Model Merging & Editing

EECE695D: Efficient ML Systems

Spring 2025

Recap

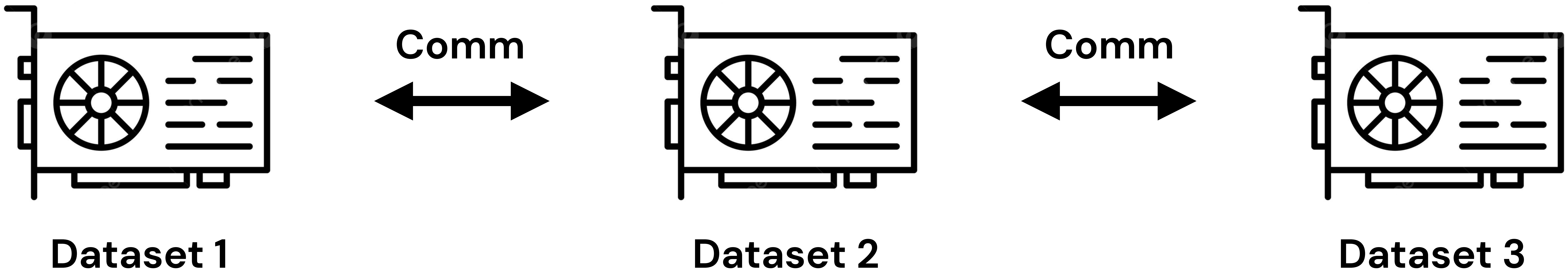
- **Last week.** Train a model, using knowledge transferred from other training runs
    - Continual Learning
    - Meta-Learning

- **Today.** Post-training methods
    - Merging. Transfer experience
    - Editing. Pinpoint fixes

Merging

Model Merging

- **Goal.** Want to aggregate the knowledge of concurrent training runs
  - Decentralize, due to privacy or computational cost
  - Depends critically on how often we can communicate
    - <u>High</u>. SGD (w/ parallelism)
    - <u>Medium</u>. Federated Learning
    - <u>Low</u>. Merging



High Comm.: SGD

- Every step, aggregating experiences of B clients (B: batch size)
  - Initialize the parameter θ_0
  - In each step $t = 0, 1, \ldots$
    - For each client $i \in \{1, \ldots, B\}$ **Local Training**
      - Draw a single sample (x_i, y_i)
      - Generate a local update $\theta_t^{(i)} = \theta_t - \eta \cdot \nabla_\theta \ell(y_i, f_{\theta_t}(x_i))$
    - Aggregate the experiences: **Aggregate**

$$\theta_{t+1} = \frac{1}{B} \sum_{i=1}^{B} \theta_t^{(i)}$$

Hospidales et al., "Meta-Learning in Neural networks: A Survey," IEEE TPAMI 2022

Medium Comm.: Federated Learning

- **FedAvg (2017).** Aggregate every E steps
  - Initialize the parameter θ_0
  - In each round $t = 0,1,\ldots$
    - For each client $i \in \{1,\ldots,B\}$
      - Initialize the local checkpoint $\theta_{t,0}^{(i)} = \theta_t$
      - For each local step $j = 1,\ldots,E$
        - Draw a batch of samples
        - Update the local checkpoint $\theta_{t,j}^{(i)} = \theta_{t,j-1}^{(i)} - \eta \sum_k \nabla_\theta \ell(y_k, f_{\theta_{t,j-1}^{(i)}}(x_k))$

Local Training with E steps

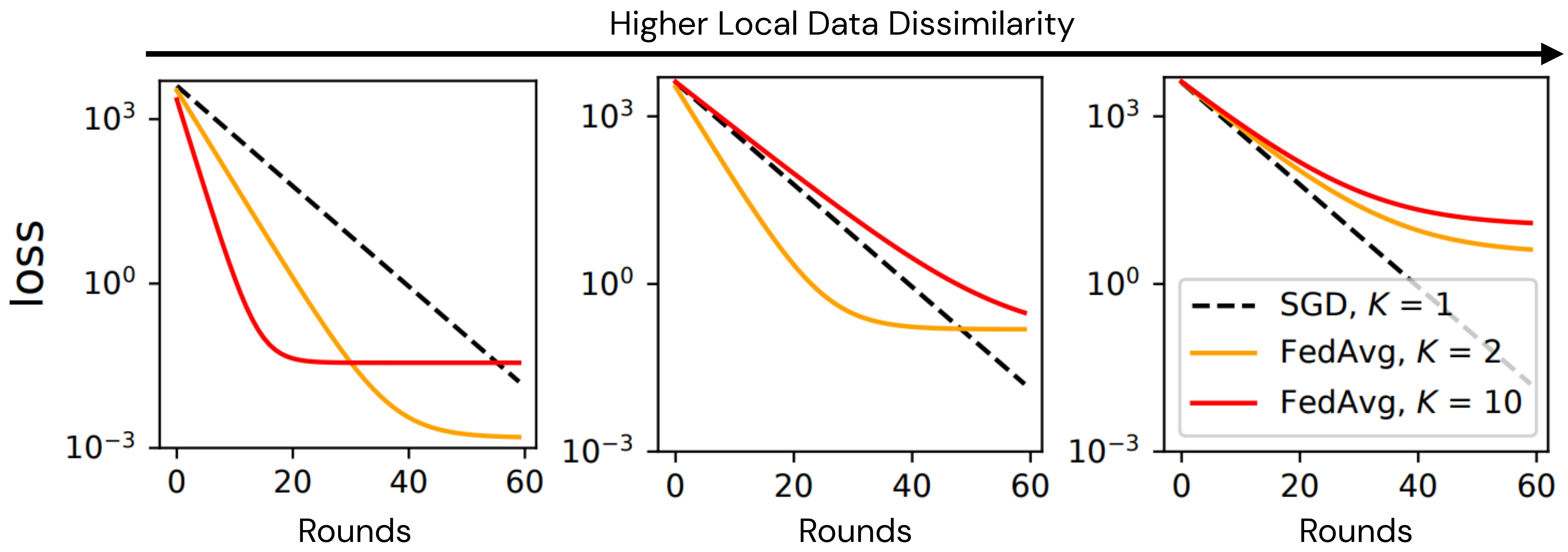
    - Aggregate the experiences:

$$\theta_{t+1} = \frac{1}{B} \sum_{i=1}^{B} \theta_{t,E}^{(i)}$$

McMahan et al., "Communication-Efficient Learning of Deep Networks from Decentralized Data," AISTATS 2017

Medium Comm.: Federated Learning

- Two factors critically affect the performance:
- **(1) Frequency.** The number of local steps should be small
  - Especially when local data are dissimilar



