Re-basin via implicit Sinkhorn differentiation
Supplementary Material

In this appendix, we provide additional details for reproducing our work. The source code[1] to reproduce our results is publicly provided. Our implementation focuses on simplicity, allowing us to pass a PyTorch module[2] as an argument to our re-basin network and performing re-basin using a standard training cycle. Any cost function can be defined to guide the re-basin. The experimental setup for every experiment in the manuscript is detailed in the following sections.

1. Finding optimal transport

We used a feedforward neural network (NN) with 2, 4, and 8 hidden layers containing 10 neurons to find the optimal transport. The hyperbolic tangent was used as an activation function. Following the polynomial approximation dataset requirements, the number of inputs and outputs is set to 1. The re-basin optimization used Adam [6] with an initial learning rate of 0.1. The method used the proposed data-free squared L2 distance cost function (Eq. 7 of the manuscript). The maximum number of iterations was set to 100. However, in practice, all methods converged to a loss of 0 on the validation set in less than 50 iterations. An early stopping strategy was implemented to avoid running after the convergence point (see Fig. 1). The performance measurement employed during evaluation (see Table 1 of the manuscript) was the L1 norm between the re-based model θ_A and the target model θ_B:

$$L1(\hat{\mathcal{P}}; \theta_A, \theta_B) = |\pi_{\hat{\mathcal{P}}}(\theta_A) - \theta_B|, \tag{1}$$

where $\theta_A, \theta_B \in \mathbb{R}^d$, d is the number of parameters in the network, and $\hat{\mathcal{P}}$ is the set of estimated permutation matrices.

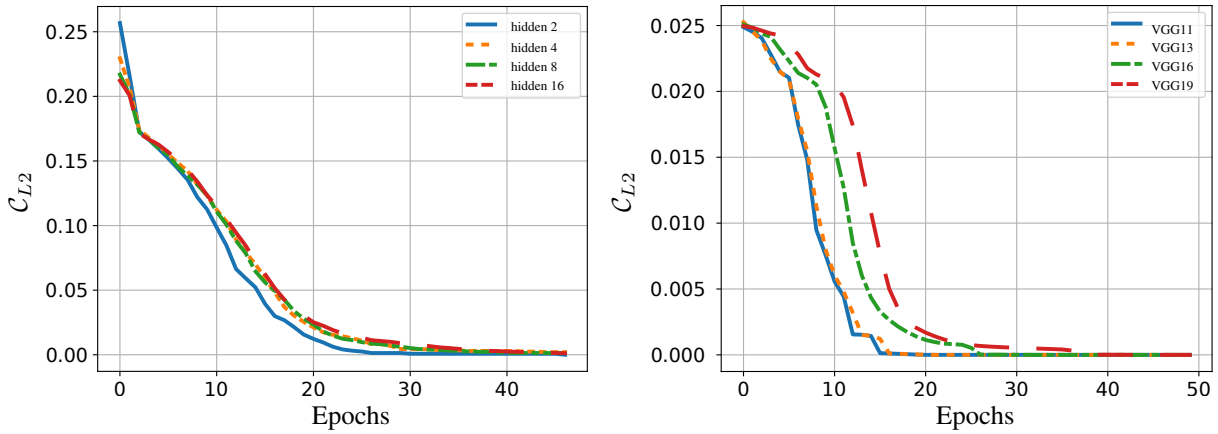

Figure 1. Validation loss during Sinkhorn re-basin training of feedforward NNs with a different number of hidden layers (left panel) and VGG with increasing depth (right panel).

