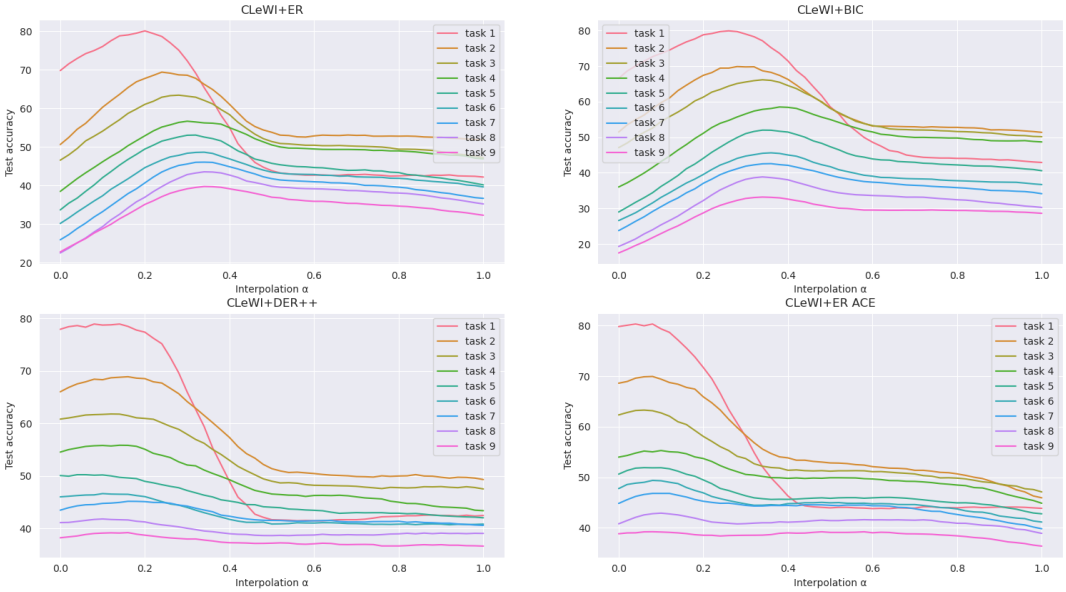

Figure 5. Interpolation plots for other forms of rehearsal

tions to prevent the variance collapse. After that, additional Batch Normalization layers are removed from the network via BatchNorm fusion. In our paper we use the observation from [20], that in architectures with Batch Normalization after each layer fixing the variance collapse via additional Batch Normalizations is equivalent to resetting the *batch_norm* statistics. We use a single epoch with data from $\mathcal{M}$ buffer to perform this reset.

10. Memory restricted evaluation of rehearsal methods

Some researchers in the continual learning community argue that a more fair comparison between algorithms would be assigning the same amount of memory for the buffer for all algorithms. All objects stored in memory, such as images, models, activations, or weights, would use this memory until the buffer is full. We are aware that our model requires additional memory for weights. For this reason, we carry out additional experiments in this memory-restricted evaluation mode to check if storing additional weights can bring improvement over increasing the buffer size alone.

As thorough evaluation requires several architectures, we employ both reduced ResNet architecture [17] with an overall number of parameters equal to 11220132 and MobileNetv2 [44] architecture with 2351972 parameters. To enable interpolation for MobileNetv2, we made a custom implementation of the REPAIR algorithm for this architecture. We are not applying the permutation for the depthwise convolution, as the same filter is applied to all input chan-

Table 5. Buffer size and accuracy for various continual learning algorithms and architectures for memory-restricted evaluation of CLeWI.

method	backbone	Cifar100(T=10)		Tiny-ImageNet(T=20)	
		#imgs in buffer	Acc (↑)	#imgs in buffer	Acc (↑)
ER	ResNet18	15000	62.61	4000	21.18
CLeWI+ER		394	38.03	348	9.45
ER	MobileNetv2	3500	26.55	1000	9.63
CLeWI+ER		438	18.67	235	6.96
ER	ResNet18	15606	61.53	4652	20.84
CLeWI+ER		1000	46.72	1000	21.52
ER	MobileNetv2	4062	27.37	1765	7.42
CLeWI+ER		1000	24.72	1000	12.16

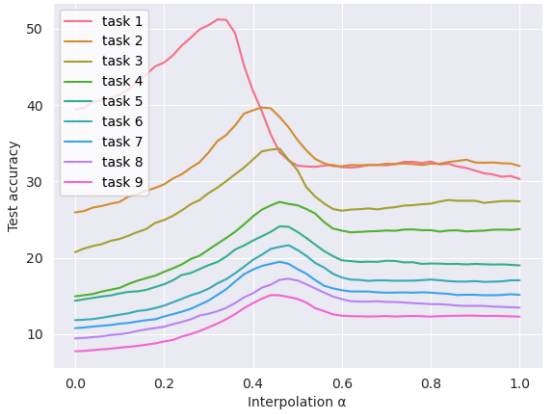

Figure 6. Test accuracy in the function of interpolation α for MobileNetv2 architecture

nels. For all other layers, permutation is applied. To verify that the algorithm works for other architecture, we made an interpolation plot, presented in Fig. 6. This figure shows that interpolation can still improve test set accuracy during training with the MobileNetv2 backbone.

We treat each network parameter as 32bit variable. We do not use mixed precision training [31], or other methods of reducing weights memory footprint. For Cifar100, each image is an array of size 32x32x3, with each color saved as an 8-bit variable. Analogically, we treat each Tiny-ImageNet image as a 64x64x3 array with 8-bit pixels.

From these assumptions, we can calculate the memory-wise equivalent of storing model weights in terms of additional images, which could be used in the rehearsal algorithm. For example, we computed that saving one ResNet corresponds to storing 14609 additional CIFAR100 images. We compare ER with CLeWI-ER to eliminate the influence of more advanced replay methods from our results. We carry out two experiments. In first we calculate the number of images that could be stored in the memory used by weights, and then we round this number up. CLeWI in this experiment has only the number of images that was rounded up, while ER has the full buffer size, which has the same memory footprint as both the memory buffer of CLeWI and model weights. In the second mode we increase the number of images stored in the CLeWI buffer to 1000, and increase the ER buffer size by the same amount. The results are provided in Tab. 5.

Two factors could greatly impact this evaluation mode: the size of the images in the dataset and the number of parameters in the model. For datasets with smaller images, such as CIFAR100, storing 32-bit weights in memory corresponds to a huge increase in the buffer size. In such cases, matching the ER's performance with a big buffer is hard. When we consider more parameter-efficient models, such as MobielNetv2, the gap between ER and CLeWI becomes smaller. The difference in the buffer size is also smaller, and the accuracy of CLeWI, especially for Tiny-ImageNet, becomes closer to ER. When we allow an increase in the buffer size for both algorithms, the gap between ER and CLeWI becomes even smaller, and we even notice accuracy improvement for the Tiny-ImageNet dataset. An increase in buffer size does not significantly impact the ER, as performance there could already be saturated.

Please note that here, we carry out experiments with benchmarks for continual learning used in the previous parts of the paper. If we consider other datasets with higher image sizes, such as 224x224 or 512×512, the comparison could be even more favorable for CLeWI. We may also employ more recent architectures for training, such as EfficientNetB0 [48], and utilize mixed precision training [31] to further reduce the memory footprint of the backbone.

We conclude that in a memory-restricted mode of evaluation, CLeWI could improve over vanilla ER, but only if the images are big and the network architecture is parameter efficient. In other cases, it could be more advisable to increase the buffer size alone and not use interpolation.