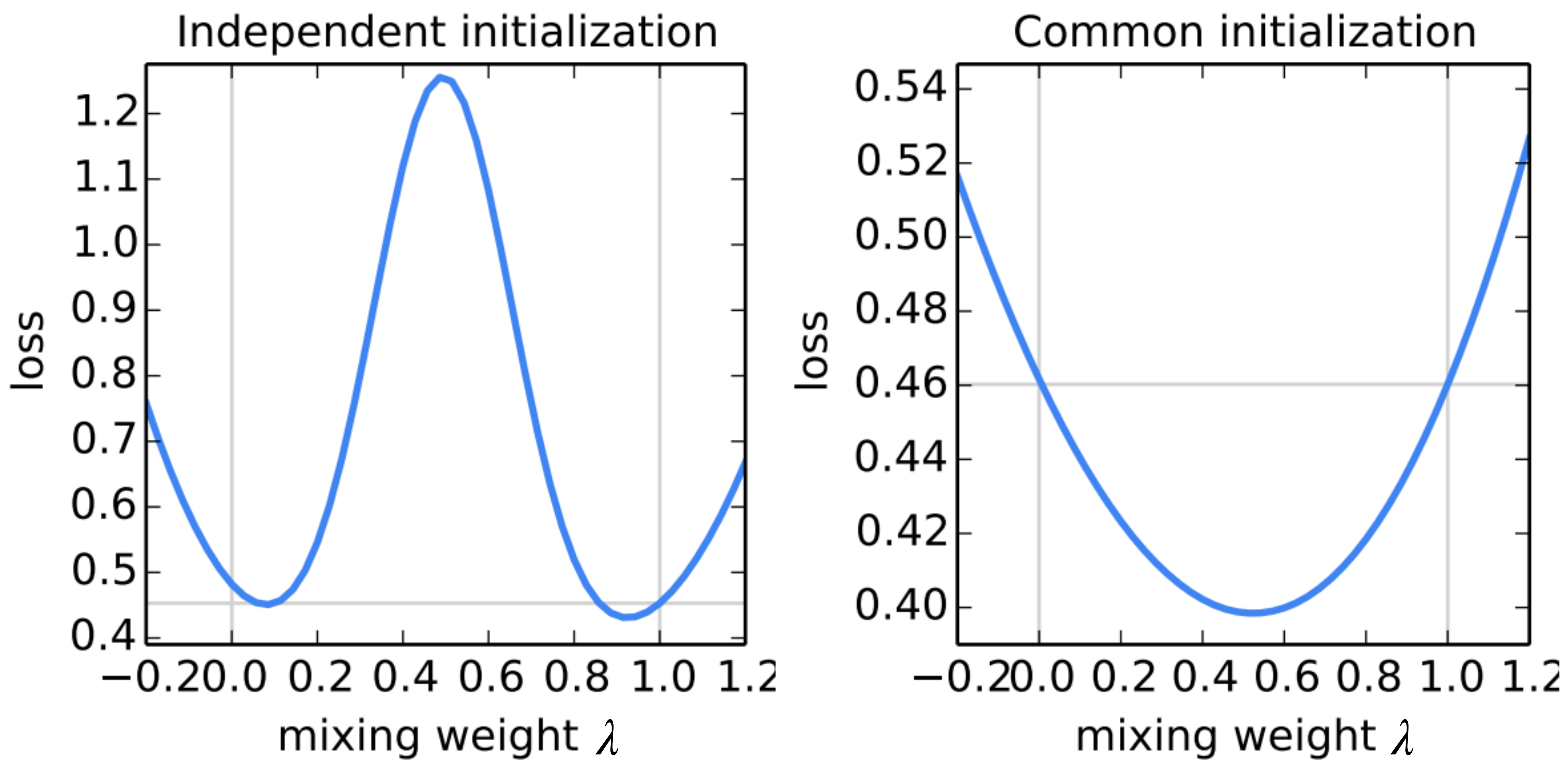
K: Number of clients

Karimireddy et al., "SCAFFOLD: Stochastic Controlled Averaging for Federated Learning," ICML 2020

Medium Comm.: Federated Learning

- **(2) Shared init.** The initial parameter θ_0 should be identical
  - Otherwise, high loss barrier between weights

$$\lambda \cdot \theta_1 + (1 - \lambda) \cdot \theta_2$$



McMahan et al., "Communication-Efficient Learning of Deep Networks from Decentralized Data," AISTATS 2017

Low Comm.: Merging

- **Challenge.** Can we merge two independently trained models, with a single aggregation after training?
    - Ideally, we would want:
        - If trained on a same dataset, achieve the accuracy of model ensemble (with cheaper inference)
        - If trained on different datasets, achieve good accuracy in both domains

McMahan et al., “Communication-Efficient Learning of Deep Networks from Decentralized Data,” AISTATS 2017

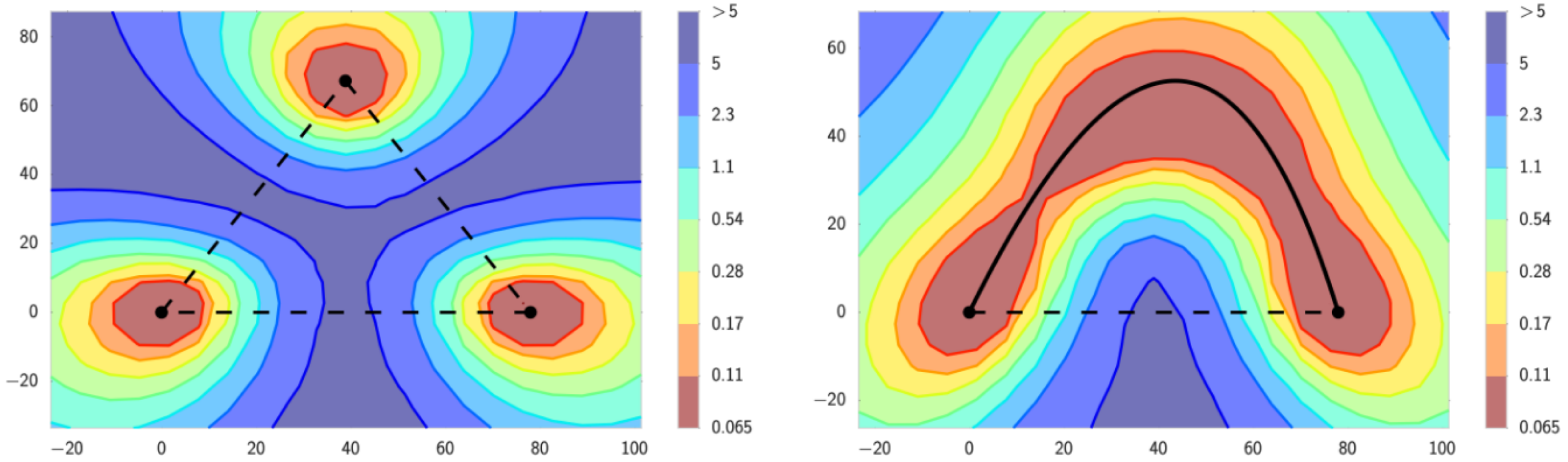
Low Comm.: Merging

- **Scenarios.** Roughly two categories:
  - Independent initialization:
    - Git Re-Basin, REPAIR, ZipIt!
  - Pre-trained model as initialization:
    - Model Soup

Merging: Independent init.

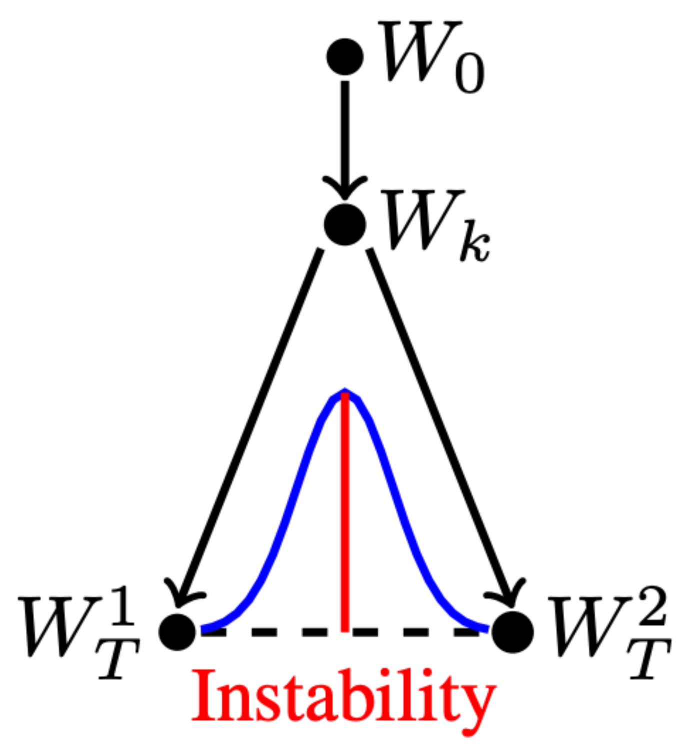
Mode connectivity

- By 2017, people realized that there exists a nonlinear low-loss curve in the parameter space between two independently trained models (w/ same data)
  - <u>Note</u>. Two sources of randomness; init & SGD ordering
- **Problem.** Nonlinear, so requires an extensive search for interpolation

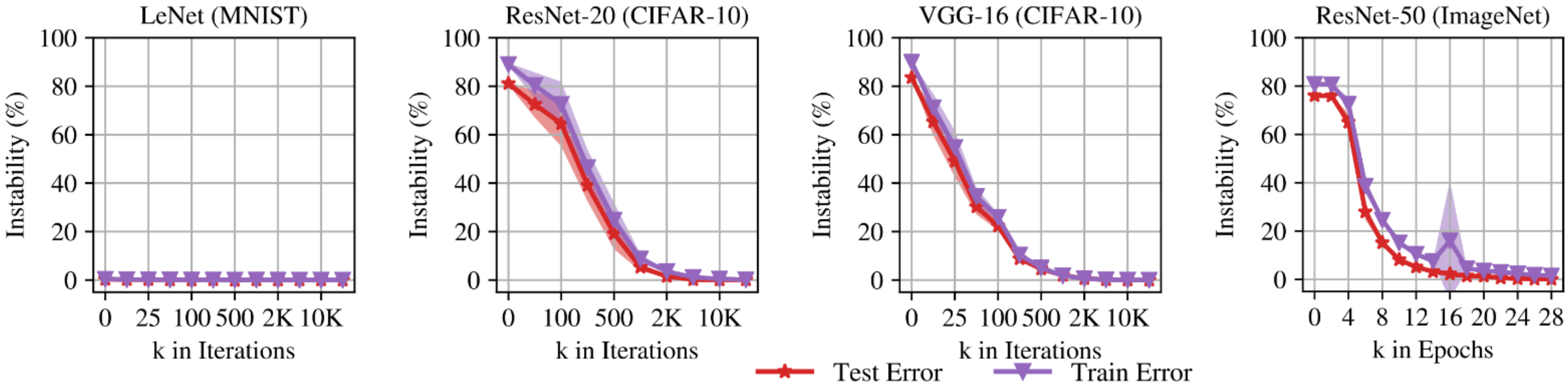


Garipov et al., "Loss Surfaces, Mode Connectivity, and Fast Ensembling of DNNs" NeurIPS 2018