To assess the impact of initialization, we propose three initial settings for re-basin. The first one is random initializations where the network's parameters were initialized following a normal distribution $\mathcal{N}(0, 1)$. The other two settings involved

[1] https://github.com/fagp/sinkhorn-rebasin

[2] Currently, our re-basin network only handles feedforward neural networks (NN), convolutional neural networks (CNN), and residual connections operations. Batch normalization suffers from variance collapse for LMC.

training NNs to perform regression tasks. The selected tasks were the first and third-degree polynomial approximation datasets [8], $\mathbb{T}_{Pol1} = \{(x, y) \mid y = x + 3, x \in (-4, -2)\}$ and $\mathbb{T}_{Pol3} = \{(x, y) \mid y = (x - 3)^3, x \in (2, 4)\}$. For training the polynomial approximation networks, a small Gaussian noise with distribution $\mathcal{N}(0, 0.05)$ was added to the regression target [8].

Similar to other related works in the literature, our re-basin method is not limited to regression tasks or shallow feedforward NNs. Although we selected regression tasks for comparison purposes, we want to highlight that our approach can also perform re-basin for classification tasks, deepest feedforward NNs, and convolutional neural networks (CNNs). In particular, we use different standards of VGG without batch normalization. Fig. 1 shows the loss curves of our Sinkhorn re-basin training using the cost in Eq. 7 of the manuscript, $\mathcal{C}_{L2}$. The initial models were obtained by training NNs and CNNs over the Mnist dataset. In all cases, the optimal transport was found in less than 20 seconds running on an Nvidia GeForce RTX 3070 GPU.

We created a more challenging permutation scenario to assess robustness against noise. We followed the experiment in Section 5.1 of the manuscript but added Gaussian noise $\mathcal{N}(0, 10^{-3})$ to the parameters of the permuted models. All obtained noisy models were valid for solving the tasks. As observed in Tab. 1, although the target permutation was not found, our proposed method obtained closer re-basins than WM.

Method	Init	2 hidden ↓	4 hidden ↓	8 hidden ↓
WM [1]	Rnd	57.81±8.70	20.71±5.57	12.44±3.05
$\mathcal{C}_{L2}$ (Ours)		**0.03±0.00**	**0.02±0.00**	**0.02±0.00**
WM [1]	Pol3	10.73±3.70	3.13±1.19	**0.02±0.00**
$\mathcal{C}_{L2}$ (Ours)		**0.03±0.00**	**0.02±0.00**	**0.02±0.00**
WM [1]	Pol1	0.48±0.31	**0.02±0.00**	**0.02±0.00**
$\mathcal{C}_{L2}$ (Ours)		**0.03±0.0**	**0.02±0.00**	**0.02±0.00**

Table 1. L1 distance between the estimated and expected re-basin with different network initialization and depth. Distances are scaled $\times 10^4$.

2. Linear mode connectivity

To assess the capacity of achieving linear mode connectivity, we employed two classification datasets – Mnist and Cifar10 – and the previously described first and third-degree polynomial approximation datasets [8]. As base architecture, we used a feedforward NN with two hidden layers. In classification cases, the activation function was ReLU with $784 = 28 \times 28$ neurons within the input layer for Mnist and $3072 = 32 \times 32 \times 3$ for Cifar10. For both benchmarks, the output layer has 10 neurons corresponding with the number of classes. The cross-entropy loss function was used for training these classification networks. As for the regression tasks, only 1 input and 1 output were required. Similarly to the previous experiment, hyperbolic tangent activation was used. The L2 loss function was used for training the regression networks.

The Sinkhorn re-basin networks used the Adam optimizer with a maximum of 1000 iterations. In practice, none of the executions trained until the maximum number of iterations, thanks to the early stopping. In the classification settings, all experiments converged in less than 50 iterations. The best initial learning rate for every configuration is given in Tab. 2. The mini-batch sizes were 100 for regression and 1000 for classification.

Dataset/Method	$\mathcal{C}_{L2}$	$\mathcal{C}_{Mid}$	$\mathcal{C}_{Rnd}$
First degree polynomial	0.10	0.10	0.01
Third degree polynomial	0.10	0.10	0.01
Mnist	0.01	0.10	0.10
Cifar10	0.01	0.10	0.10

Table 2. Initial learning rate for every dataset and method in experiments with linear mode connectivity.