“Linear” mode connectivity



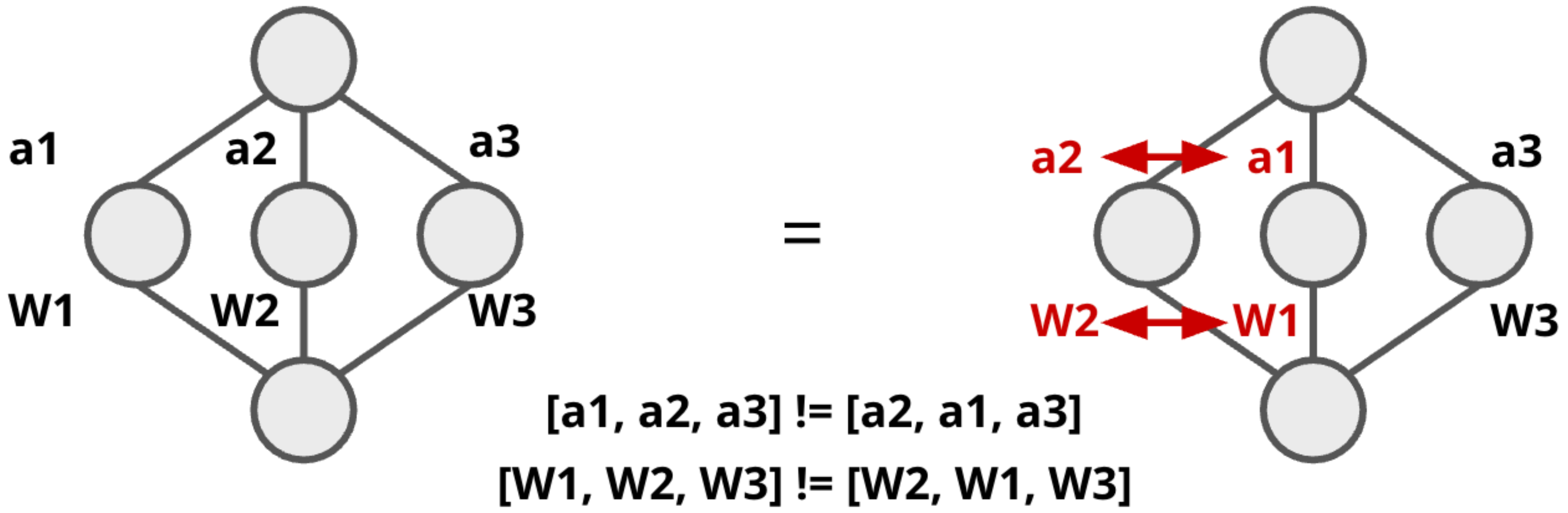
- If two models share the initialization & first k SGD iterations, then the linear interpolation suddenly works
  - i.e., converge to a linearly connected basin
- **Question.** Can we do a similar thing without much shared randomness?



Frankle et al., “Linear Mode Connectivity and the Lottery Ticket Hypothesis,” ICML 2020

Permutation Invariance

- Turned out that **permutation-invariance** of neural nets play a role:
  - If we permute some neurons of a net:
    - Function does not change
    - Parameter does change
  - That is, there are "equivalent params"



Permutation Invariance

- More generally, consider an MLP

$$f_\theta(x) = W_L \sigma(W_{L-1} \sigma(\cdots \sigma(W_1 x) \cdots))$$

- Suppose that we construct another MLP using the same parameters, except

$$\tilde{W}_i = P W_i, \qquad \tilde{W}_{i+1} = W_{i+1} P^\top$$

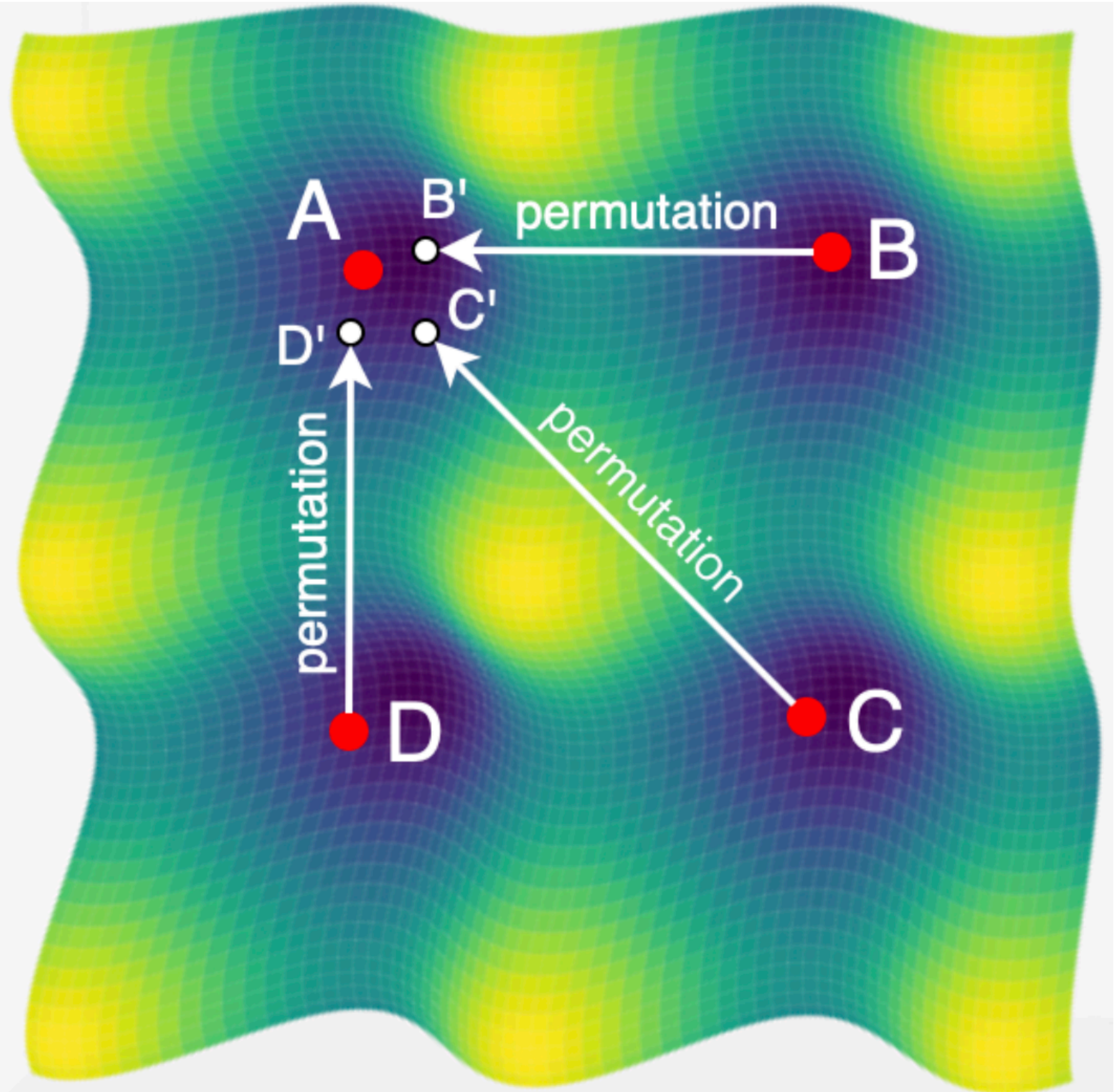
  - Here, P is a permutation matrix
    (binary matrix with only one 1 in each col/row)

- Then, we have

$$f_\theta(x) = f_{\tilde{\theta}}(x)$$

Permutation Invariance

- **Conjecture.** If we permute neurons in a correct way,
  any two modes are linearly connected with each other
  - To merge the knowledge, simply permute & linearly interpolate



Entezari et al., "The Role of Permutation Invariance in Linear Mode Connectivity of Neural Networks," ICLR 2022

Matching the neurons

- **Question.** Given two nets, how can we find the best permutation?
- **Naïve.** Try all permutations, interpolate, find the best one.
  - Challenge. The solution space is too large
    - For a two-layer MLP with d neurons, exists $d!$ permutations

ARCHITECTURE	NUM. PERMUTATION SYMMETRIES
MLP (3 layers, 512 width)	10 ∧ 3498
VGG16	10 ∧ 35160
ResNet50	10 ∧ 55109
Atoms in the observable universe	10 ∧ 82

Ainsworth et al., "Git Re-Basin: Merging Models Modulo Permutation Symmetries," ICLR 2023

Matching the neurons

- Many solutions, but the activation matching is popular
- **Idea.** Match the neurons with the most similar activations
  - Suppose that we have one sample.
    - Let $\mathbf{z}^{(A)}, \mathbf{z}^{(B)} \in \mathbb{R}^d$ be the layer i input activation of model A&B, resp.
    - Solve the ℓ^2 minimization $$\min_P \|\mathbf{z}^{(A)} - P\mathbf{z}^{(B)}\|^2$$
  - If we do extend this multiple samples, becomes equivalent to:

$$\max_P \langle P, \mathbf{Z}^{(A)}(\mathbf{Z}^{(B)})^\top \rangle_F, \qquad \mathbf{Z}^{(A)}, \mathbf{Z}^{(A)} \in \mathbb{R}^{d \times n}$$

Ainsworth et al., "Git Re-Basin: Merging Models Modulo Permutation Symmetries," ICLR 2023

Matching the neurons

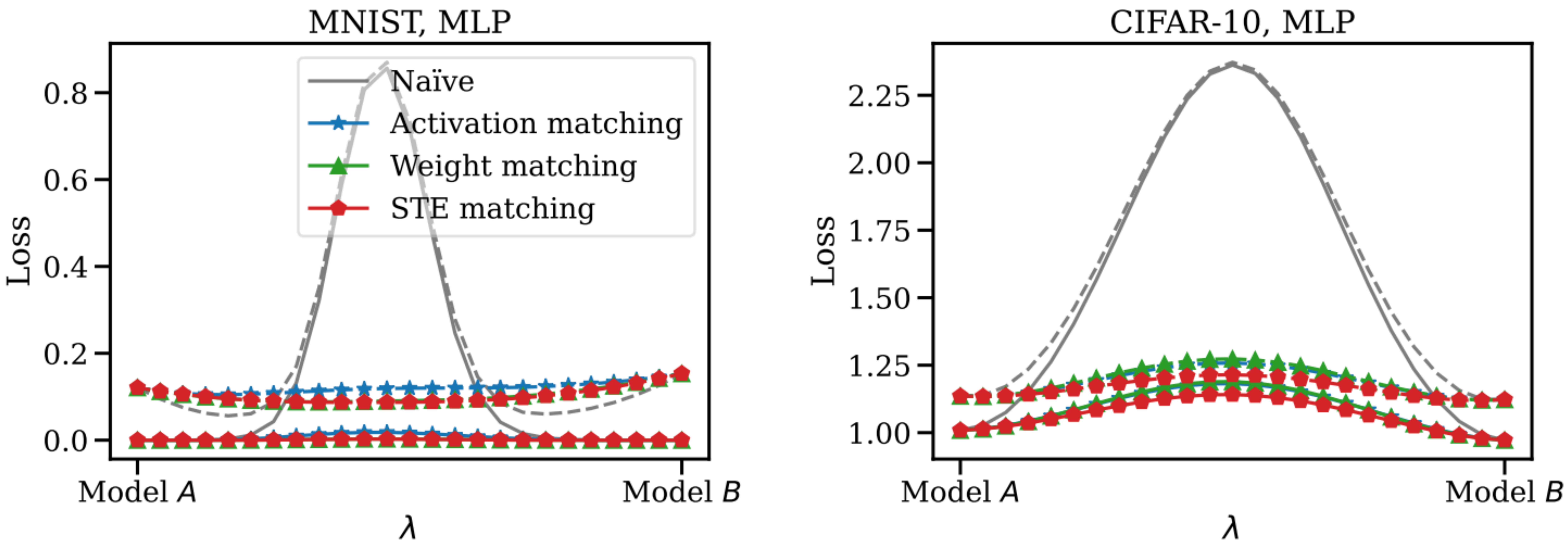
$$\max_P \langle P, \mathbf{Z}^{(A)}(\mathbf{Z}^{(B)})^\top \rangle_F, \qquad \mathbf{Z}^{(A)}, \mathbf{Z}^{(A)} \in \mathbb{R}^{d \times n}$$

- The problem is the linear assignment problem:
    - Place exactly one 1 at row/col, so that the inner prod is maximized:
    - A well-known solver called "Hungarian method":
        - https://en.wikipedia.org/wiki/Hungarian_algorithm

- Solve this, starting from layer 1 to layer L.

Matching the neurons

- The matching-based methods greatly improve interpolated performance
  - STE-based matching works better with models with BatchNorm



Matching the neurons

- Recent works use these techniques to merge models trained on different dataset

- **Promise.**
  - Less inference cost than ensembling
  - No further training cost