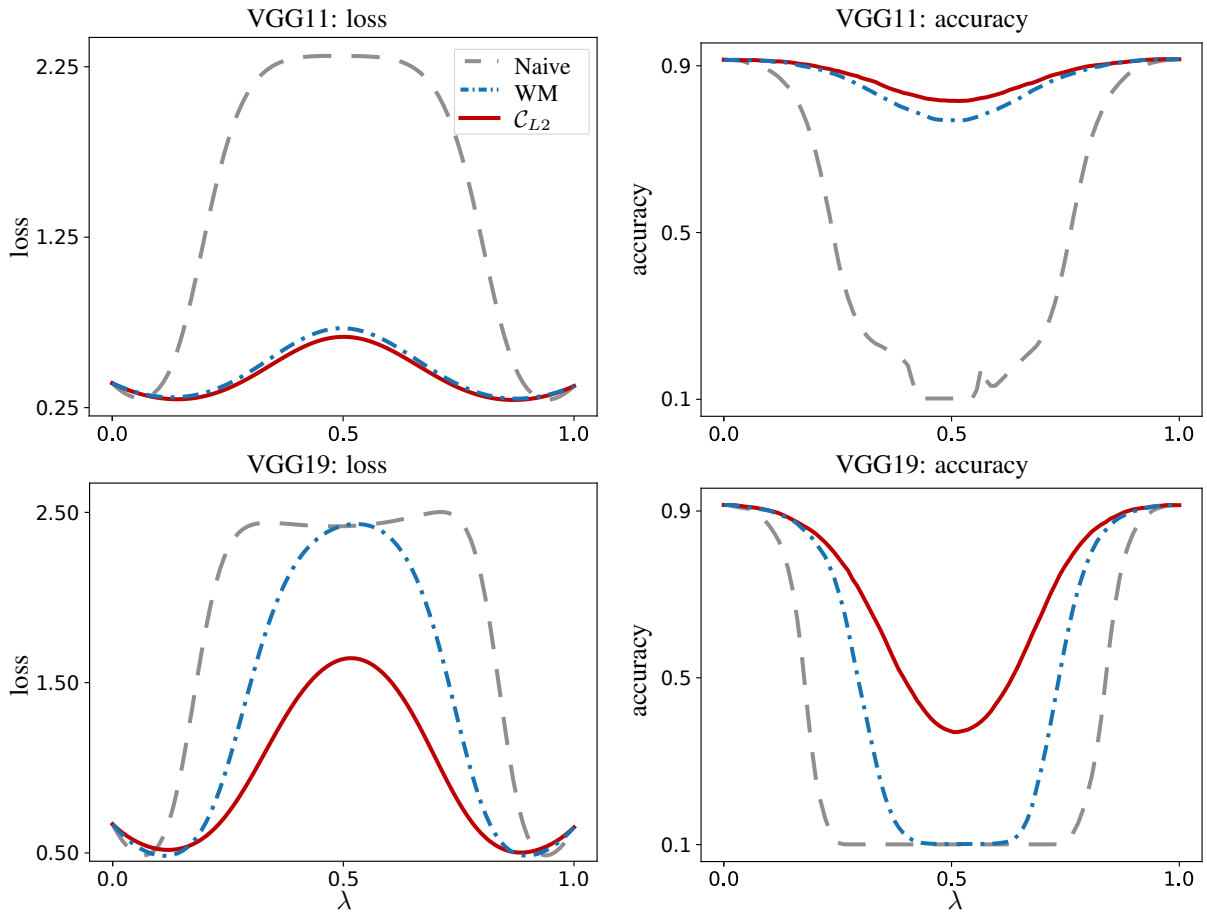

Figure 2. Linear mode connectivity using VGG11 and VGG19 networks trained over Cifar10 dataset. The left panel shows the loss over the linear path, while the right panel presents the accuracy.

To assess performance, we use the Barrier [4]:

$$B(\theta_A, \theta_B) = \sup_{\lambda}[[\mathcal{C}((1-\lambda)\theta_A + \lambda\theta_B)] - [(1-\lambda)\mathcal{C}(\theta_A) + \lambda\mathcal{C}(\theta_B)]], \tag{2}$$

with $\lambda \in (0, 1)$, and Area Under the Curve (AUC) over the estimated cost curve within the linear path:

$$AUC(\theta_A, \theta_B) = \int_{\lambda=0}^{1} [\mathcal{C}((1-\lambda)\theta_A + \lambda\theta_B)] - [(1-\lambda)\mathcal{C}(\theta_A) + \lambda\mathcal{C}(\theta_B)]\mathrm{d}\lambda. \tag{3}$$

Our method is not limited to NNs for linear mode connectivity as in the previous experiment. Examples of LMC using two VGG11 trained over the Cifar10 dataset are shown in Fig. 2. The naive path is presented with dashed gray lines while the cost and accuracy after re-basin with our Sinkhorn network with L2 loss are shown in solid red lines, and Weights Matching (WM) [1] in dashed blue. An example with VGG19 is also presented in the figure to exemplify some complex cases where LMC is not achieved. Although both methods struggle, our data-free approach can generally find better re-basins. The mean barrier and AUC of VGGs with different depths are given in Fig. 3 for our proposal with L2 cost and WM [1]. Similarly, we exemplify in Fig. 4 the ability of our method to perform re-basin in residual networks like ResNet18 and ResNet34 trained over Imagenet subset, Imagenette [3]. As observed, the obtained LMC after re-basin also suffered from a high barrier. One possible explanation for such a result is that interpolated networks suffer a collapse in the variance of their activations needing a re-normalization as proposed by Jordan *et al*. [5]. We leave as future work to incorporate REPAIR [5] approach to our LMC approach.

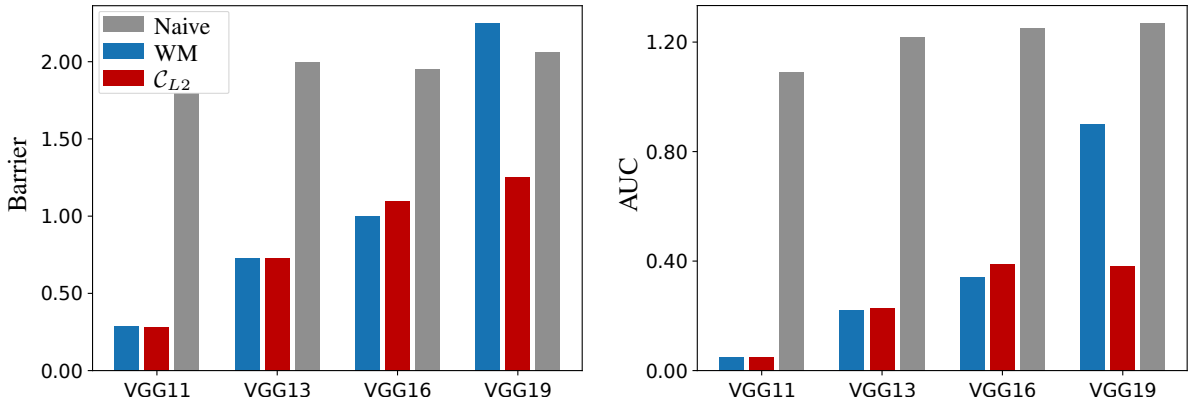

Figure 3. Average Barrier (left panel) and AUC (right panel) of VGG with different depths.