- **Limitation.**
  - Still far from the goal

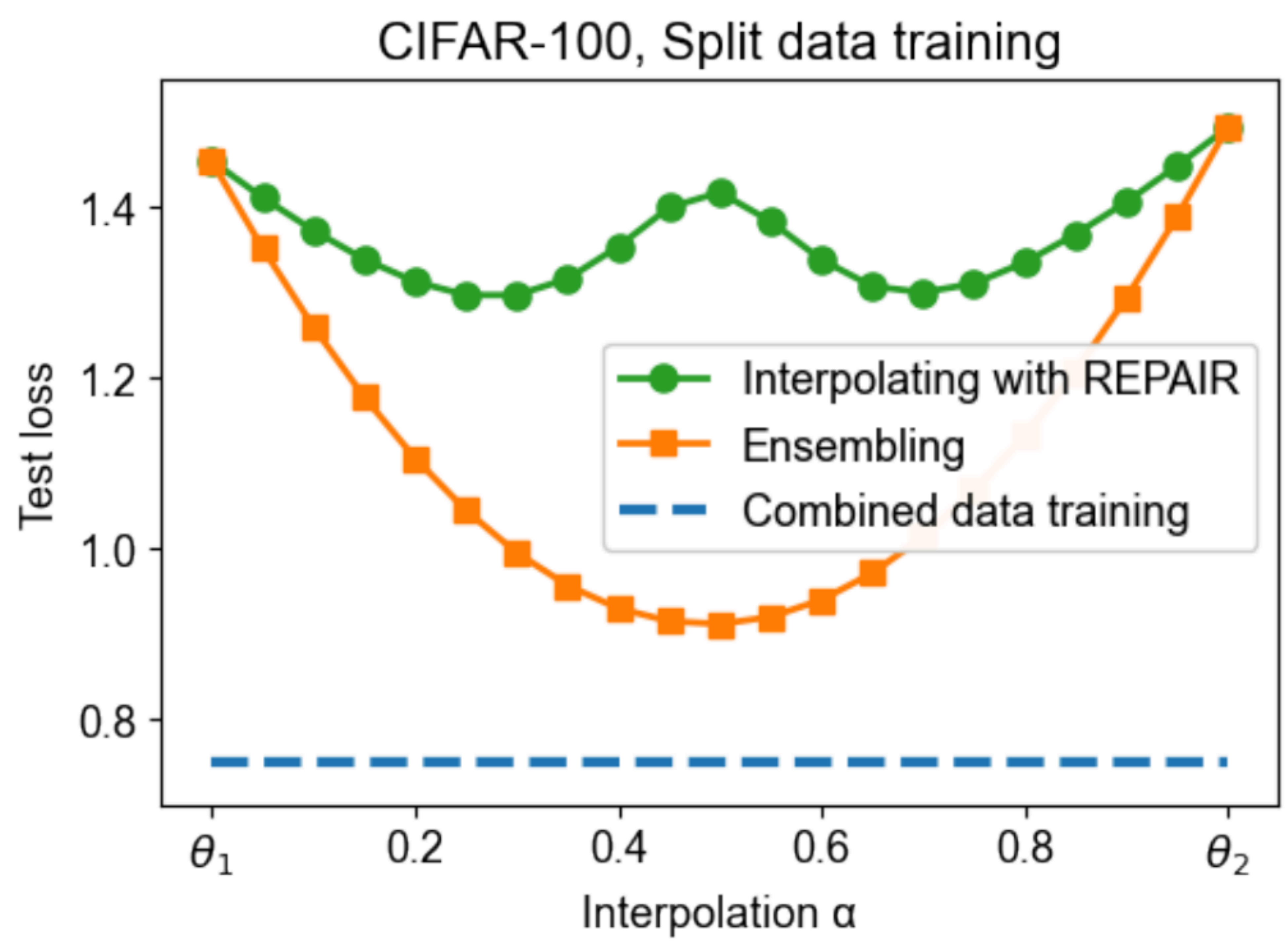
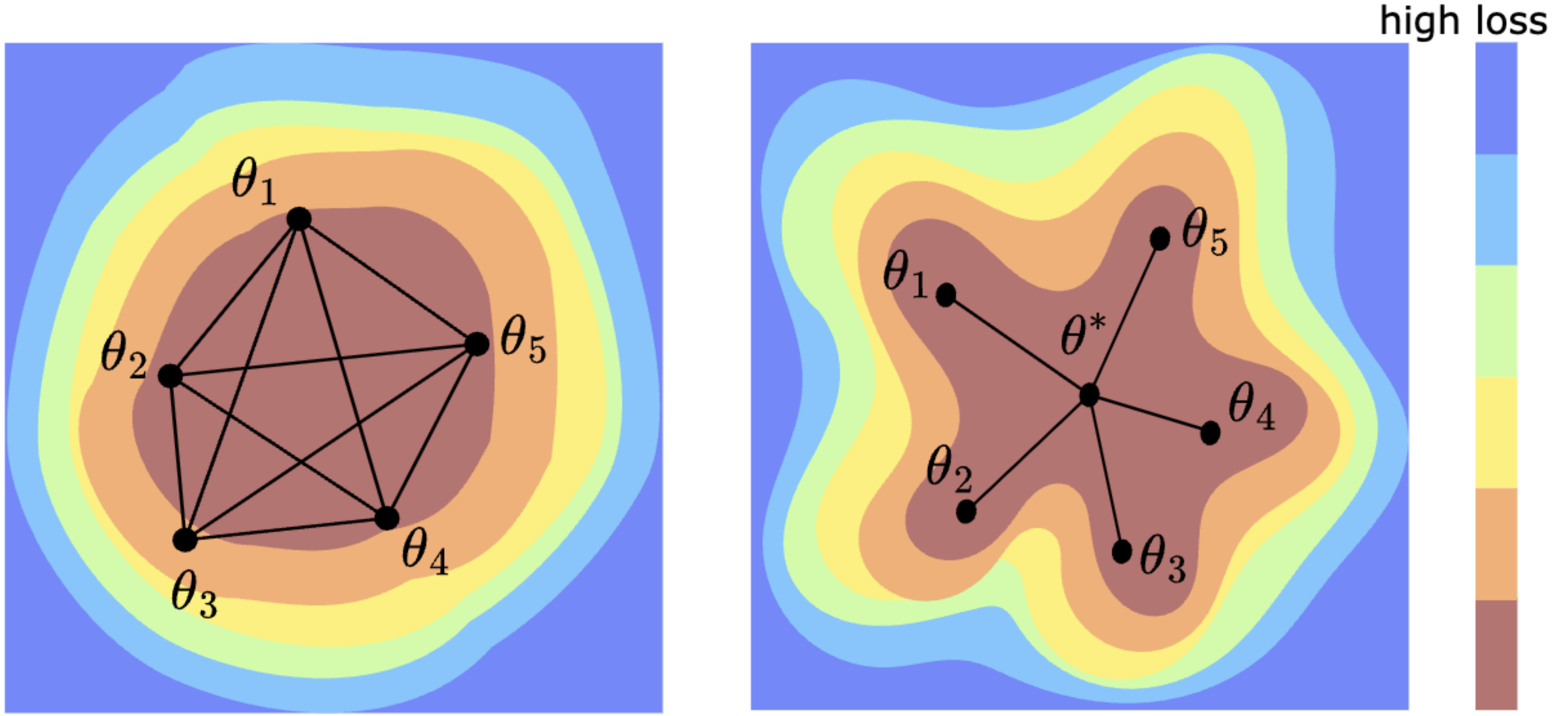


Figure 7: **Split data training.** When two networks are trained on disjoint, biased subsets of CIFAR-100, their REPAIRed interpolations outperform either endpoint with respect to the combined test set.

Jordan et al., "REPAIR: REnormalizing Permuted Activations for Interpolation Repair," ICLR 2023

Recent work: Star conjecture

- Recent work proposes a "star conjecture":
  - Weaker than linear interpolation, stronger than simple mode connectivity



Convexity conjecture (Entezari et al.) holds only for wide networks.

Star domain conjecture (ours) holds even for narrower networks.

Sonthalia et al., "Do Deep Neural Network Solutions Form a Star Domain?," ICLR 2025

Further readings

- **REPAIR.** Fixed for models with BatchNorms
  - https://arxiv.org/abs/2211.08403
- **ZipIt.** Merges only several layers for better performance
  - https://arxiv.org/abs/2305.03053
- **Deep Weight Space Alignment.** Learn to predict the permutation
  - https://arxiv.org/abs/2310.13397
- **Star Domain.** Alternative conjecture
  - https://arxiv.org/abs/2403.07968

Sonthalia et al., "Do Deep Neural Network Solutions Form a Star Domain?," ICLR 2025

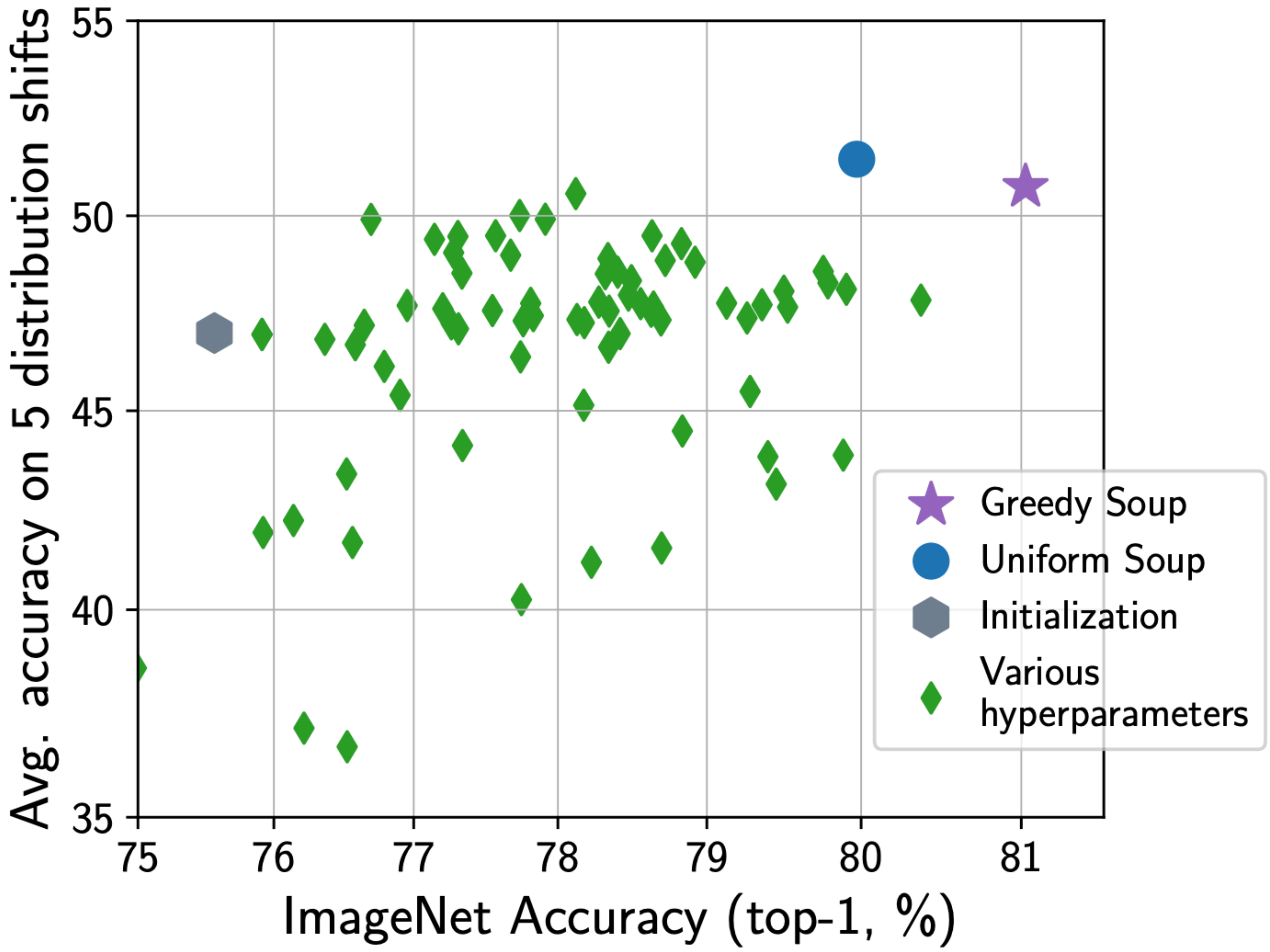
Pretrained model as initialization

Model soup

- **Idea.** Use large pre-trained models as a shared init
    - Generate multiple fine-tuned versions for a target task
        - Diverse hyperparameters
    - Average the fine-tuned weights

$$\theta = \sum_{i=1}^{M} w_i \theta_i$$

- Not as good as ensemble, but cheap



Wortsman et al., "Model soups: averaging weights of multiple fine-tuned models improves accuracy without increasing inference time" ICML 2022