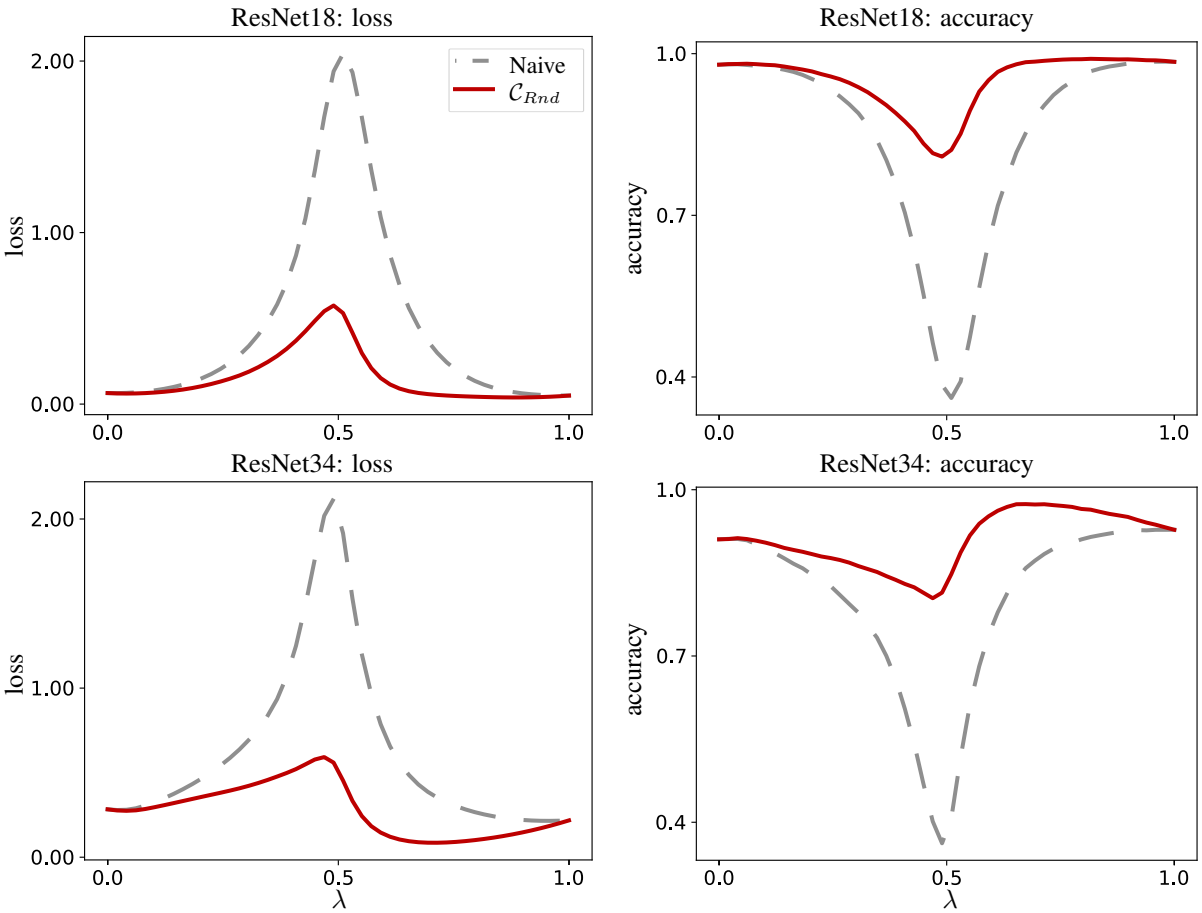

Figure 4. Linear mode connectivity using ResNet18 and ResNet34 networks trained over Imagenette dataset. The left panel shows the loss over the linear path, while the right panel presents the accuracy.