Model soup

- Selecting the nice ingredients is critical
- **Greedy Soup.**
  - Sort each ingredient by validation acc.
  - Add one and taste:
    - If tastes better, keep it
    - Otherwise, remove

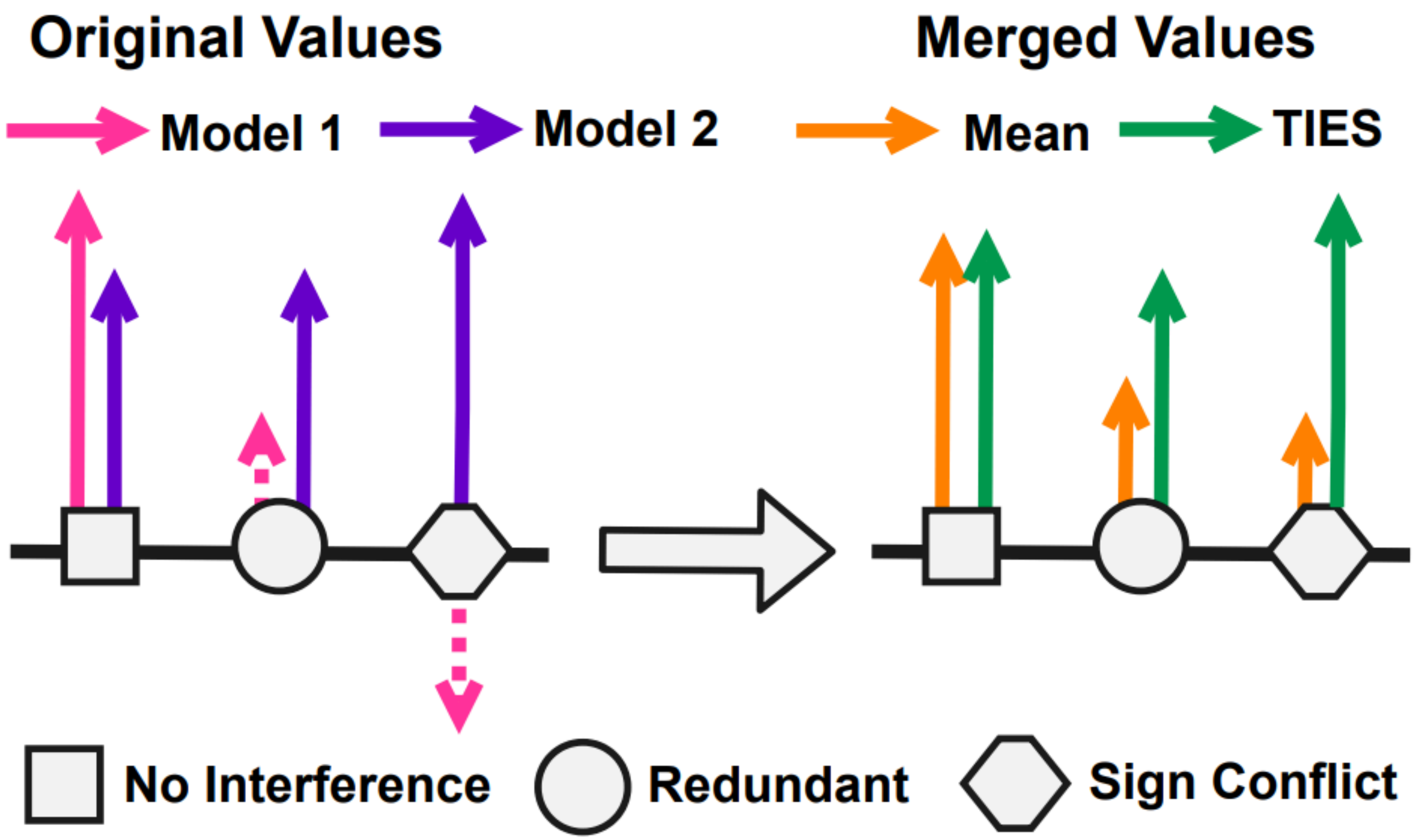
Recipe 1 GreedySoup

Input: Potential soup ingredients $\{\theta_1, ..., \theta_k\}$ (sorted in decreasing order of $\mathsf{ValAcc}(\theta_i)$).
$\mathsf{ingredients} \leftarrow \{\}$
for $i = 1$ **to** k **do**
  if $\mathsf{ValAcc}(\mathsf{average}(\mathsf{ingredients} \cup \{\theta_i\})) \geq$
      $\mathsf{ValAcc}(\mathsf{average}(\mathsf{ingredients}))$ **then**
    $\mathsf{ingredients} \leftarrow \mathsf{ingredients} \cup \{\theta_i\}$
return $\mathsf{average}(\mathsf{ingredients})$

Wortsman et al., “Model soups: averaging weights of multiple fine-tuned models improves accuracy without increasing inference time” ICML 2022

Removing the noise in parameter updates

- Turns out that these parameter updates are quite noisy:
- **TIES-merging (2023).** Resolving conflicts between updates
  - Sign conflict. Ignore a smaller one
  - Redundant update. Ignore small one
  - Others. Average out



Yadav et al., "TIES-MERGING: Resolving Interference When Merging Models" NeurIPS 2023

Further readings

- **DARE.** Pruning-inspire version of TIES
  - https://arxiv.org/abs/2311.03099
- **Model Stock.** Layerwise merging & smaller ingredients for soup
  - https://arxiv.org/abs/2403.19522
- **Evolutionary Optimization.** Use EO to find the best weight combination
  - https://arxiv.org/abs/2403.13187

Editing

Motivation

- The knowledge of neural nets is not perfect
  - Factual mistake
  - Outdated information
  - Cannot access certain info

Who teaches EECE695D-01 at POSTECH right now?

As of March 24, 2025, the instructor for EECE695D-01 at POSTECH is not publicly listed. For the most accurate and current information, I recommend checking POSTECH's official course catalog or contacting the Electrical Engineering department directly.

Motivation

- **Option#1.** Retrain the model from scratch, with original dataset + patch data
    - Too Costly!
- **Option#2.** Fine-tune with patch
    - Costly, and can affect other predictions
- **Option#3.** Retrieval-augmented generation
    - Good, but sometimes conflict with the original model