3. Incremental learning algorithm

Incremental learning scenarios followed the standard procedure found in the literature, i.e., a benchmark is divided into several episodes, and new knowledge is learned incrementally by the model at episode i, θ_i. In our experiments, we employed 20 episodes. Regarding the benchmarks, we used the classical Rotated Mnist, consisting of rotated versions of the Mnist dataset from 0 degrees to 180 degrees. At each episode, a clockwise rotation of 9.47 degrees was applied to every image in

the previous episode. Note that, independently of the rotation applied, the classes remain the same. To address this challenge, we used a feedforward NN with 1 hidden layer and 256 neurons within the layer. ReLU activation was used for the hidden layer. The input layer size was $784 = 28 \times 28$. The number of neurons in the last layer was 10, corresponding with the number of classes. The algorithm for performing the re-basin incremental learning is outlined in Algorithm 1. Note that this algorithm is defined using Stochastic Gradient Descent (SGD) for simplicity, but in practice, there is no constraint in which an optimizer can be employed. In our experiments, we used Adam with an initial learning rate $\eta = 0.001$ for the first task, and $\eta = 0.1$ for the continual learning. As for the residual model, we used SGD with a learning rate of $\gamma = 0.05$, and weight decay $\beta = 0.1$. A total of 5 epochs were used to incorporate the new knowledge at each episode using a mini-batch size of 500. As the manuscript relates, the fusion hyper-parameter $\alpha = 0.8$ was used in all our experiments.

As for the second benchmark, we used the Split Cifar100 – the Cifar100 dataset was partitioned into 20 smaller datasets with 5 classes each. Every episode had labeled images corresponding to the 5 categories at hand. The architecture was the multi-head ResNet18 from [7]. Similarly to the previous benchmark, we used Adam for the re-basin and SGD for learning the residual, this time using a mini-batch size of 10 and 20 training epochs for the initial model. The rest of the parameters remained the same as in the previous benchmark, except for the residual vector training, which needed a learning rate of $\gamma = 0.5$, and weight decay of $\beta = 10^{-4}$.

Procedure 1 Re-basin incremental learning.

Input: A set of episodes $\mathbb{T} = \{\mathbb{T}_1, ..., \mathbb{T}_n\}$, and a model f_{θ_0} trained on $\mathbb{T}_0$.
Output: A model θ_n.

for $e = 1$ **to** n **do**
- $\mathcal{P}^0 \leftarrow \{P_j | P_j = \boldsymbol{I}\}, 1 \leq j \leq h$ ▷ Initialize permutation matrices to identity. h is the number of hidden layers.
- $\delta^0 \leftarrow \mathrm{U}(0, 10^{-6})^d$ ▷ Initialize values of vector δ^0 following $\mathrm{U}(0, 10^{-6})$. d is the number of parameters.
- **for** $i = 0$ **to** it **do**
  - $(x, y) \leftarrow \mathrm{U}(\mathbb{T}_e)$ ▷ Sample a mini-batch (x, y) from $\mathbb{T}_e$.
  - $\theta^i \leftarrow \frac{\theta_{e-1} + \pi_{\mathcal{P}^i}(\theta_{e-1})}{2} + \delta^i$ ▷ Compute the model in the middle of the re-basin line, plus the residual.
  - $\mathcal{C}(\delta^i, \mathcal{P}^i; \theta^i) = \mathcal{L}(y, f(x; \theta^i)) + \beta ||\delta^i||^2$ ▷ $\mathcal{L}$ depends on the task, i.e., cross entropy for classification.
  - $\mathcal{P}^{i+1} \leftarrow \mathcal{P}^i - \eta \frac{\partial \mathcal{C}(\delta^i, \mathcal{P}^i; \theta^i)}{\partial \mathcal{P}^i}$ ▷ Backpropagation and gradient descent for permutation matrices $\mathcal{P}$.
  - $\delta^{i+1} \leftarrow \delta^i - \gamma \frac{\partial \mathcal{C}(\delta^i, \mathcal{P}^i; \theta^i)}{\partial \delta^i}$ ▷ Backpropagation and gradient descent for residual δ.
- **end for**
- $\theta_e \leftarrow (1 - \alpha)\theta_{e-1} + \alpha\theta_{e-1} + \delta^{it}$ ▷ Fuse models for task $\mathbb{T}_{e-1}$ and $\mathbb{T}_e$.

end for

The accuracy at episode $e = 20$ is computed as the average accuracy of model θ_e over every test dataset from current and previous tasks. On the other hand, forgetting measurement seeks to measure the ability to retain knowledge by computing the highest difference in accuracy between the current model and the previous ones for every task.

As can be seen in algorithm Algorithm 1, the value of α defines the balance between new and previous knowledge. We allow the user to select this hyper-parameter according to the problem at hand instead of learning it. Values of α around 0.5 lead to better models for the new task, while α closer to 0.0 or 1.0 favors the previous knowledge. We performed an ablation over the Rotated Mnist benchmark to analyze the influence of α and β. Our method yields a similar accuracy of 0.79 for $\alpha \in (0.2, 0.8)$ and $\beta \in (10^{-4}, 10^{-1})$. Similar forgetting is also achieved in the same range, where the smaller value of 0.10 is obtained for $\beta = 0.1$. Overall, our method shows robustness to the choice of α and β as observed in Tab. 3. Accuracy's evolution for tasks 1, 5, 10, 15, and 20 were analyzed for the best hyper-parameters. As shown in Fig. 5, the proposed re-basin incremental learning outperformed those from the literature showing a significant improvement in terms of forgetting while maintaining high levels of accuracy.

4. Computational complexity

Our Sinkhorn re-basin uses the method in [2] with time and memory complexity of $\mathrm{O}(n^3)$ and $\mathrm{O}(n^2)$, respectively, where n is the size of the square permutation matrix. These complexities are equivalent to Hungarian algorithm-based approaches like weight matching (WM). Furthermore, our method has an additional memory requirement of $\mathrm{O}(n^3)$ to store gradient values during training. We measured the average execution time needed for re-basin the same 50 models using the original

	β=10^{-4}		β=10^{-3}		β=10^{-2}		β=10^{-1}		β=1	
α	Acc ↑	Forg ↓	Acc ↑	Forg ↓	Acc ↑	Forg ↓	Acc ↑	Forg ↓	Acc ↑	Forg ↓
0.0	0.69	0.02	0.69	0.04	0.71	0.02	0.69	0.00	0.44	0.00
0.2	0.79	0.14	0.80	0.14	0.79	0.14	0.80	0.10	0.69	0.02
0.4	0.78	0.16	0.80	0.15	0.80	0.13	0.80	0.10	0.68	0.02
0.6	0.78	0.16	0.80	0.13	0.79	0.14	0.79	0.10	0.68	0.02
0.8	0.79	0.14	0.81	0.13	0.80	0.13	0.80	0.10	0.69	0.02
1.0	0.69	0.02	0.69	0.04	0.71	0.02	0.69	0.00	0.44	0.00

Table 3. Accuracy (Acc) and forgetting (Forg) of the proposed continual learning method on Rotated Mnist with 20 episodes for $\alpha \in \{0.0, 0.2, 0.4, 0.6, 0.8, 1.0\}$ and $\beta \in \{10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}, 1\}$.