Goal

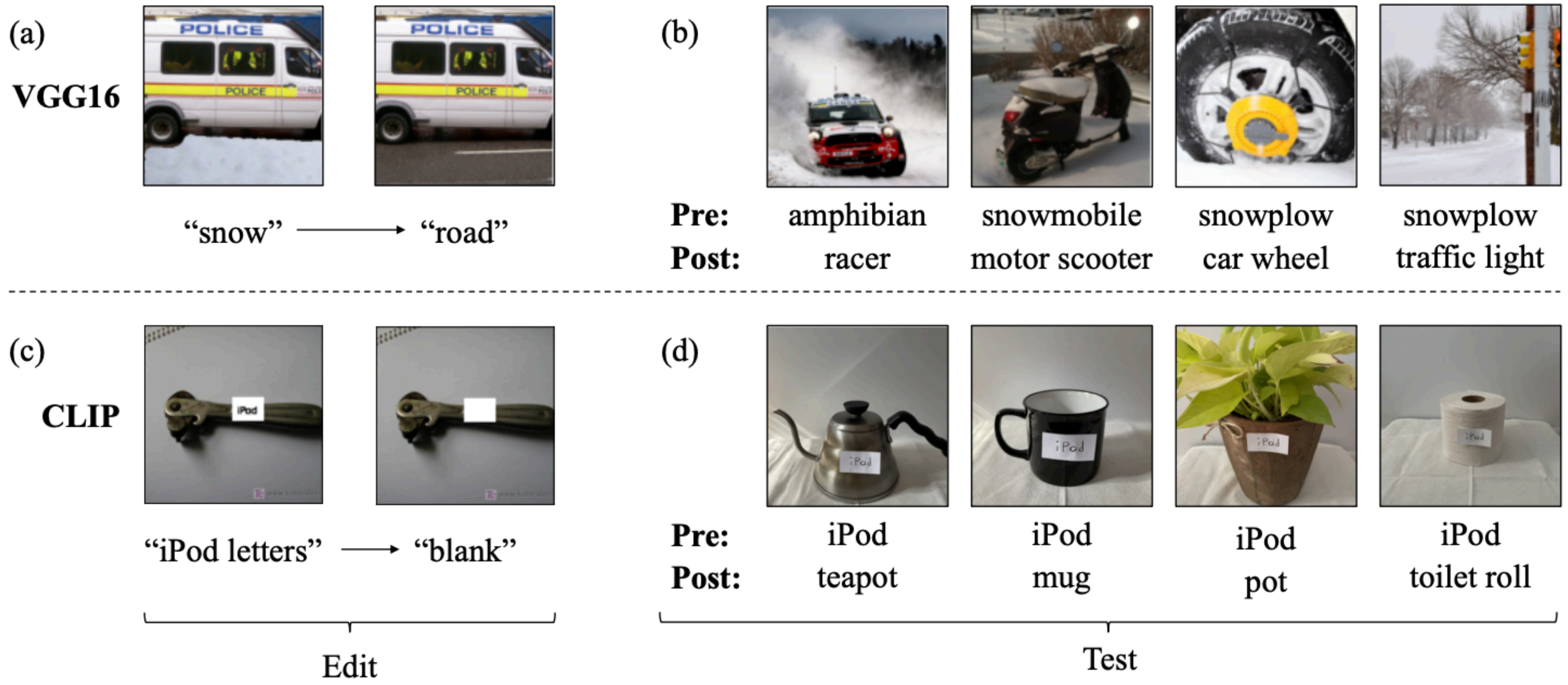
- Given a model $f_\theta(\,\cdot\,)$, modify the prediction on a sample $\mathbf{x}^*$ to be y^*
- We want to find $\tilde{\theta}$ such that:
  - **Reliable.** Makes desired changes $f_{\tilde{\theta}}(\mathbf{x}^*) \approx y^*$
    (e.g., “who’s the president of United States?”)
  - **Local.** Minimally affects unrelated info $f_{\tilde{\theta}}(\mathbf{x}) \approx f_\theta(\mathbf{x}), \quad \mathbf{x} \neq \mathbf{x}^*$
    (e.g., “which team does Messi play for?”)
  - **Generalizes.** Corrects output for related input $f_{\tilde{\theta}}(\mathbf{x}) \approx y^*, \quad \mathbf{x} \approx \mathbf{x}^*$
    (e.g., “who’s the US president?”)
  - Plus, we want to minimize the computational cost of doing so

Approaches

- Many approaches:
    - Partial Retraining
    - Meta-Learning
    - Task Arithmetics

Partial Retraining

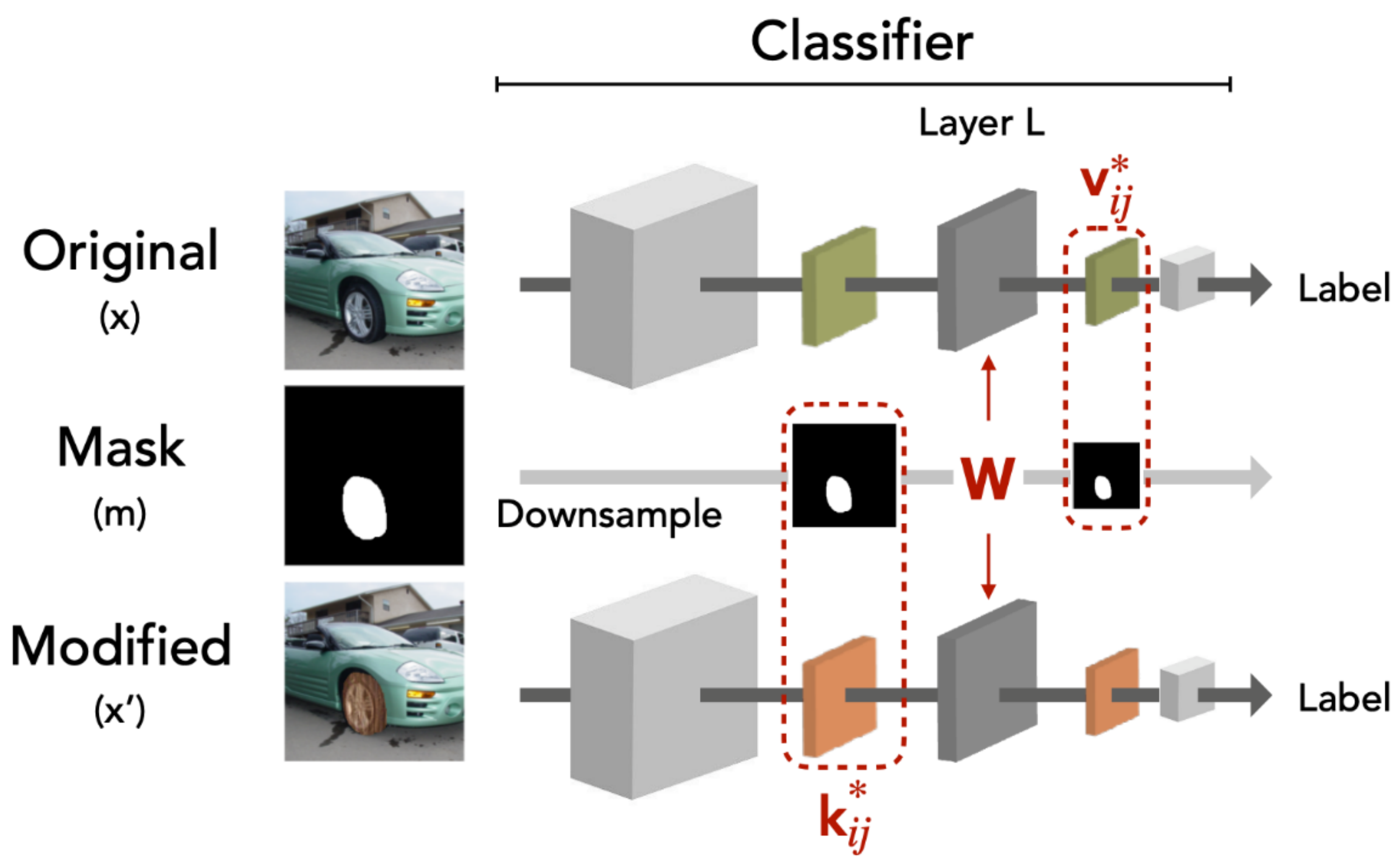
- Retrain only one (or few) layers
- We study the example of Santurkar et al., (2021)
  - Given a single pair of exemplar, edit prediction rules to equate them
    - e.g., replace certain concepts / robustness to attacks



Santurkar et al., "Editing a Classifier by Rewriting Its Prediction Rules," NeurIPS 2021

Partial Retraining

- Update the layer i as follows:
  - **Input.** Layer (i-1) activation of a model that sees modified input (called "keys" $k^* \in \mathbb{R}^m$)
  - **Output.** Layer i activation of a model that sees the original input (called "values" $v^* \in \mathbb{R}^n$)



(a) Edit


(b) Test

Partial Retraining

- Find a matrix W' which solves

$$W' = \arg\min \|V - WK\|^2, \qquad \text{subject to} \quad v* = W'k*$$

  - V, K are values/keys for unmodified locations

- This is a least-squares with constraints, with solution expressed as:

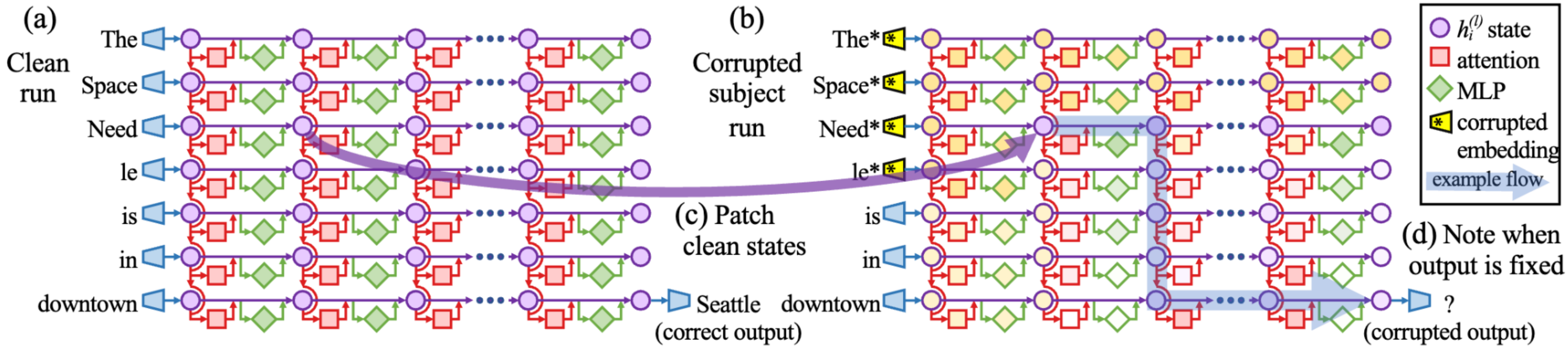
$$W' = W + \Lambda(KK^\top)^{-1}k*)^\top$$

  - Λ can be found by gradient descent
  - For updating a single concept, rank-1 update is enough!

Bau et al., "Rewriting a Deep Generative Model,"ECCV 2020

Further ideas

- This approach requires pinpointing the which-tensor-to-update:
- **Idea.** Causality-based analysis (will not go into details)
  - i.e., corrupt-and-restore several tokens, and trace the corruptions
  - e.g., ROME (https://arxiv.org/abs/2202.05262)
    MEMIT (https://arxiv.org/abs/2210.07229)



Meta-Learning

- Train a model editor which maps

“editing task” → weight updates

  - Super-fast editing
  - **Problem.** “Editing task” is difficult to formalize as a model input

- We study the example of Mitchell et al., (2022)

Mitchell et al., “Fast model editing at scale” ICLR 2022

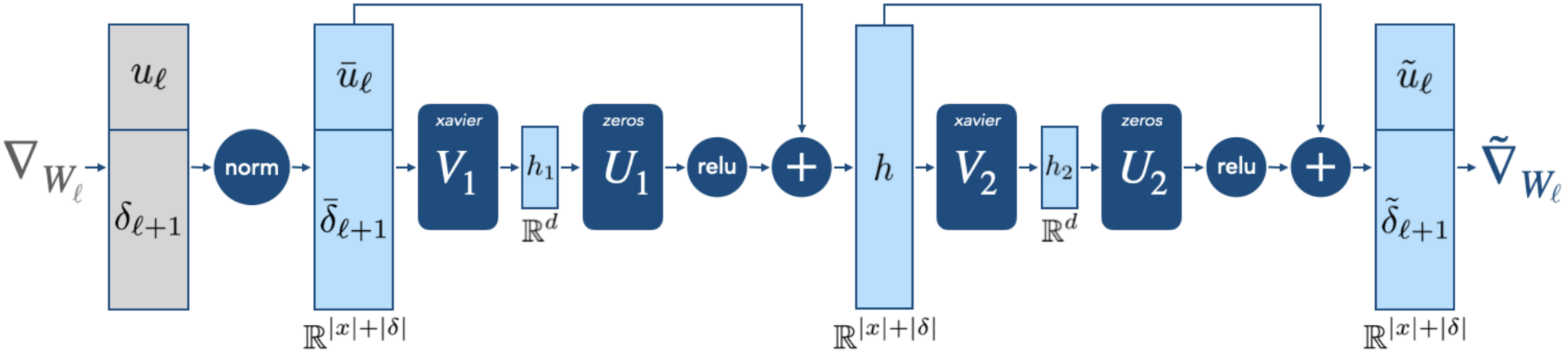
Meta-Learning

- **Idea.** Train a model that uses the loss gradient as an input, and the actual update as an output
  - Train separate predictors for each tensor (Reduced computational cost)

Mitchell et al., "Fast model editing at scale" ICLR 2022

Meta-Learning

- **Trick.** Weight gradients for each sample are rank-1
  - Linear model: $\nabla_{\mathbf{W}} ||\mathbf{y} - \mathbf{W}\mathbf{x}||^2 = 2(\mathbf{y} - \mathbf{W}\mathbf{x})\mathbf{x}^\top$
  - Deeper model: (Handle similarly)
    - Thus, predict from/to concatenated rank-1 vectors



Mitchell et al., "Fast model editing at scale" ICLR 2022

Meta-Learning

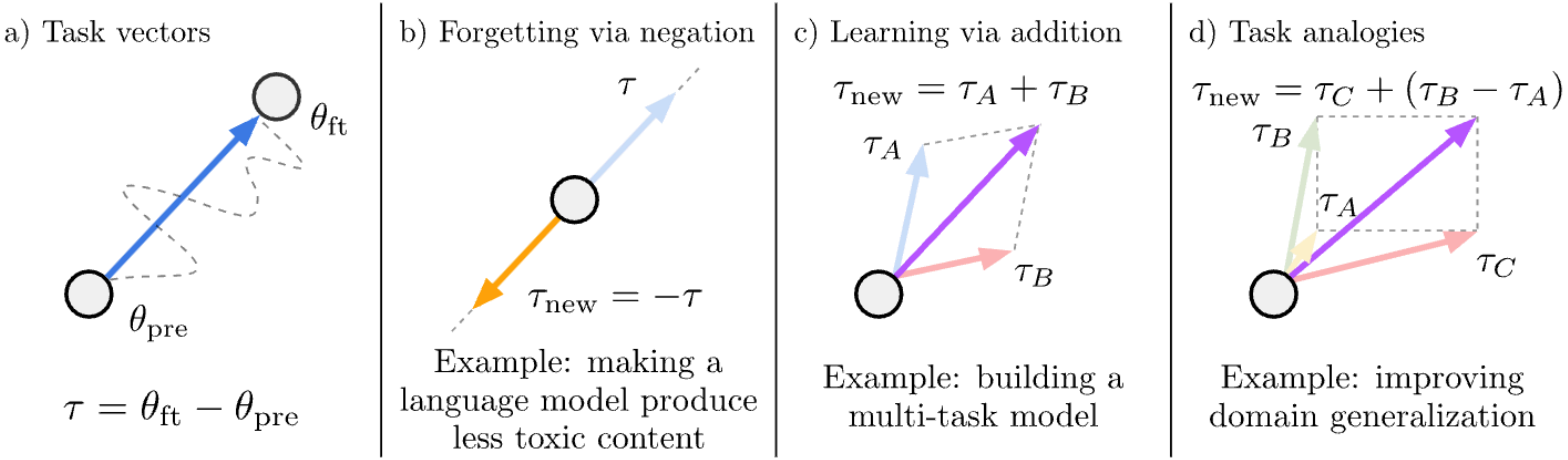
- **Meta-Training.**
  - At each step, sample:
    - Edit sample $(\mathbf{x}_e, y_e)$
    - Equivalence sample $(\mathbf{x}'_e, y'_e)$
      - Generated by removing some prefix tokens from edit
    - Locality example $\mathbf{x}_{\text{loc}}$
  - Then, train with the joint loss

$$\textbf{MEND losses:} \quad L_{\text{e}} = -\log p_{\theta_{\tilde{\mathcal{W}}}}(y'_{\text{e}}|x'_{\text{e}}), \quad L_{\text{loc}} = \text{KL}(p_{\theta_{\mathcal{W}}}(\cdot|x_{\text{loc}}) \| p_{\theta_{\tilde{\mathcal{W}}}}(\cdot|x_{\text{loc}})). \quad \text{(4a,b)}$$

Mitchell et al., “Fast model editing at scale” ICLR 2022

Task arithmetics

- Suppose that we have a large pre-trained base model
  - Then, we can do arithmetics with task-specific fine-tuned weight updates
    - Add knowledge: Fine-tune and add
    - Remove knowledge: Fine-tune and subtract



Ilharco et al., "Editing models with task arithmetic" ICLR 2023

Challenges

- Scaling up to trillion-scale models
- Editing black-box models:
  - https://arxiv.org/abs/2211.03318
- Applying massive edits in parallel
- Transferring edits from model to model

That’s it for today 🙌