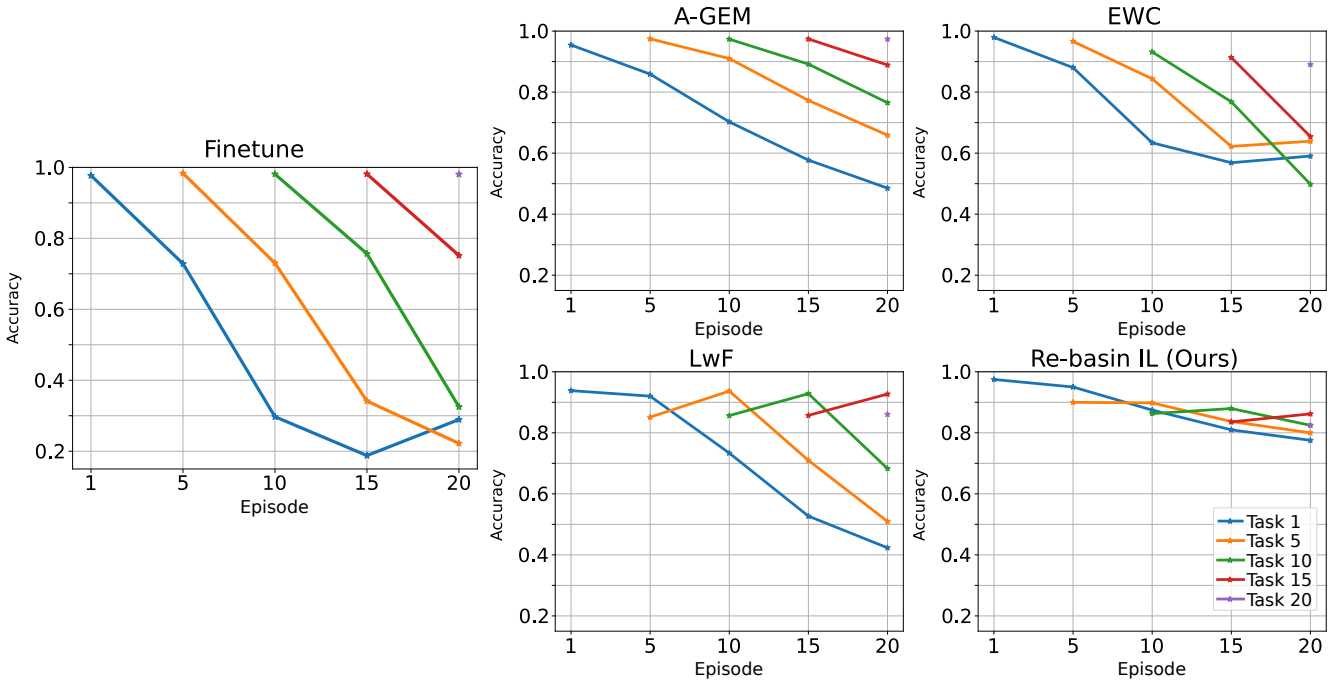

Figure 5. Evolution of task's accuracy during the incremental learning experience on Rotated Mnist with 20 episodes. Only tasks 1, 5, 10, 15, and 20 are shown for simplicity.

WM and ours. Both methods were executed on an Intel i7 CPU at 4.6GHz. Our Sinkhorn-based re-basin ran in 0.18 ± 0.03 secs, while WM ran in 0.35 ± 0.02 secs. Although our method executes faster, we are aware that the frameworks – Jax for WM and PyTorch for ours– and implementation details differ.

References

[1] Samuel K Ainsworth, Jonathan Hayase, and Siddhartha Srinivasa. Git re-basin: Merging models modulo permutation symmetries. *arXiv preprint arXiv:2209.04836*, 2022. 2, 3

[2] Marvin Eisenberger, Aysim Toker, Laura Leal-Taixé, Florian Bernard, and Daniel Cremers. A unified framework for implicit sinkhorn differentiation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 509–518, 2022. 5

[3] FastAI. Imagenette dataset, 2020. 3

[4] Jonathan Frankle, Gintare Karolina Dziugaite, Daniel Roy, and Michael Carbin. Linear mode connectivity and the lottery ticket hypothesis. In *International Conference on Machine Learning*, pages 3259–3269. PMLR, 2020. 3

[5] Keller Jordan, Hanie Sedghi, Olga Saukh, Rahim Entezari, and Behnam Neyshabur. Repair: Renormalizing permuted activations for interpolation repair. *arXiv preprint arXiv:2211.08403*, 2022. 3

[6] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, 2015. 1

[7] Seyed Iman Mirzadeh, Mehrdad Farajtabar, Dilan Gorur, Razvan Pascanu, and Hassan Ghasemzadeh. Linear mode connectivity in multitask and continual learning. In *International Conference on Learning Representations*, 2021. 5

[8] Johannes von Oswald, Christian Henning, Benjamin F. Grewe, and João Sacramento. Continual learning with hypernetworks. In *International Conference on Learning Representations*, 2020. 2